МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-08

Дипломний проект

здобувачки освіти денної форми навчання
РП.08.24.000.ДП

ФРОЮК
ДАРИНИ МАКСИМІВНИ

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітньо-професійна програма: **«Розробка програмного забезпечення»**
Група: **4РП-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності

Проектний матеріал складається з пояснювальної записки на 87 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Фроюк Д.М.)
Керівник _____ (Жадан А.С.)

Консультанти:
з економічного розділу _____ (Канський М.Ю.)
з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)
з нормоконтролю _____ (Петрашова В.І.)
старший консультант _____ (Кривченко Ю.В.)

До захисту допущений
Голова циклової комісії _____ (Кривченко Ю.В.)
Завідувач відділення _____ (Краснокутська К.Г.)

Захист «30» 06 2025 р. Протокол ЕК № 3
Оцінка ЕК 5/95

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"12" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачці освіти *Фроюк Дарині Максимівні*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності.*

затверджена наказом по коледжу від "*14*" *листопада* 2024р. № *246*

2. Термін здачі закінченого проекту

3. Вихідні данні до проекту

1. Використати ігровий рушій Unity;

2. Застосувати design-pattern Singleton;

3. Реалізувати базові механіки пересування гравця і ворога;

4. Використати json-файли для зберігання інформації.

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

1. Огляд ознак ігрового жанру horror-survival; 2. Створення 3D-сцени оточення гри;

3. Написання основної програмної логіки гри на мові програмування високого рівня C#;

4. Створення ігрового графічного інтерфейсу користувача гри;

5. Економічний розрахунок. 6. Аспекти охорони праці та техніки безпеки.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Титул; Використані ознаки жанру Horror-Survival у проєкті; Ознаки жанру гри; Етапи розробки гри; Схема рхітектури гри; Схема структри меню; гри; Схема методу UpdateGameInformation(); Вигляд гравця, ворога та іграшки; Вигляд лісу з камери гравця; Створені локації у лісі; Тестування гри; Висновки.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Жадан А.С.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.2025

Керівник *Жадан А.С.* _____
(підпис)

Завдання прийня до виконання *Фроюк Д.М.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Формування вступу*	15.05.25	виконано
2	*Дослідження предметної області*	16.05.25	виконано
3	*Огляд аналогів*	20.05.25	виконано
4	*Вибір технічної літератури*	22.05.25	виконано
5	*Аналіз технологій розробки*	23.05.25	виконано
6	*Проєктування ігрового застосунку*	28.05.25	виконано
7	*Розробка ігрового застосунку*	30.05.25	виконано
8	*Тестування створеного ігрового застосунку*	05.06.25	виконано
9	*Оформлення пояснювальної записки*	09.06.25	виконано
10	*Підготовка графічних матеріалів*	11.06.25	виконано
11	*Економічний розрахунок*	12.06.25	виконано
12	*Опис аспектів охорони праці та техніки безпеки*	13.06.25	виконано
13	*Підведення висновків*	13.06.25	виконано
14	*Підготовка доповіді для захисту*	16.06.25	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 08. 24 000. 00 ДП	Дипломний проект		
А4			РП. 08. 24 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 08. 24 000. 00 ДП		
Розробив		Фроюк Д.М.		16.06	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів	Літ.	Аркуш 4 / Аркушів 87
Перевірив		Жадан А.С.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06		ВСП "ОТФК ОНТУ" гр.4РП-08	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

Ігрова індустрія на сьогоднішній день це дуже потужна сила, яка впливає на мільйони людей. Ігри здатні викликати емоції у людей і занурювати їх у вигадані світи, дозволяючи переживати унікальні історії, відчувати напругу, радість, страх або захоплення, а також співпереживати персонажам і приймати складні моральні рішення. Але ігри це не лише про розважальний контент, за допомогою ігор можна навчати, мотивувати, об’єднувати людей навколо спільних ідей або цінностей, формувати світогляд, доносити інформацію, тощо.

В Україні ця сфера досі розвивається, хоча вже є приклади успішних проєктів таких як серія ігор Stalker, Metro, Cossacks 3, Sherlock Holmes series, тощо. Тому треба розширювати асортимент українських ігор, пробувати різні жанри, вивчати різні інструменти і тендендеції для подальшої розробки проєктів, які будуть цікаві людям.

Мета розробки цієї гри дослідження такого жанру як horror-survival і вивчення методів і інструментів для розробки ігор в цілому. Проєкт дозволяє ознайомитись з етапами розробки ігор: від ідеї і формування концепції механік, до створення візуального стилю, написання скриптів і їх тестування.

Для досягнення мети був обраний ігровий рушій Unity[1], який має низький поріг входу, багато навчальних матеріалів, прикладів і доволі велику спільноту, тому для першої гри Unity – це ідеальний інструмент. Також цей рушій підтримує розробку під різні пристрої. У процесі створення гри використовуються різні асети по типу NavMesh, який забезпечує можливість навігації персонажа по ігровому середовищу шляхом прокладання маршрутів а допомогою ШІ, і 3D-модельки з asset store для створення сцени. Застосовуються інші пакети для роботи зі звуком, анімацією, освітленням та фізикою, що пришвидшує розробку подібних проєктів. Також при розробці виникали деякі проблеми, які зміг вирішити паттерн програмування Singletone[2].

Результатом розробки є гра в жанрі horror-survival, в якій реалізовані базові механіки такі як патрулювання ворога, переслідування гравця якщо його виявлено, а також взаємодія з об'єктами оточення. Гру можна запускати на різних пристроях,

зокрема на персональних комп’ютерах, а за потреби – адаптувати для мобільних платформ. Проєкт може слугувати навчальним прикладом для початківців у сфері розробки ігор, а також базою для подальшого вдосконалення та розширення функціоналу, вивчення нових технологій і експериментів у межах жанру.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд жанру horror-survival

1.1.1 Головні ознаки жанру

Horror-survival[3] являє собою жанр гри, який включає у себе елементи жаху і виживання. Вперше такий термін використали у 1996 для релізу гри Resident Evil, яка заклала основи жанру.

Основною метою ігор у жанрі horror-survival є виживання у ворожому, небезпечному середовищі. Гравець вразливий і не має повного контролю над ситуацією у грі. Гравцю може надаватись зброя і всілякі ресурси для виживання і протистояння ворогам, але в обмеженій кількості, або вони можуть бути взагалі відсутніми. Гравець має досить обмежені фізичні можливості – зазвичай він може лише йти, бігти й підніматися на невеликі підвищення. Таким чином гравець не може довго протидіяти ворогам і йому доводиться ухилятись, вирішувати головоломки, ховатись, тікати, тощо для просування по сюжету або проходженню гри.

Часто метою гри є збирання речей. Гравець знаходиться у закритій локації, де на нього полює монстр. Для проходження гри треба зібрати всі речі і не бути зловленим ворогом. У разі програшу все треба починати спочатку. Також може бути присутнім сюжет і деякі головоломки.

Важливою складовою horror-survival є атмосфера, яка викликає у гравця страх, відчуття напруги і тривожності. Для її створення важливі візуальні і звукові ефекти.

У локаціях використовують приглушене тускле світло, туман, дощ, або події відбуваються вночі. Такі умови погіршують навігацію гравця і змушують його бути більш обережними. Часто локації розробляються темними та клаустрофобними, із вузькими проходами, обмеженими ракурсами камери та тінями. Хоча ігри цього жанру також використовують величезні пусті простори.

Звукові ефекти створюють відчуття занурення у гру і можуть бути повноцінним елементом геймплею, коли для проходження треба ще й

прислухатись. Звуками позначають наближення ворога, вставляють у джампскейри або просто обманюють гравця, щоб викликати тривогу і змусити боятися кожного шороху. У джампскейрах звук це найважливіша деталь, або саме різкий, гучний звук у поєднанні з неочікуваною візуальною появою монстра викликає сильну емоційну реакцію – здивування або переляк. Без звуку ефект несподіванки значно слабший.

1.1.2 Огляд аналогів

Eyes the horror game – безкоштовна гра жахів, розроблена на рушії Unity 3D.

Гравець – це злодій, який прокрався в одну з локацій(особняк, лікарня, школа) і повинен зібрати 6/12/20/30, залежно від вибраного рівня складності, мішків з грошима. Під час цього гравця буде переслідувати один змонстрі монстр: Крейсі, Чарлі або Дружок. Гравець може ховатись у кімнатах, до яких монстри не мають доступу.

Кожна локація має свої особливості. До прикладу в школі під час перерв двері всіх кімнати в будівлі відкриваються і гравець стає дуже уразливим в цей час доки перерва не закінчиться, а двері не закриються.

Гра має чотири рівні складності. З кожним рівнем збільшується кількість мішків і швидкість монстрів. На рисунку 1.1 зображено постер гри “Eyes the horror game”.

Рисунок 1.1. Постер гри “Eyes the horror game”

Outlast[4] – комп'ютерна відеогра в жанрі survival horror від першої особи, розроблена й випущена канадською компанією Red Barrels на платформу

Microsoft Windows у 2013 році. Пізніше були випущені версії гри для macOS, Linux, PlayStation 4, Xbox One й Nintendo Switch. Події в Outlast відбуваються в покинутій психіатричній лікарні, де герой-журналіст стикається з жорстоко налаштованими божевільними. Протагоніст не має зброї й змушений тікати та ховатися від ворогів, що переслідують його.

Гравці можуть ховатися в шафках і під ліжками, або зачиняти двері перед ворогами. Однак вороги шукають їх і можуть протиснутись через невеликі простори, тому потрібно бути обережним.

Єдиний предмет, який має гравці, це відеокамера. Відеокамера працює від батареї та має систему нічного бачення, яка використовується для навігації в темних місцях. Батарейки потрібно збирати протягом усієї гри, щоб нічне бачення відеокамери працювало. На рисунку 1.2 зображено постер даної гри.

Рисунок 1.2. Постер гри «Outlast»

Amnesia: The Dark Descent[5] – гра в жанрі survival horror, випущена студією Frictional Games у 2010 році. Гравець грає за Даніеля, який прокидається в темному замку без пам'яті про своє минуле. Йому потрібно знайти барона Олександра Бренненбурга,. Поступово гравець розкриває секрети, які приховує замок, та долю самого Даніеля.

Гра не передбачає прямого бою з ворогами, вона фокусується на стелс-елементах і виживанні. Гравець змушений уникати ворогів, ховатися або

використовувати різні механізми. Однією з основних особливостей є система психічного стану персонажа: якщо він перебуває в темряві, дивиться на монстрів його психічний стан погіршується. Це проявляється в зміненому зорі та слухових галюцинаціях. Щоб зберегти розум, гравець повинен використовувати джерела світла, наприклад, лампи або смолоскипи.

У процесі гри також зустрічаються головоломки. Гравець повинен збирати предмети, відкривати двері, активувати механізми й розгадувати загадки. Інвентар обмежений, що створює додаткову складність, оскільки ресурси швидко закінчуються і потрібно обережно їх використовувати. На рисунку 1.3 зображено постер гри “ Amnesia ”.

Рисунок 1.3. Постер гри “Amnesia”

1.2 Створення 3D-сцени

1.2.1 Пошук асетів.

Для гри була вирішено створювати локацію страшного лісу і деякі міні-локації у ньому, а пошук асетів проводився в unity asset store. Для створення локації були підібрані наступні асети:

1. Environmental Asset Pack (див.1.4):

- Префаби дерев Tree 1, 2, 3, 4;
- Текстура ґрунту Grass3;

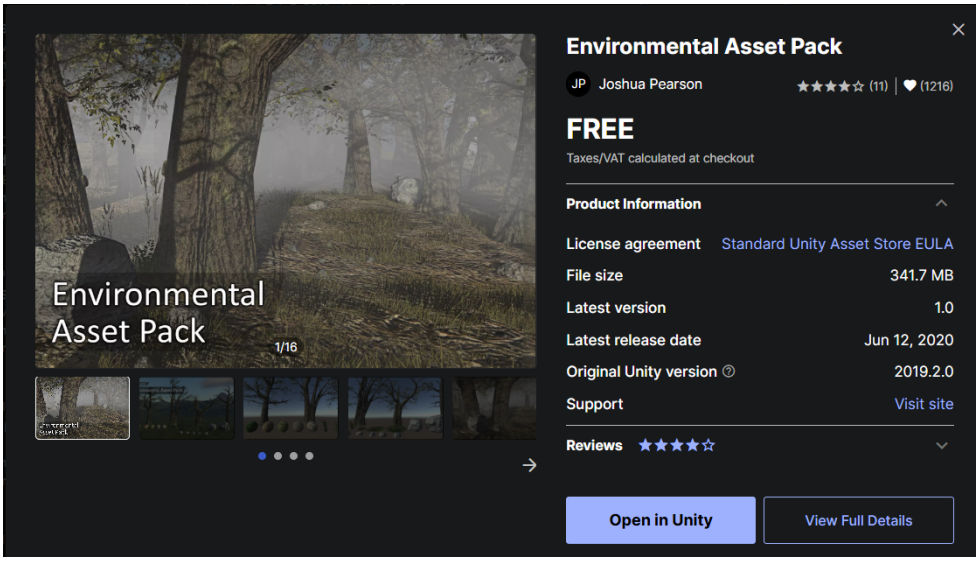


Рисунок 1.4. Асет Environmental Asset Pack

2. Grass And Flowers Pack (див. 1.5):

- Текстура трави Grass2.

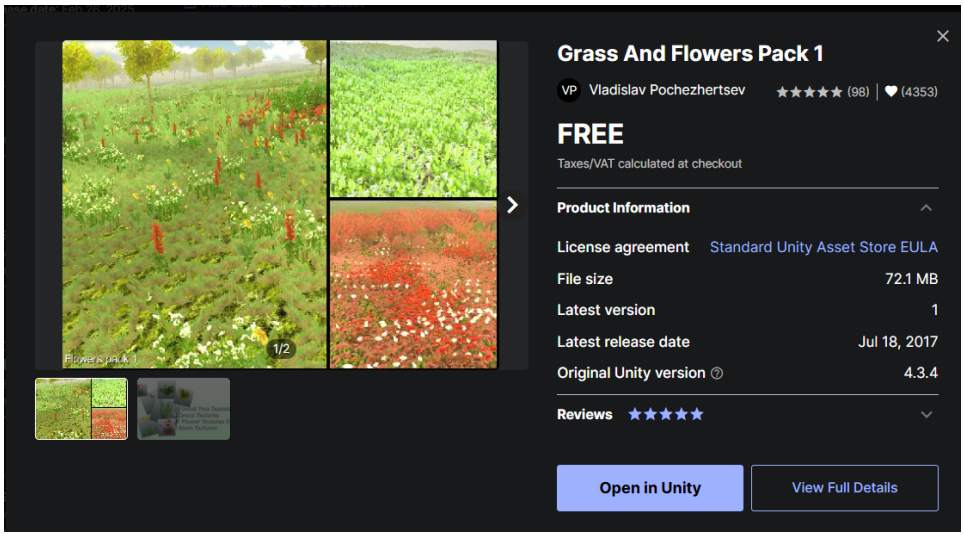


Рисунок 1.5. Пак текстур Grass And Flowers Pack

3. Cathedral and Cemetery Kit (див. 1.6):

- Хрести: Cross 1, 2, 3, 6;
- Надгробки: Grave 5, 6, 10;
- Каплиця: Chapel_Part_1A;
- Сходи: Steps_1A;
- Паркан: Fence_1A, Fence_1C, Fence_1F, Fence_Pillar_B;
- Ворота: Fence_Gate_1_R і Fence_Gate_1_L;
- Домовина: Coffin Closed.

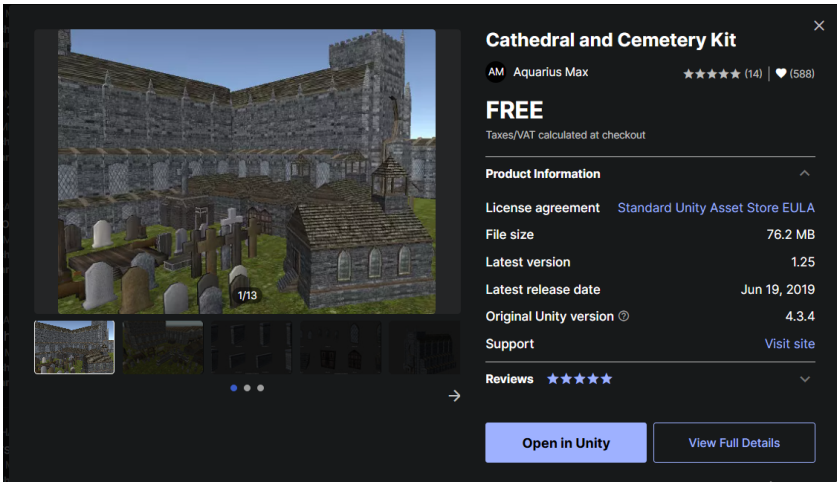


Рисунок 1.6. Асет Cathedral and Cemetery Kit

4. Traditional Water Well (див. 1.7):

- Модель колодязя.

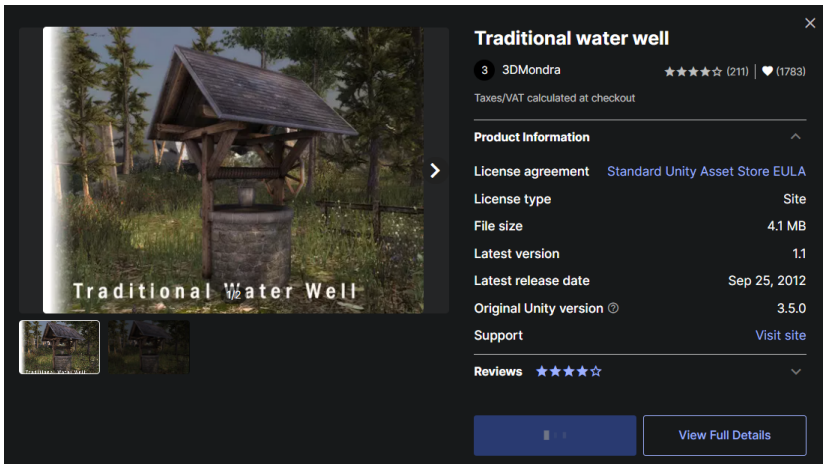


Рисунок 1.7. Асет Traditional Water Well

5. Wood Bucket: Standard + Dirty (див. 1.8):

- Дерев’яні відра.

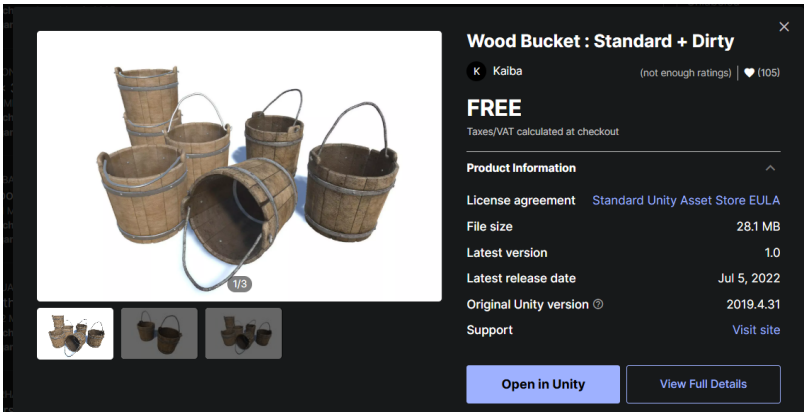


Рисунок 1.8. Асет Wood Bucket: Standard + Dirty

6. Cursed Toy: Shadowlop (див. 1.9):

- Модель ворожого NPC з лякаючим виглядом та анімаціями

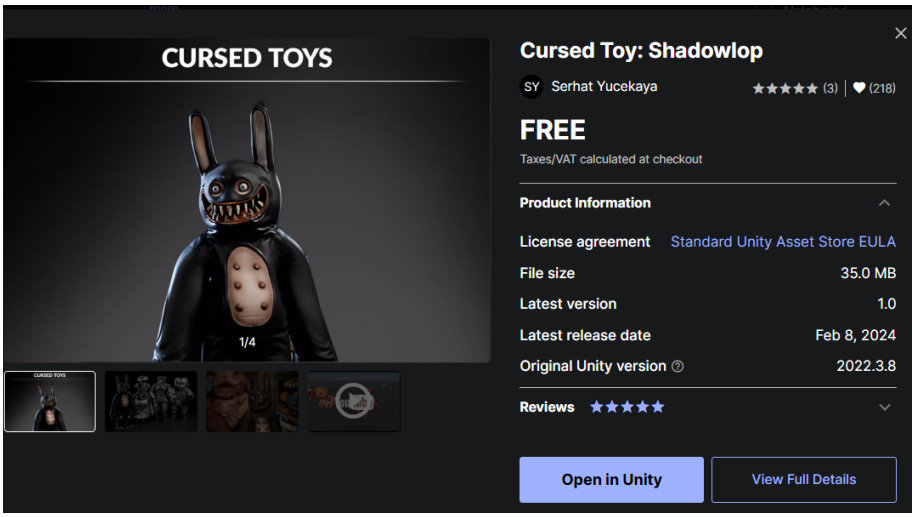


Рисунок 1.9. Асет Cursed Toy: Shadowlop

7. 6 x 3D Cute Toy Models (див. 1.10):

- Модель м’якої іграшки кролика Rabbit.

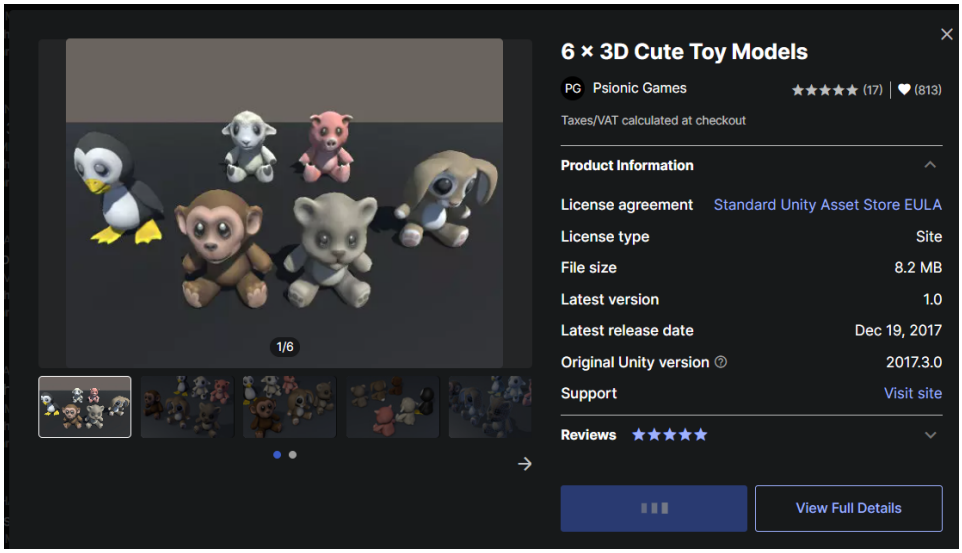


Рисунок 1.10. Асет 6 x 3D Cute Toy Models

1.2.2 Налаштування Terrain

Для створення ігрової площі використовується 3-D об’єкт Terrain[6], тому що на ньому можна створити реалістичний рельєф поверхні, а також від має всі потрібні інструменти для налаштування дерев, трави, тектури поверхні, тощо.

У Terrain Settings були задані розміри поверхні 248*248 і висота 600 (див рис.1.11).

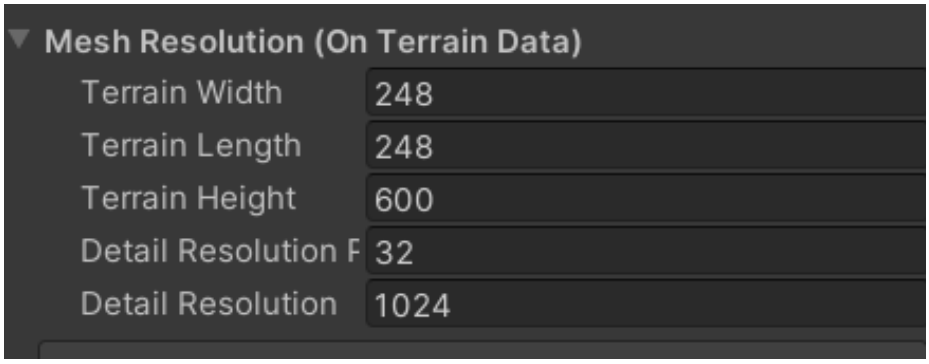


Рисунок 1.11. Налаштування розмірів Terrain

У вкладці Paint Terrain обирається інструмент Raise and Lower Terrain. Його основна функція полягає в піднятті або опусканні частини поверхні ландшафту шляхом малювання пензлем по Terrain. В Unity є доступний набір пензлів за замовчуванню. Для створення рельєфу був обраний пензель builtin_brush_6 з налаштуваннями Brush size - 320 і Opacity – 3.5. На рис. 1.12 наведено результат і налаштування.

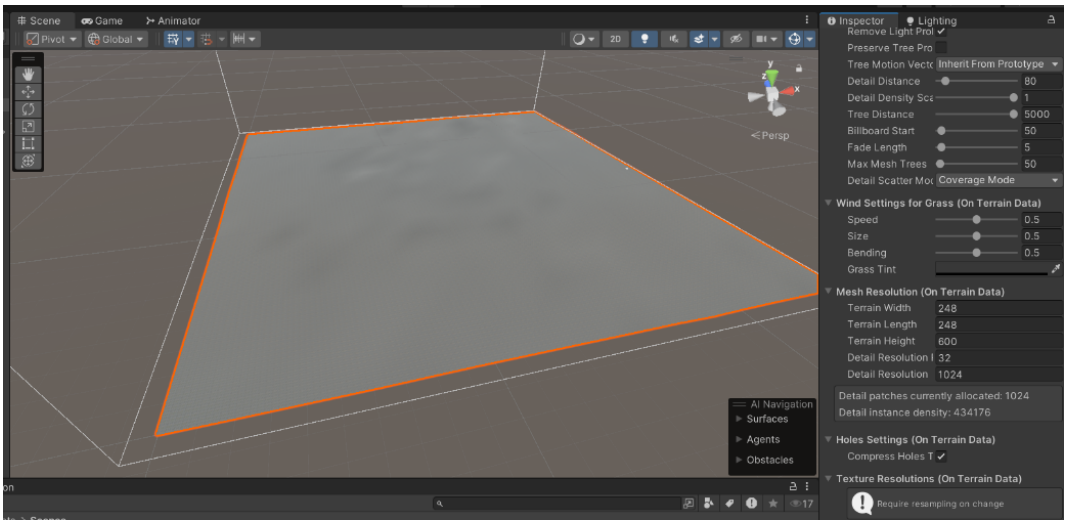


Рисунок 1.12. Вигляд terrain з доданим рельєфом

Після створення рельєфу поверхні треба надати більш реалістичного виду. Для цього використовується інструмент інтрумент Paint texture у тій же вкладці Paint Terrain. Він дозволяє накладати текстури на різні ділянки Terrain, створюючи природне покриття: траву, ґрунт, каміння тощо. Щоб додати тектуру треба натиснути на Edit Terrain Layers, потім обрати Create Layer і обрати потрібний файл, щоб додати текстуру для малювання. Для створення покриття з асету JP Environmental Asset Pack була використана текстура трави Grass 3 У

налаштуваннях Tiling Settings для параметрів Size по осях X та Y було встановлено значення 2. На рис. 1.13 наведені налаштування для Paint Texture.

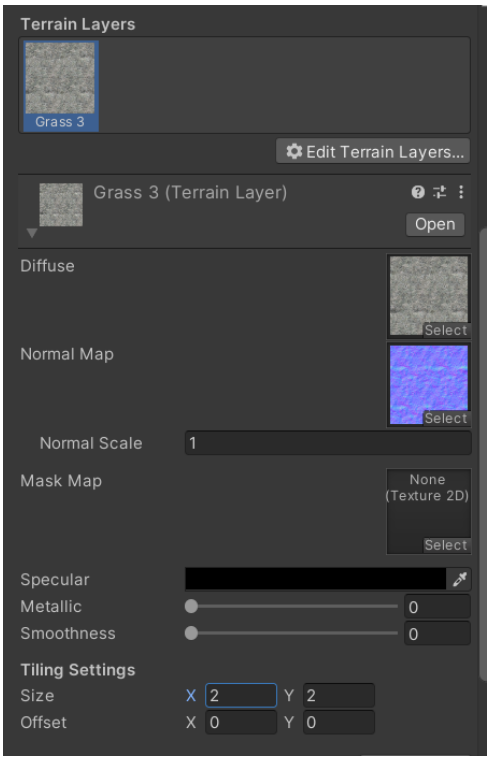


Рисунок 1.13. Налаштування інструменту Paint texture

Нижче наведено вигляд terrain з текстурою (див. рис. 1.14).

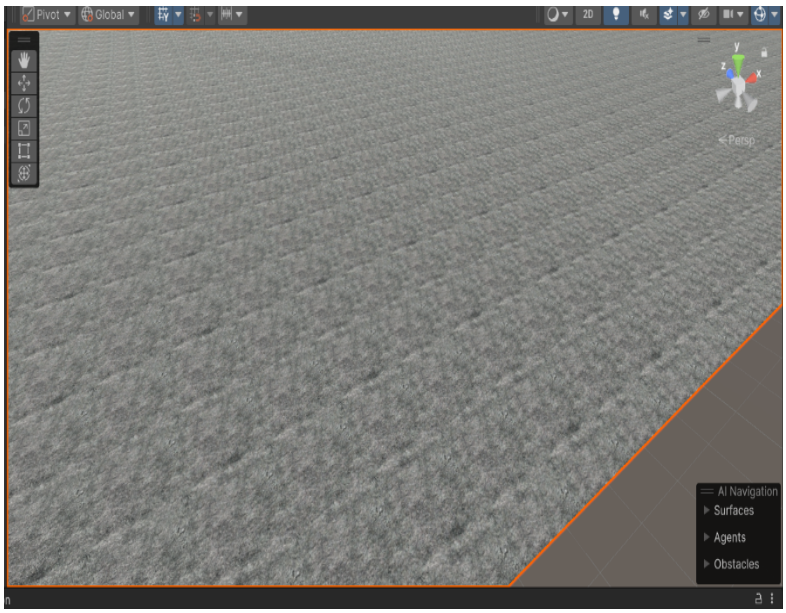


Рисунок 1.14. Вигляд текстури на terrain

Щоб створити враження живого середовища використовується інтрумент Paint Details, який дозволяє розміщувати траву, кущі, квіти та інші дрібні об'єкти на поверхні ландшафту. Щоб додати тектуру трави потрібно натиснути Edit

Models і обрати Add Grass Texture. У цьому вікні додається необхідна текстура трави. Для трави була використана тектура Grass 2 з асету Grass And Flowers Pack 1. Потім у вікні обирається потрібна тектура. Для даної тектури були застосовані наступні налаштування:

- Min Width: 0.5;
- Max Width: 1;
- Min Height: 0.1;
- Max Height: 0.6;
- Detail Density: 5;
- Healthy Color: 636363;
- Dry Color: A8A8A8.

Також налаштування наведені на рис. 1.15.

Замість яскраво-зеленої трави було обрано сірі відтінки для здорової та сухої трави, що краще відповідає приглушеній, похмурій і занедбаній атмосфері horror-сцени.

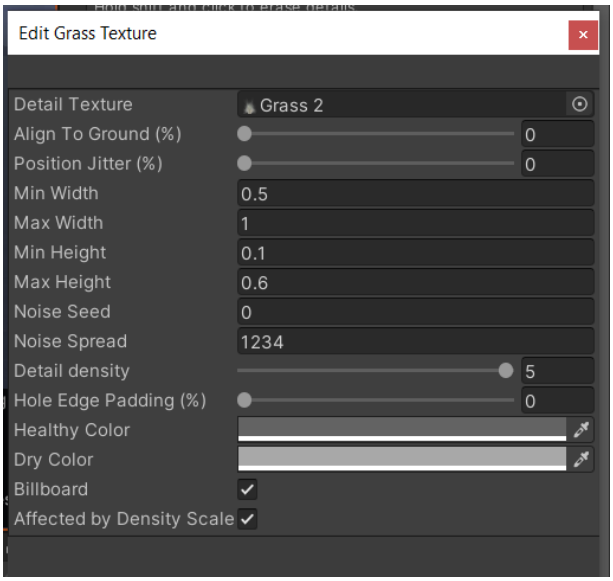


Рисунок 1.15. Налаштування текстури Grass 2

Після того, як тектура трави була додана, налаштовується пензель. Був обраний пензель builtin_brush_1 І наступні параметри:

- Brush Size:100;
- Opacity: 0.335;
- Target Strenght: 0.627451.

Також налаштування наведені на рис. 1.16.

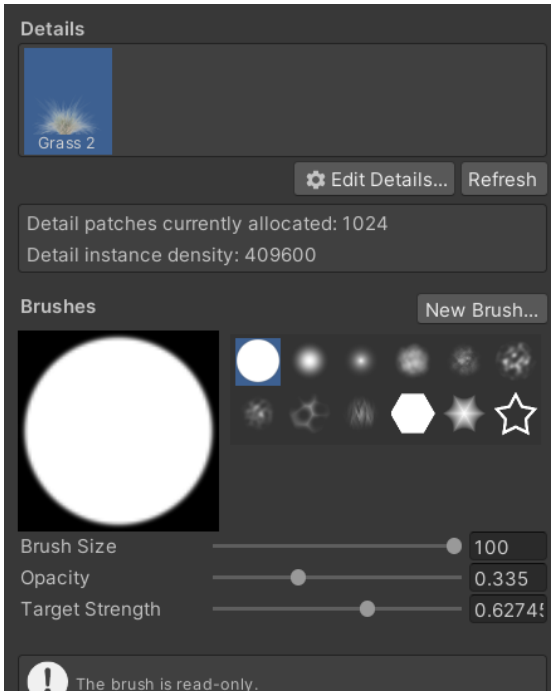


Рисунок 1.16. Налаштування інструменту Paint Details

Нижче наведено вигляд terrain з травою (див. рис. 1.17).

Рисунок 1.17. Вигляд terrain з доданою травою

Щоб створити ліс спочатку було використано інструмент Paint Trees, який дозволяє додавати дерева на Terrain у довільному порядку з автоматичним варіюванням масштабу, обертання та висоти. Щоб додати 3-D моделі дерев треба натиснути Edit Trees, потім вибрати Add Trees і обрати потрібну модель у всплиаючому вікні.

Рисунок 1.18. Налаштування інструменту Paint Trees

Поступово додаються всі моделі дерев трохи збільшуючи значення Tree Density від 50 до 62. Також був встановлений параметр Random біля налаштування Tree height і Tree weight, щоб дерева були різної висоти і ширини. Налаштування наведені на рис. 1.18, а результат на рис. 1.19.

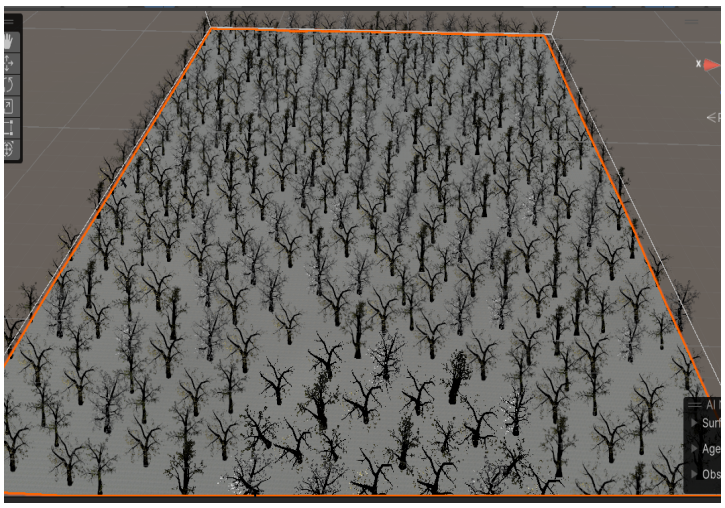


Рисунок 1.19. Вигляд terrain з доданими деревами

1.2.3 Створення лісу

Дерева, які були додані на terrain, вважаються одним цілим з ним. Це заважало роботі іншого компонента NavMeshAgent, тому був створений скрипт CreateForest для отримання всіх координат дерев і запису їх у json файл, а потім створення їх як окремих об’єктів на сцені.

Спочатку було створено клас обгортку TreeWrapper з полями для

координат, висоти, ширини і індексу дерева. Клас позначений атрибутом Serializable для того, щоб цей клас міг бути конвертованим у json формат у Unity.

```
namespace Assets.Logic.Scripts
{
    [Serializable]
    public class TreeWrapper
    {
        public float x;

        public float y;

        public float z;

        public float widthScale;

        public float heightScale;

        public int prototypeIndex;
    }
}
```

Так як дерев багато, то потрібно десь зберігати ці дані, тому був створений другий клас обгортка ArrayTreeWrapper. Він також позначений атрибутом Serializable. У класі є поле масив типу TreeWrapper, так як лише масив можна конвертувати у Json. Щоб задати розмір масиву, у конструктор класу передається змінна length типу int.

```
namespace Assets.Logic.Scripts
{
    [Serializable]
    class ArrayTreeWrapper
    {
        public TreeWrapper[] Array;

        public ArrayTreeWrapper(int length)
        {
            this.Array = new TreeWrapper[length];
        }
    }
}
```

Для створення самого json файлу у скрипті CreateForest був написаний метод CreateJsonFile(). В ньому за допомогою активного terrain отримуються екземпляри дерев у змінну . Далі змінна trees перебирається у циклі foreach. Для кожного дерева вираховується глобальні координати і всі дані записуються у екземпляр класу TreeWrapper. Об'єкт TreeWrapper серіалізується у формат JSON за допомогою JsonUtility.ToJson.

Після завершення циклу використовується конструкція using, яка відкриває файл для запису і автоматично закриває його після завершення блоку. В середині

блоку створюється об'єкт StreamWriter з методом File.CreateText("trees.json")для створення нового файлу з назвою trees.json у кореневій папці проєкту. Потім метод sw.Write(jsonString) записує текст змінної jsonString у json файл. Цей метод викликається у методі Start() і при запуску гри спрацьовує.

```
private void CreateJsonFile()
{
    var trees = terrain.terrainData.treeInstances;
    string jsonString = "{\"Trees\":[";
    foreach (var tree in trees)
    {
        Vector3 localPos = new Vector3(
            tree.position.x * terrain.terrainData.size.x,
            tree.position.y * terrain.terrainData.size.y,
            tree.position.z * terrain.terrainData.size.z
        );
        Vector3 worldPos = terrain.transform.position + localPos;
        var item = new TreeWrapper
        {
            x = worldPos.x,
            y = worldPos.y,
            z = worldPos.z,
            heightScale = tree.heightScale,
            widthScale = tree.widthScale,
            prototypeIndex = tree.prototypeIndex
        };

        jsonString += JsonUtility.ToJson(item) + ",";
    }

    jsonString += "]}";

    using (StreamWriter sw = File.CreateText("trees.json"))
    {
        sw.Write(jsonString);
    }
}
```

Для десеріалізації вмісту файлу trees.json у класі ArrayTreeWrapper був написаний метод ReadJson(string fileName). У методі відкривається блок using, де створюється екземпляр класу StreamReader і зчитаю вміст файлу методом `ReadToEnd()` і зберігає його у рядок `json`. Потім цей рядок десеріалізується і потрапляє у масив Array.

```
public void ReadJson(string fileName)
 {
     string json = "";
     using (StreamReader sw = File.OpenText(fileName))
     {
         json = sw.ReadToEnd();
     }
     var info = JsonUtility.FromJson<ArrayTreeWrapper>(json);
     Array = info.Array;
 }
```

Далі у скрипті був дописаний метод CreateForestFromJson(). Спочатку в ньому створюється екземпляр класу ArrayTreeWrapper і викликає метод ReadJson(). Потім масив перебирається циклом. Для кожного дерева створюється об'єкт на сцені методом `Instantiate` і задає всі потрібні параметри.

```
private void CreateForestFromJson()
{
    ArrayTreeWrapper listTrees = new
ArrayTreeWrapper(terrain.terrainData.treeInstances.Length);
    listTrees.ReadJson("trees.json");

    foreach (var tree in listTrees.Array)
    {
        GameObject gameObject = Instantiate(gameObjects[tree.prototypeIndex], new
Vector3(tree.x, tree.y, tree.z), new Quaternion(0, 0, 0, 0));
        gameObject.transform.localScale = new Vector3(tree.widthScale,
tree.heightScale, tree.widthScale);
    }
}
```

Метод CreateForestFromJson викликається у Start() і при запуску проекту створюються об’єкти дерев. У ієрархії виділяються усі дерева і копіюються. Після завершення проєкту створюється пустий об’єкт у ієрархії і туди вставляються усі скопійовані об’єкти (див. рис. 1.20).

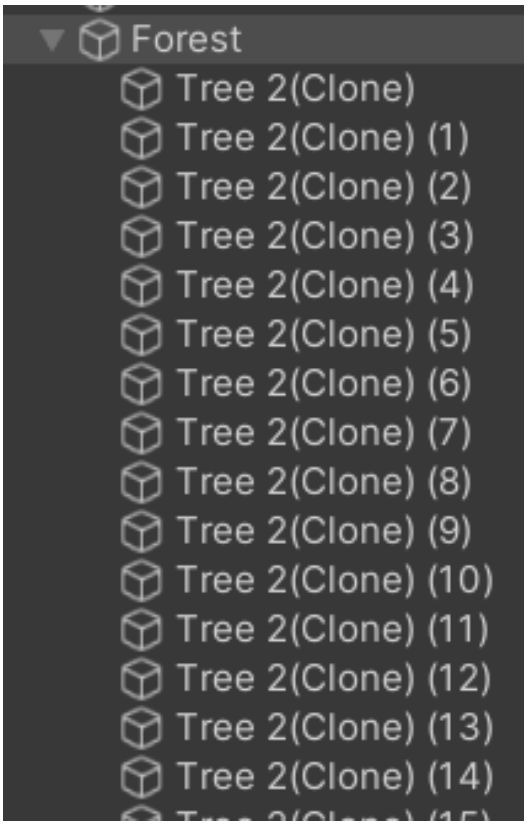


Рисунок 1.20. Вигляд об’єкту Forest у ієрархії

Нижче наведено вигляд лісу після виконання скрипту (див. рис. 1.21).

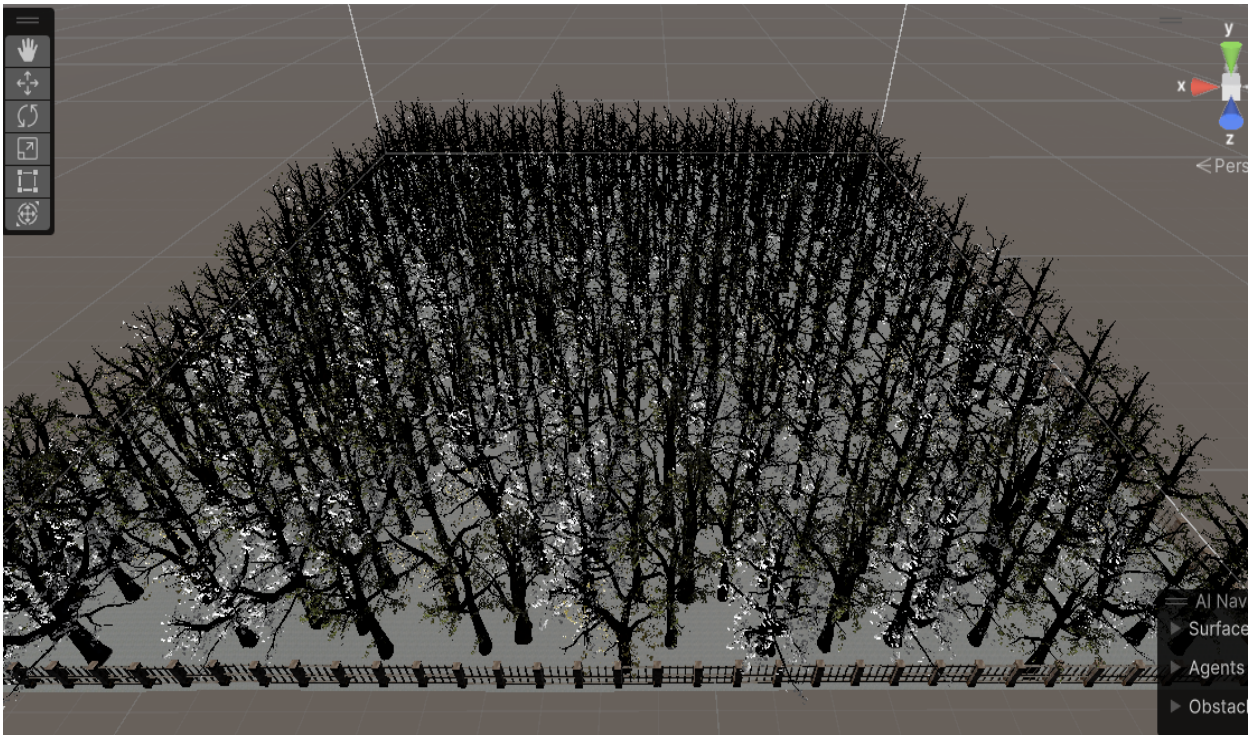


Рисунок 1.21. Вигляд сцени з деревами

1.2.4 Створення локацій

По периметру terrain був розставлений паркан (див. рис. 1.22). Паркан потрібен, щоб обмежити простір де можна пересуватися. Так гравець не може випадково чи навмисно вийти за межі мапи. Також до паркана були додані закриті ворот. Вони відкриваються лише після завершення гри.

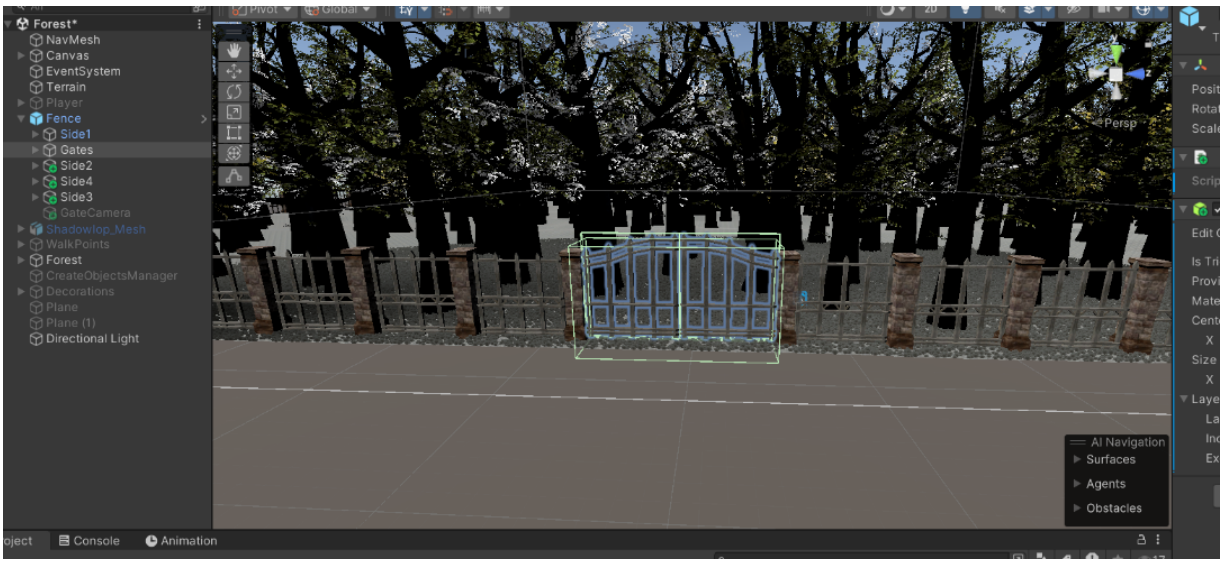


Рисунок 1.22. Локація з доданими воротами і парканом

Також у лісі були створені деякі міні локації, щоб урізноманітнити сцену і дати можливість гравцю орієнтуватись.

Перша локація це занедбаний колодязь, біля якого лежать старі відра.(див. рис. 1.23).

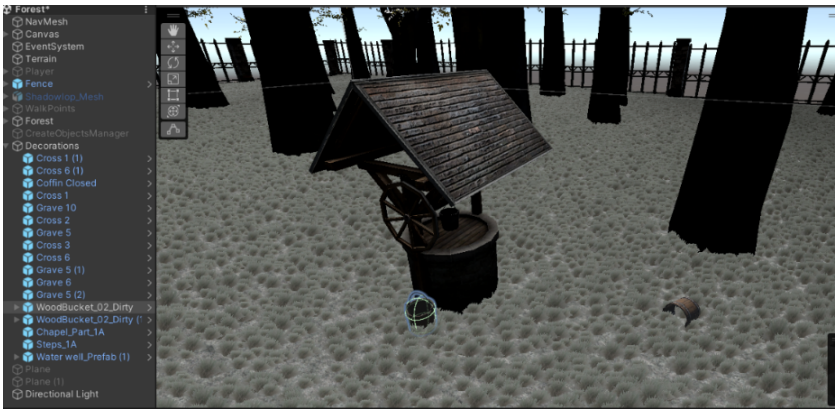


Рисунок 1.23. Локація з колодязем

Друга локація являє собою лише каплицю зі сходами (див. рис.1.24).

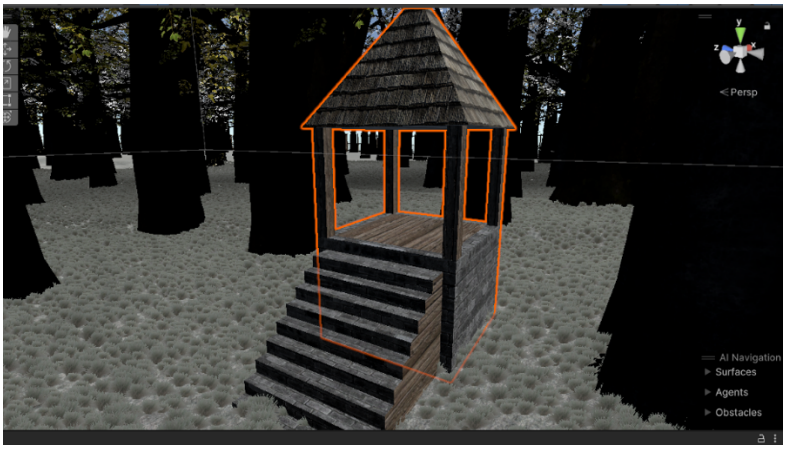


Рисунок 1.24. Локація з каплицею

Третя локація являє собою домовину, яка лежить на землі, з хрестом (див. рис.1.25).

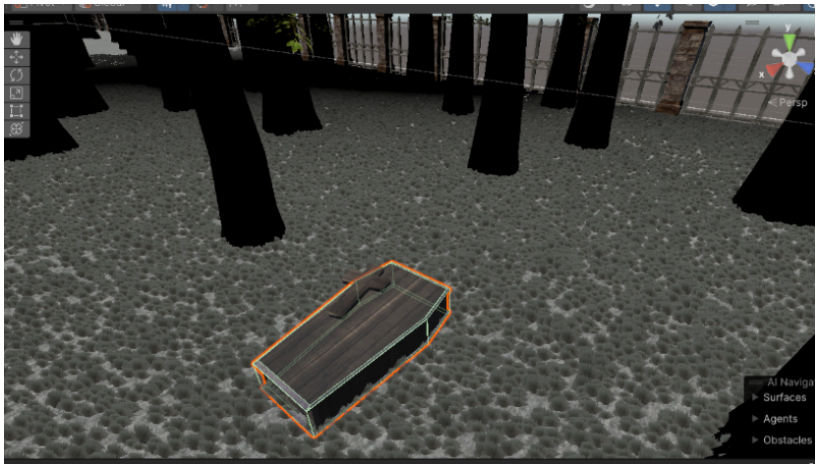


Рисунок 1.25. Локація з домовиною

Також по лісу були розставлені хрести і надгробки (див. рис. 1.26).

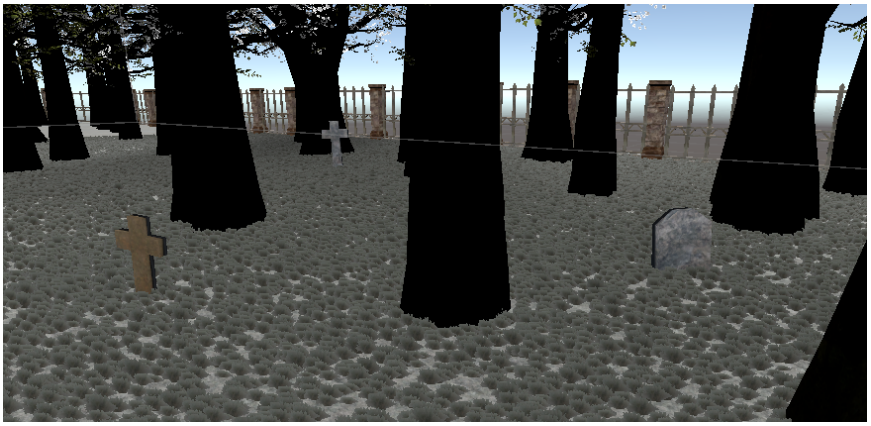


Рисунок 1.26. Хрести і надгробки

1.2.5 Налаштування туману

Налаштування туману виконується через панель *Lighting*, у вкладці *Environment*. Потрібно встановити прапорець біля пункту Fog і тоді з'являться налаштування для туману. Були налаштовані такі параметри:

- Color: 6B797D;
- Density: 0.1;
- Source: Color ;
- Ambient Color: 363A42.

Налаштування наведені на рисунку 1.27.

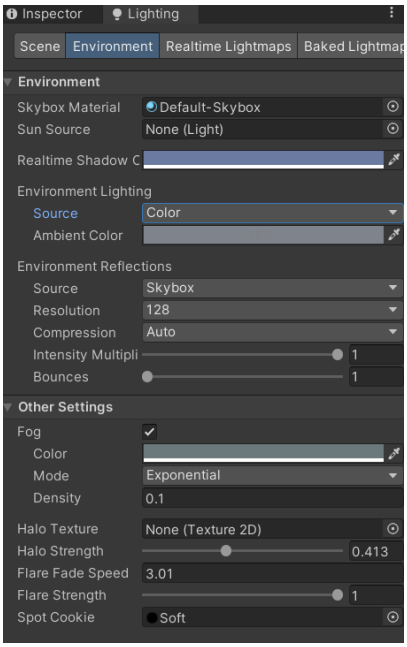


Рисунок 1.27. Налаштування туману у вкладці Environment

Нижче на рисунку 1.28 зображено рендер гри з туманом.

Рисунок 1.28. Вигляд туману у грі

1.2.6 Налаштування звукового супроводу

У ієрархії проєкту було створено пустий об’єкт з назвою BackgroundSound. Щоб зробити звуковий супровід використовується компонент AudioSource[6], який додано до об’єкту BackgroundSound. У AudioClip обрано звук mystic-forest-ambient-23812.mp3 з папки проєкту Sound. Біля параметру Play On Awake і Loop поставлені прапорці, щоб звук починав грати одразу після запуску гри і постійно повторювався. Звук був встановлений в 0.024. Всі налаштування AudioSource на рис. 1.29.

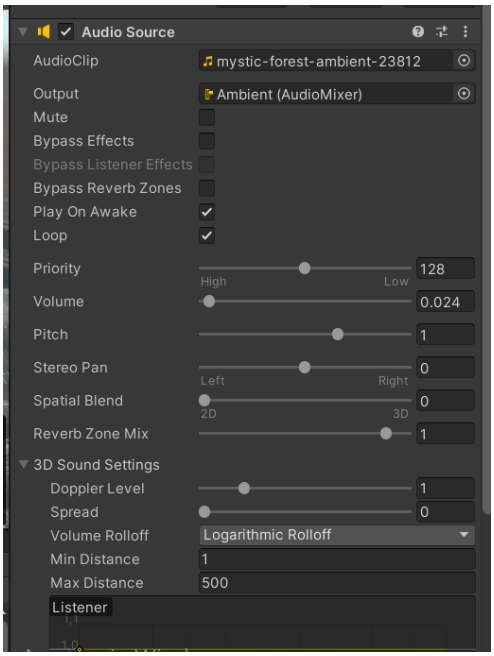


Рисунок 1.29. Налаштування Audiosource об’єкту BackgroundSound

Також для подальшої можливості регулювати гучність звуків у папці Sounds було додано AudioMixer з групами Ambient і Sounds (див. рис. 1.30). Тоді мікшер з назвою Ambient був обраний в параметрі Output

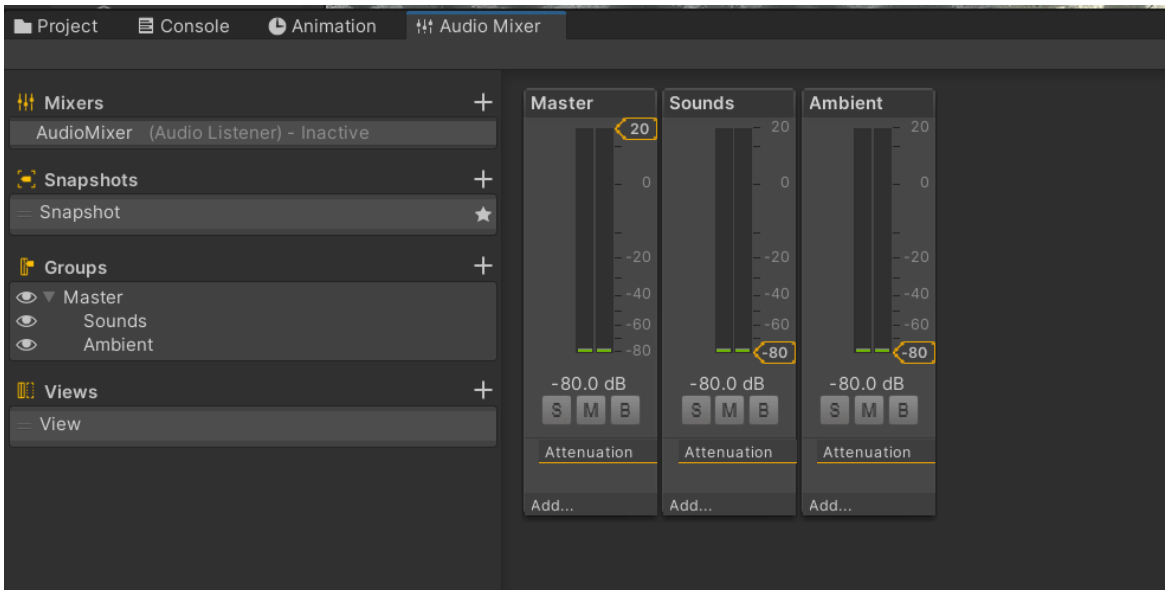


Рисунок 1.30. AudioMixer

1.3 Написання основної логіки гри

1.3.1 Налаштування основних класів і скриптів

Для початку роботи було створено декілька класів які не є скриптами, а є допоміжними.

Розглянемо клас GameInfo. В класі реалізовується дизайн патерн програмування Singleton[2] для того, щоб мати змогу отримувати одну і ту ж інформацію з різних скриптів при виклику цього класу. Щоб реалізувати патерн конструктор класу було зроблено private і було створено єдиний приватний статичний екземпляр цього ж класу _instance. Щоб мати змогу взаємодіяти з цим екземпляром створено властивість Instance, яка посилається на _instance.

У класі створено наступні публічні поля:

- DifficultyType — поточний рівень складності гри;
- CollectedToys — кількість зібраних іграшок;
- SumOfToys — загальна кількість іграшок, яку потрібно зібрати;
- EnemyWalkSpeed — швидкість ходьби ворога;
- EnemyRunSpeed — швидкість бігу ворога;

- SightRange — дальність зору ворога;
- IsGameOver — булевий прапорець, що показує, чи завершена гра;
- PlayerSpeed —швидкість ходьби гравця;
- PlayerRunningSpeed — швидкість бігу гравця;
- PlayerStaminaCapacity — об’єм витривалості гравця;
- CursorSensitivity — чутливість керування курсором.

```
public class GameInfo
{
    private GameInfo() { }

    public DifficultyType DifficultyType;

    public int CollectedToys;

    public int SumOfToys;

    public float EnemyWalkSpeed;

    public float EnemyRunSpeed;

    public float SightRange;

    public bool IsGameOver;

    public float PlayerSpeed = 0.05f;

    public float PlayerRunningSpeed = 0.1f;

    public float PlayerStaminaCapacity = 2.0f;

    public float CursorSensitivity = 200;

    private static GameInfo _instance = new GameInfo();

    public static GameInfo Instance => _instance;
}
```

Також у класі є метод UpdateGameInformation(), який зображено на рисунку 1.31. У Switch визначається рівень складності і в залежності від нього задається значення полям EnemyWalkSpeed, EnemyRunSpeed, SightRange і SumOfToys.

Метод UpdateGameInformation() дозволяє налаштовувати основні параметри гри відповідно до вибраного рівня складності, що забезпечує динамічну зміну умов гри. Це дозволяє гнучко адаптувати ігровий процес під досвід гравця та збалансувати виклик. Такий підхід підвищує реіграбельність гри, оскільки кожен рівень складності має унікальні характеристики. Всі зміни зберігаються у єдиному екземплярі класу GameInfo, завдяки чому забезпечується

узгодженість ігрового стану між різними скриптами.

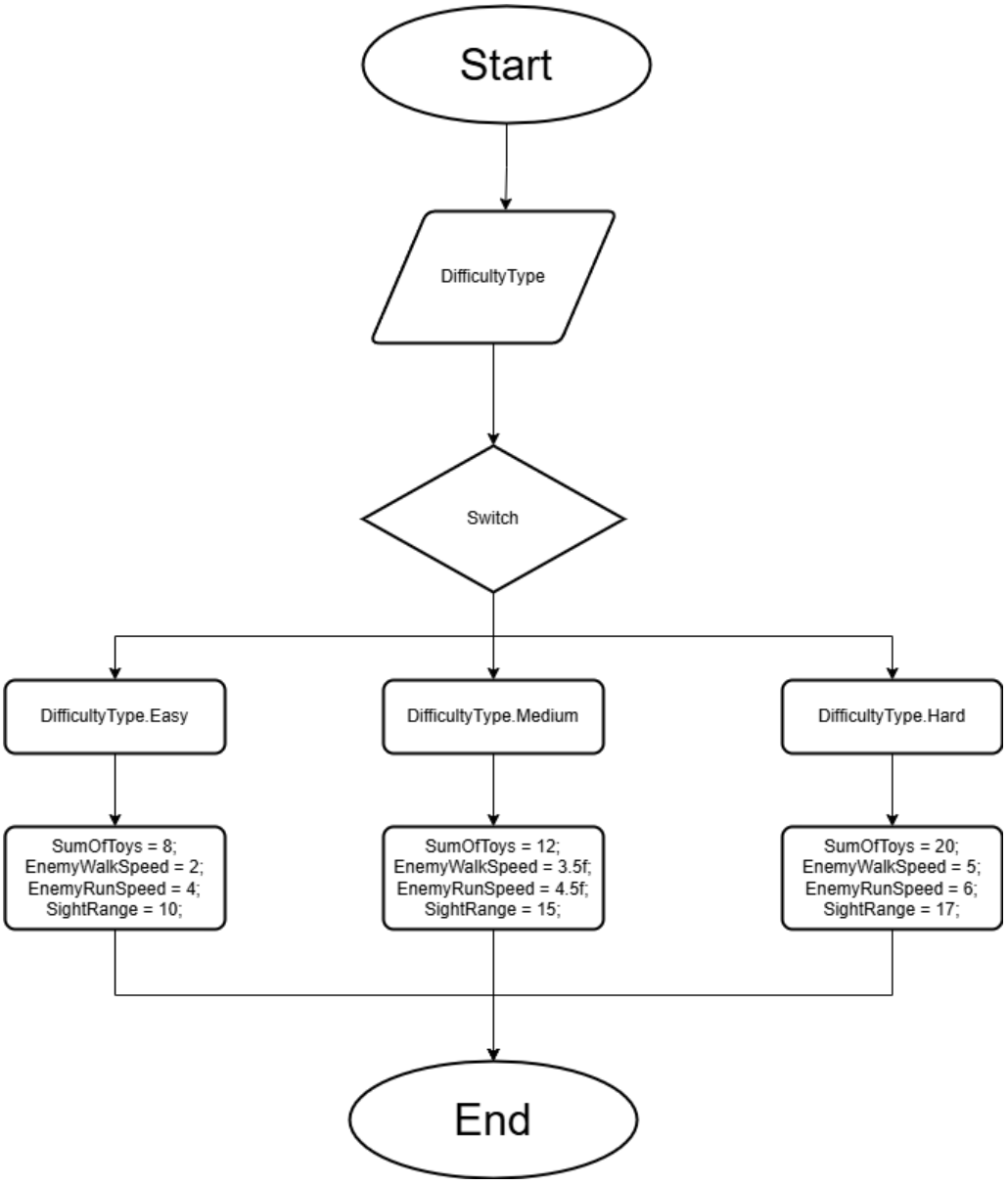


Рисунок 1.31. Схема методу UpdateGameInformation()

Далі розглянемо клас TextManager. Він теж реалізовує патерн Singleton[2], щоб мати можливість змінювати UI-тексти з інших скриптів. У класі є два публічних поля:

- CounterText — об’єкт TMP_Text, у якому, відображається кількість зібраних іграшок;
- PressC — об’єкт TMP_Text, що може показуватись на екрані, коли гравець наближається до конкретного об’єкту.

```
public class TextManager
{
    private TextManager() { }
```

```
    public TMP_Text CounterText { get; set; }

    public TMP_Text PressC { get; set; }

    private static TextManager Instance = new TextManager();

    public static TextManager TextManagerInstance => Instance;
}
```

Розглянемо схожий клас EndTextManager, який також реалізовує патерн Singleton[2]. Цей клас потрібен для передачі інформації у іншу сцену про текст, який має з'явитись після завершення гри. У класі є два публічні властивості:

- Text – напис, який має з'явитись після завершення гри;
- Color – колір, якого має бути напис.

```
class EndTextManager
{
    private EndTextManager() { }

    public string Text;

    public Color Color;

    private static EndTextManager _instance = new EndTextManager();

    public static EndTextManager Instance => _instance;
}
```

Щоб мати можливість вийти з гри, створено пустий об'єкт CloseManager зі скриптом Close. У скрипті перевіряється чи натиснута кнопка Escape і якщо натиснута, то завантажується сцена Menu, і гравець бачить головне меню гри.

```
public class Close : MonoBehaviour
{
    [SerializeField]
    private InputActionReference Escape;

    void Update()
    {
        if (Escape.action.IsPressed())
            SceneManager.LoadScene("Scenes/Menu");
    }
}
```

1.3.2 Проектування ігрового об'єкту Player

Гравець представлений у вигляді 3-D капсули з назвою Player. В ієрархії об'єкту було створено пустий об'єкт PlayerLook і джерело світла Flashlight. Також до об'єкту було додано компоненти AudioSource, Character Controller, Player Input і скрипт Player.

AudioSource потрібен для того, щоб при пересуванні гравця було чутно

звуки кроків. Звук кроків Footstep_29 був взятий з асету Forest Sounds 2023 Lite. Цей звук було обрано у параметрі AudioClip. У параметрі Output обрано мікшер Sounds. Також були налаштовані наступні параметри:

- Loop: true;
- Volume: 0.015;
- Pitch: 0.75.

Налаштування AudioSource для Player наведено на рисунку 1.32.

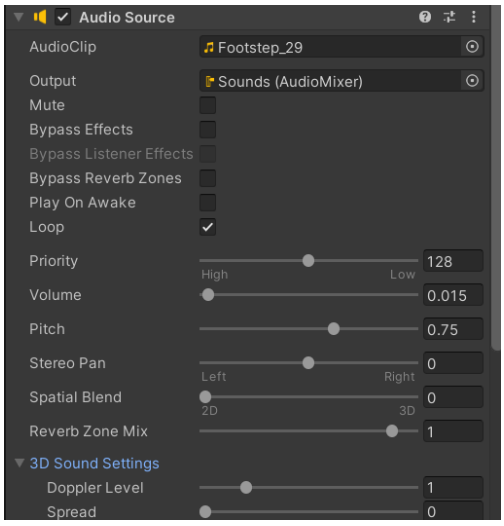


Рисунок 1.32 Налаштування AudioSource об’єкту Player

Компонент PlayerInput[6] використовується для того, щоб гравець міг керувати персонажем за допомогою створеної вище мапи дій Game Actions. У параметрі Actions було обрано мапу Game Actions (див. рисю 1.33).

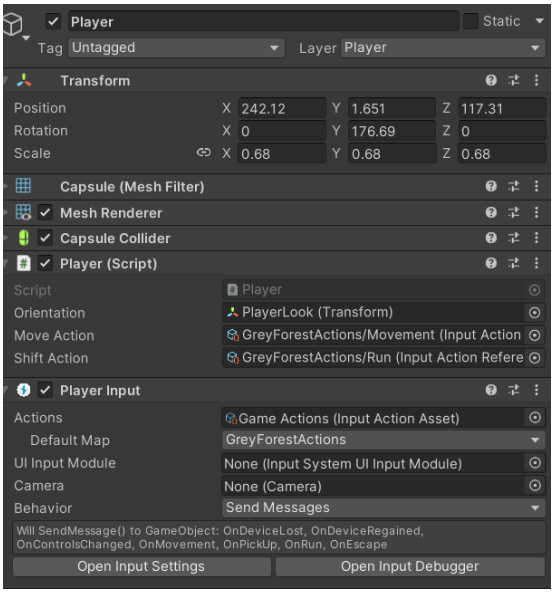


Рисунок 1.33. Inspector об’єкту Player

Далі було створено скрипт Player. На початку оголошені серіалізовані поля orientation, moveAction і runAction. Поле orientation приймає у себе об'єкт PlayerLook, moveAction і runAction – відповідні дії з Game Actions. Далі оголошуються наступні приватні поля: characterController для контролю руху, audioSource для звуку, speed і runningSpeed для швидкості пересування, staminaCapacity та timeStamina для витривалості, gravity і velocity для гравітації.

```
public class Player : MonoBehaviour
{
    [SerializeField]
    private Transform orientation;

    [SerializeField]
    private InputActionReference moveAction;

    [SerializeField]
    private InputActionReference runAction;

    private CharacterController characterController;

    private AudioSource audioSource;

    private float speed = GameInfo.Instance.PlayerSpeed;

    private  float runningSpeed = GameInfo.Instance.PlayerRunningSpeed;

    private float staminaCapacity = GameInfo.Instance.PlayerStaminaCapacity;

    private float timeStamina = GameInfo.Instance.PlayerStaminaCapacity;

    private float gravity = -9.81f;

    private float velocity;
}
```

Далі у методі Start() audioSource і characterController ініціалізуються за допомогою методу GetComponent<>().

```
void Start()
{
    characterController = GetComponent<CharacterController>();
    audioSource = GetComponent<AudioSource>();
}
```

У методі Stamina() перевіряється чи натиснуто кнопку бігу Left Shift і чи більше нуля витривалість timeStamina. Якщо так тоді швидкість змінюється на runningSpeed і змінюється pitch звуку. Якщо ж умова не виконується, то швидкість і звук повертаються до звичайного стану. На рисунку 1.34 наведено схему цього методу.

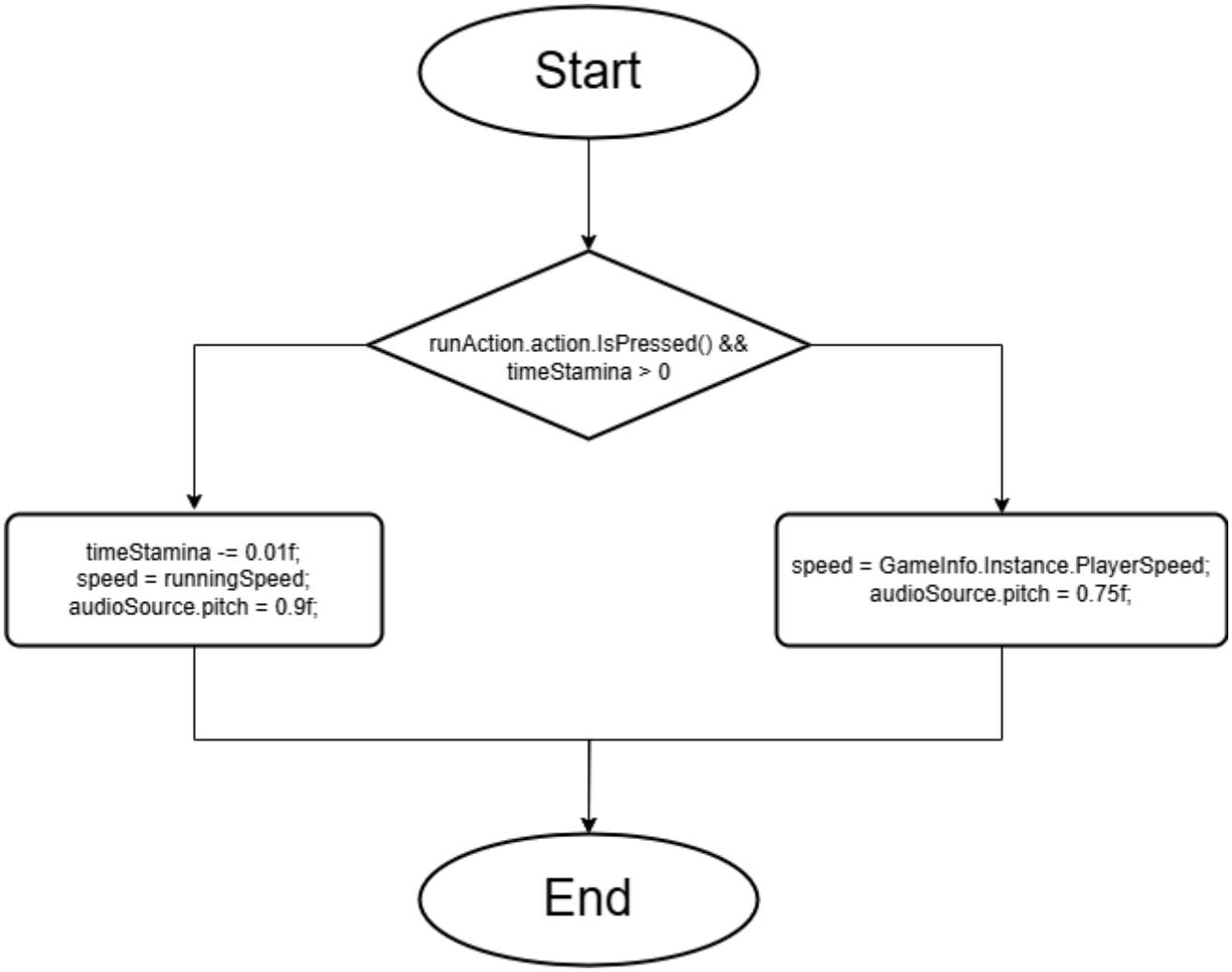


Рисунок 1.34. Схема методу Stamina()

Метод StaminaRecharging() перевіряє чи натиснуто кнопку бігу. Якщо ні, то швидкість гравця дорівнює GameInfo.Instance.PlayerSpeed. Також якщо змінна timeStamina менша за об’єм витривалості staminaCapacity, то до timeStamina буде додаватись по 0.01, щоб відновити об’єм витривалості.

```
private void StaminaRecharging()
{
    if (!runAction.action.IsPressed())
    {
        speed = GameInfo.Instance.PlayerSpeed;

        if (timeStamina < staminaCapacity)
        {
            timeStamina += 0.01f;
        }
    }
}
```

У методі Gravity() перевіряється, чи гравець знаходиться на землі через characterController.isGrounded і velocity . Якщо так, то від змінної velocity віднімається 0.1, щоб гравець не "підскакував". Інакше до velocity додається

гравітація помножена на Time.deltaTime.

```
private void Gravity()
{
    if (characterController.isGrounded && velocity < 0.0f)
    {
        velocity = -0.1f;

    }
    else
        velocity += gravity * Time.deltaTime;
}
```

Тепер можна розглянути метод Update(). Спочатку положення об'єкта transform.rotation синхронізується з напрямком orientation.rotation, щоб персонаж дивився у той же бік, що й камера. Потім зчитується напрямок руху гравця через moveAction.action.ReadValue<Vector2>() і записується у змінну action. Далі вираховується напрямок руху movementVector і викликається метод Gravity().

Якщо координати змінної action не дорівнюють нулю, об’єкт Player рухається зі швидкістю зі змінної speed за допомогою методу characterController.Move(). Якщо звук кроків ще не відтворюється, він вмикається методом audioSource.Play().

Далі викликається метод Stamina для бігу і якщо рівень складності не дорівнює DifficultyType.Hard, викликається StaminaRecharging(), який повільно відновлює timeStamina.

Якщо координати змінної action дорівнюють нулю, то звук кроків вимикається методом audioSource.Pause() і викликається метод StaminaRecharging() для відновлення timeStamina.

```
void Update()
 {
     transform.rotation = orientation.rotation;

     var action = moveAction.action.ReadValue<Vector2>();

     var movementVector = orientation.forward*action.y + orientation.right*action.x;

     Gravity();

     movementVector.y = velocity;

     if (action.y != 0f || action.x != 0f)
     {
         characterController.Move(movementVector.normalized * speed);

         if(!audioSource.isPlaying)
              audioSource.Play();
```

```
            Stamina();

            if (GameInfo.Instance.DifficultyType != DifficultyType.Hard)
            {
                StaminaRecharging();
            }
        }
        else if(action.y == 0f && action.x == 0f)
        {
            audioSource.Pause();
            StaminaRecharging();
        }
    }
```

1.3.3 Налаштування камери

Для того щоб гравець міг бачити, було налаштовано головну камеру, яка використовується для реалізації вигляду від першого лиця. У параметрах камери значення Clear Flags встановлено як Solid Color, щоб фон сцени був кольору 6B797D, заданого у параметрі Background. Всі налаштування камери наведені на рисунку 1.35.

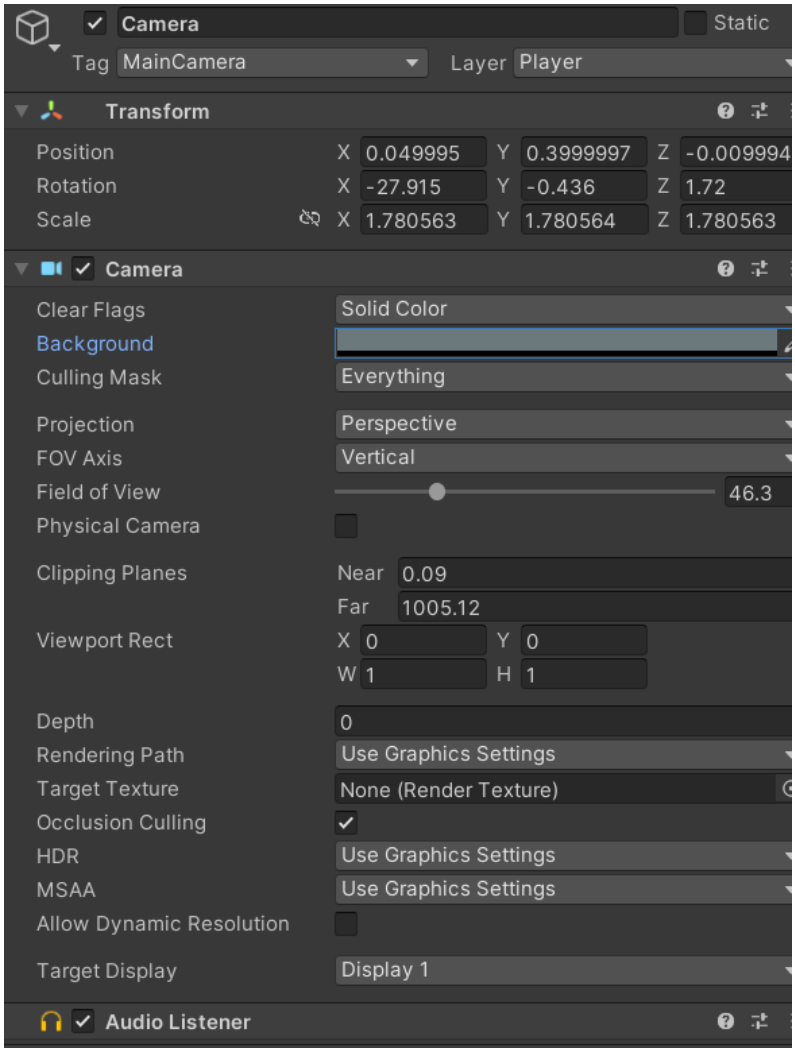


Рисунок 1.35. Налаштування камери

Також камера була переміщена у об’єкт Player.

Для камери був написаний скрипт PlayerCamera. У методі `Start()` вмикається блокування курсора і робиться його невидимим. У методі `Update()` зчитуються рухи миші по горизонталі та вертикалі, які множаться на `Time.deltaTime` та чутливість. Значення `rotationX` обмежується в межах від -90 до 90 градусів. Після цього оновлюється обертання об'єкта `orientation` за віссю Y, що визначає напрямок руху гравця. Камера, ліхтарик і позиція камери також синхронізуються з об'єктом `orientation`.

```
public class PlayerCamera : MonoBehaviour
{
    [SerializeField]
    private GameObject flashlight;
    [SerializeField]
    private Transform orientation;
    private float cursorSensitivity = GameInfo.Instance.CursorSensitivity;
    private float rotationX = 1.0f;
    private float rotationY = 1.0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
    void Update()
    {
        var mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime *
cursorSensitivity;
        var mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime *
cursorSensitivity;
        rotationY += mouseX;
        rotationX -= mouseY;
        rotationX = Mathf.Clamp(rotationX, -90f, 90f);
        orientation.rotation = Quaternion.Euler(0, rotationY, 0);
        transform.rotation = Quaternion.Euler(rotationX, rotationY, 0);
        flashlight.transform.rotation = Quaternion.Euler(rotationX, rotationY, 0);
        transform.position = orientation.position;
    }
}
```

1.3.4 Проектування ігрового об’єкту Enemy

Розглянемо створення ігрового ворога. У сцену перетягується префаб Shadowlop_Mesh. До об’єкту було додано компоненти Animator, Nav Mesh Agent, Audio Source і скрипт Enemy (див. рис. 1.36). Ці компоненти забезпечують базову функціональність ворога: анімацію, переміщення по навігаційній сітці та відтворення звуків. Скрипт Enemy відповідає за логіку поведінки ворога, зокрема переслідування гравця та реакцію на події у грі.

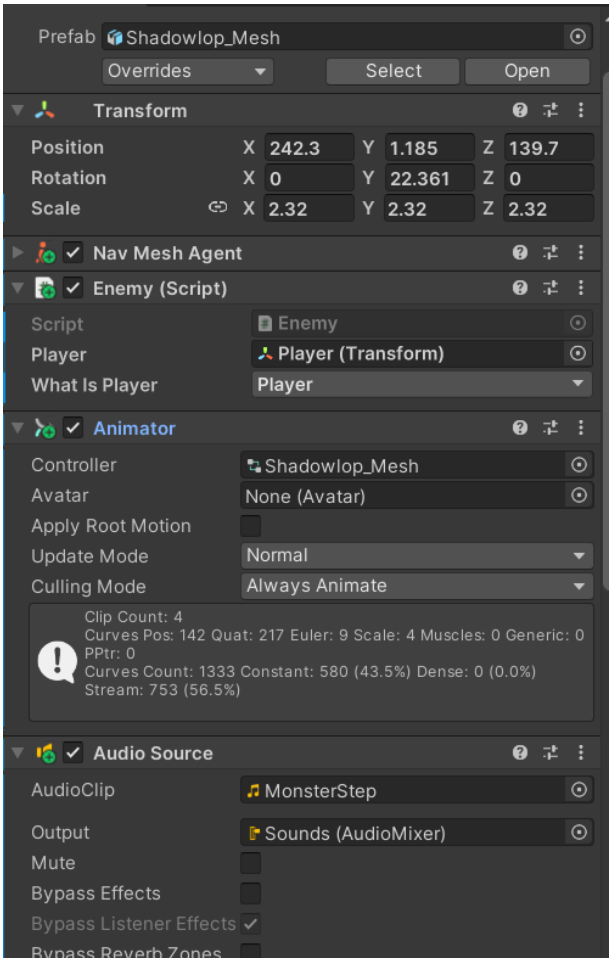


Рисунок 1.36. Inspector об’єкту Shadowlop_Mesh

Для того щоб використовувати анімації об’єкту Enemy було створено Animator. У вікні аніматор було створено Blend Tree і анімація Screamer (див. рис. 1.37).

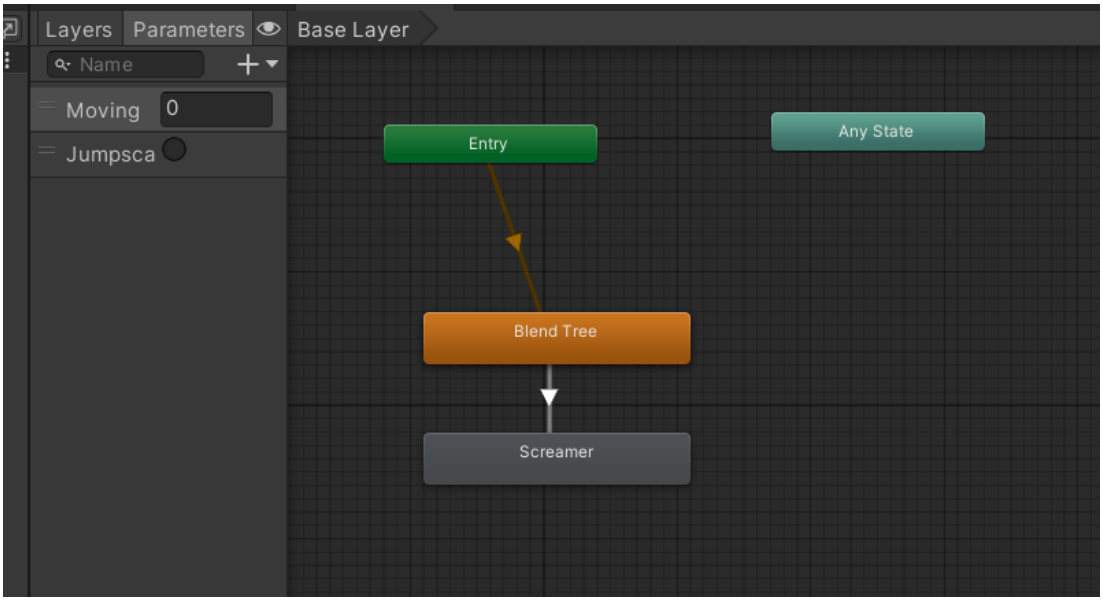


Рисунок 1.37. Вигляд Animator

До Blend Tree додано три анімації Idle, Walk і Chase і відповідно до них виставлено значення Threshold 0, 1 і 2 (див. рис. 1.38). Також щоб використовувати це дерево створено Параметр типу float з назвою Moving. Щоб використовувати анімацію Screamer було створено параметр типу bool з назвою Jumpscare.

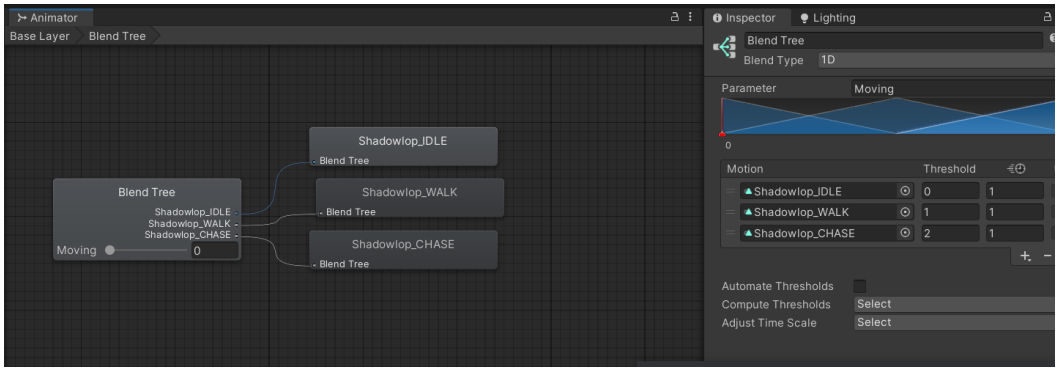


Рисунок 1.38. Вигляд Blend Tree

Для того щоб ворог міг пересуватись по сцені було встановлено пакет NavMeshAgent. Далі у ієрархії проєкту було створено пустий об'єкт NavMesh. До нього було додано компонент NavMeshSurface (див. рис. 1.39), щоб створити поверхню по якій можна пересуватись.

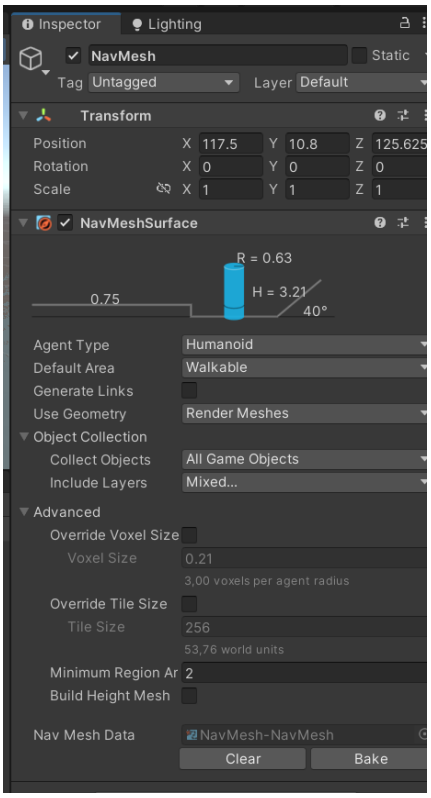


Рисунок 1.39. Налаштування NavMeshSurface

Розглянемо скрипт Enemy. У поле player передається компонент У методі Start() виконується ініціалізація полей agent, animator і audioSource методом GetComponent<>(), а також зчитуються точки для патрулювання з файлу “walkPoints.json”.

```
[SerializeField]
private Transform player;

[SerializeField]
private LayerMask whatIsPlayer;

private Animator animator;

private AudioSource audioSource;

private NavMeshAgent agent;

private float speed = GameInfo.Instance.EnemyWalkSpeed;

private float runSpeed = GameInfo.Instance.EnemyRunSpeed;

private float sightRange = GameInfo.Instance.SightRange;

private bool playerInSightMesh;

private Vector3 walkPoint;

private bool walkPointIsSet;

private bool isCoroutineEnd;

private ArrayWalkPointWrapper arrayWalkPointWrapper = new
ArrayWalkPointWrapper(29);

void Start()
{
    agent = GetComponent<NavMeshAgent>();
    agent.speed = speed;
    animator = GetComponent<Animator>();
    audioSource = GetComponent<AudioSource>();
    arrayWalkPointWrapper.ReadJson("walkPoints.json");
}
```

Метод ChasePlayer() реалізує логіку переслідування гравця. У ньому встановлюється швидкість агента agent.speed на значення runSpeed, перевіряється чи грає вже звук через audioSource.isPlaying і якщо ні, то звук вмикається. Метод transform.LookAt() повертає об’єкт Enemy до гравця і встановлює позицію гравця player.position як ціль для руху. Також викликається метод animator.SetFloat(), який парамету “Moving” присвоює значення 2, що запускає анімацію бігу Run.

```
private void ChasePlayer()
      {
          agent.speed = runSpeed;
          audioSource.pitch = 1.5f;
```

```
        if (!audioSource.isPlaying)
        {
            audioSource.Play();
        }

        transform.LookAt(player);

        agent.SetDestination(player.position);

        animator.SetFloat("Moving", 2);
    }
```

Метод SetWalkPoint() використовується для вибору випадкової точки патрулювання Викликається метод animator.SetFloat(), який парамету “Moving” присвоює значення 1, що запускає анімацію ходьби Walk. Потім

випадковим чином обирає індекс з масиву arrayWalkPointWrapper і присвоює координати змінній walkPoint. Після цього змінна walkPointIsSet стає true.

```
private void SetWalkPoint()
{
    animator.SetFloat("Moving", 1);
    int walkPointIndex = Random.Range(0, arrayWalkPointWrapper.WalkPoints.Length -
1);
    walkPoint.x = arrayWalkPointWrapper.WalkPoints[walkPointIndex].x;
    walkPoint.y = 0;
    walkPoint.z = arrayWalkPointWrapper.WalkPoints[walkPointIndex].z;
    walkPointIsSet = true;
}
```

У метод Patroling() перевіряється, чи задани координати через walkPointIsSet.Якщо walkPointIsSet дорівнює false, звук призупиняється, встановлюється анімація Idle в і запускається корутина Wait(), після завершення якої викликається метод SetWalkPoint(). Якщо walkPointIsSet == true, об’єкт починає рух до заданої точки за допомогою методу від NavMeshAgent agent.SetDestination(). Далі якщо звук не відтворюється, він запускається. Після цього виконується перевірка дистанції між ворогом і точкою призначення методом Vector3.Disance(). Якщо ворог наблизився на відстань менш ніж 1 одиниця, змінна walkPointIsSet стає false. На рисунку 1.40 наведено схемо даного методу. Такий підхід дозволяє реалізувати поведінку патрулювання, яка виглядає природно та змінюється залежно від умов гри. Завдяки використанню корутини Wait() ворог робить паузи між переміщеннями, що додає реалістичності його поведінці. Комбінація перевірки відстані та стану анімації дозволяє плавно переходити між станами очікування та руху.

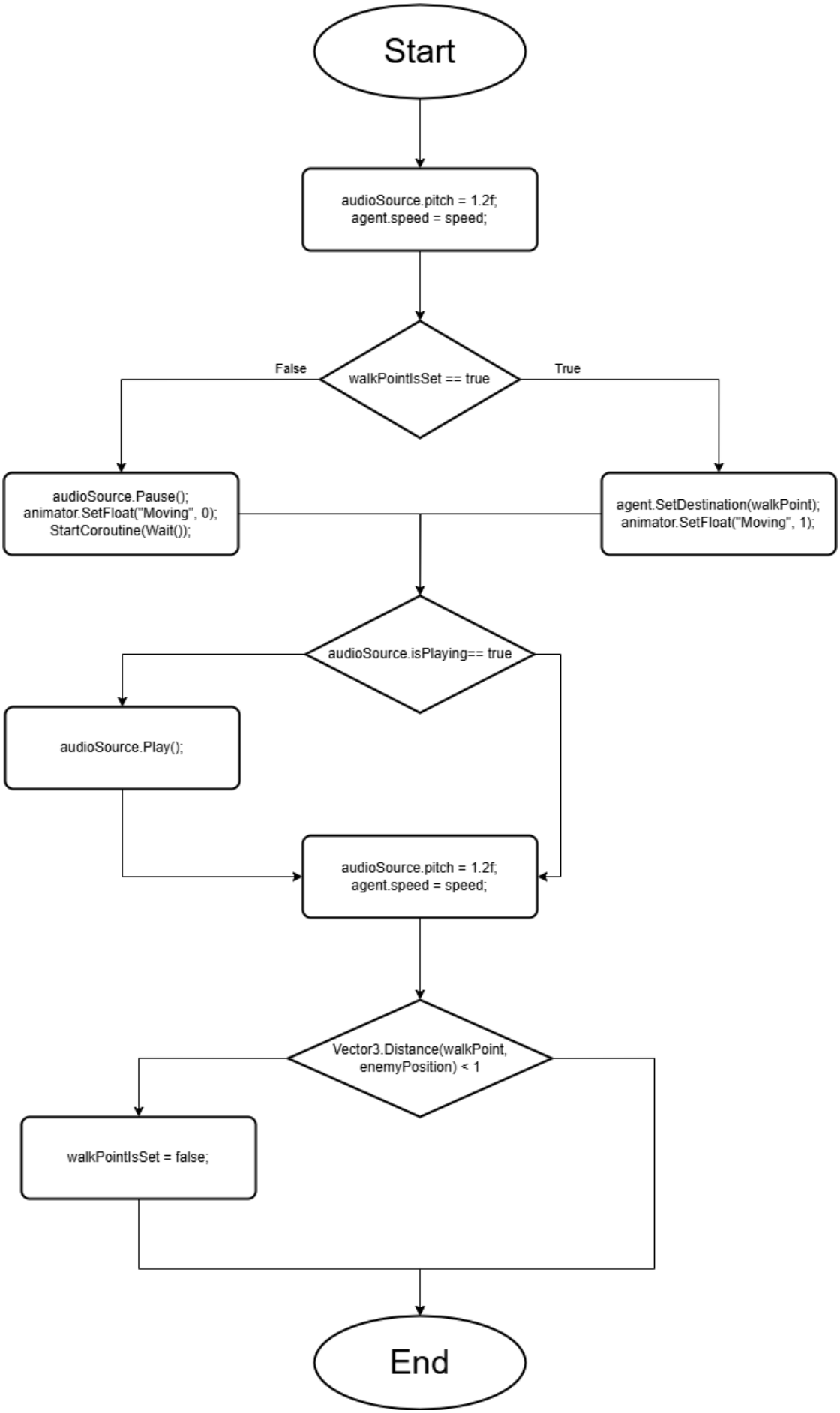


Рисунок 1.40. Схема методу Patroling()

Тепер розглянемо метод Update(). У ньому перевіряється чи знаходиться гравець у полі ору ворога за допомогою метода Physics.CheckSphere(). Якщо playerInSightMesh дорінює true, то викликається метод ChasePlayer() для переслідування гравця, якщо false – то Patroling() для пересування по сцені.

```
void Update()
{
    animator.SetFloat("Moving", 1);
    playerInSightMesh = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);


    if (playerInSightMesh)
    {
        ChasePlayer();
    }

    else
        Patroling();
}
```

1.3.5 Створення поганої кінцівки

Для кінцівки де гравця зловили до об’єкту Shadowlop_Mesh пустий об’єкт CameraHolder і Ending. У об’єкт CameraHolder додано камеру і spotlight. Нижче на рисунку 1.41 зображено актуальний вигляд Shadowlop_Mesh.

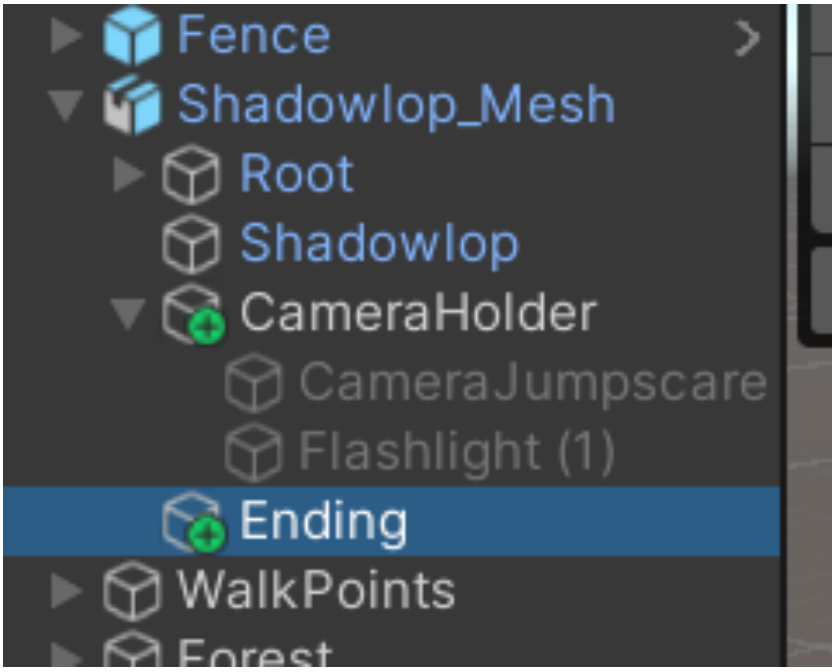


Рисунок 1.41. Вигляд об’єкту Shadowlop_Mesh у ієрархії

До об’єкту Ending було додано коспоненти Capsule Collider, AudioResource і скрипт Bad Ending (див. рис. 1.42).

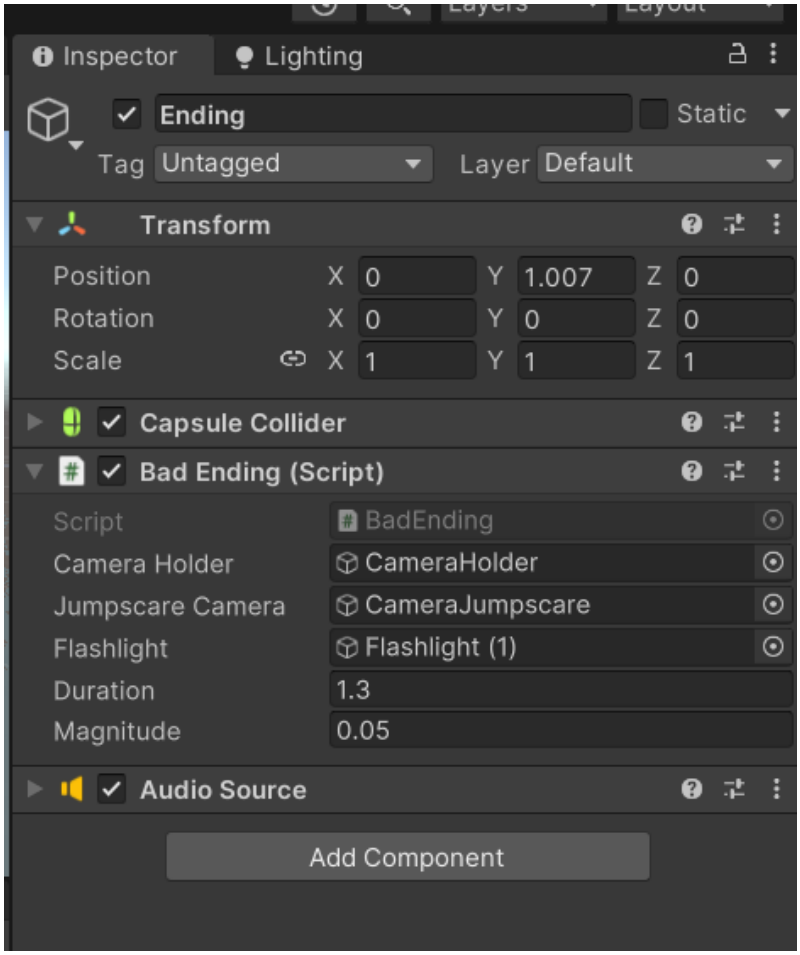


Рисунок 1.42. Inspector об’єкту Ending

Capsule Collider є тригером тому розглянемо OnTriggerEnter. Спочатку у цьому методі визначається батьківський об'єкт та отримується компонент `Animator`. Потім вимикається головна камера `Camera.main` і активується камера для скримера `jumpscareCamera` та ліхтарик. Тригер анімації `Jumpscare` методом `SetTrigger("Jumpscare")`) і відтворюється звук скрімера через `PlayOneShot()`. В кінці значення змінної `IsAnimationEnd` встановлюється в `true`.

```
public void OnTriggerEnter(Collider other)
{
    var parent = transform.parent.gameObject;
    animator = parent.GetComponent<Animator>();

    Camera.main.gameObject.SetActive(false);
    jumpscareCamera.SetActive(true);
    Flashlight.SetActive(true);

    animator.SetTrigger("Jumpscare");

    AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);

    audioSource.PlayOneShot(audioSource.clip);
    IsAnimationEnd = true;
}
```

Тепер розглянемо корутину Shake(). Вона відповідає за ефект тремтіння камери. Спочатку зберігається початкове положення камери. Далі в циклі while позиція камери змінюється випадковим чином по осях X та Z. Цикл триває протягом вказаного часу duration. Після завершення камера повертається у своє початкове положення, а змінна IsShakingEnd встановлюється в true.

```
private IEnumerator Shake()
{
    Vector3 vector = jumpscareCamera.transform.localPosition;

    float elapsed = 0.0f;

    while (elapsed < duration)
    {
        float x = Random.Range(-1f, 1f) * magnitude;
        float z = Random.Range(-1f, 1f) * magnitude;

        jumpscareCamera.transform.localPosition = new Vector3(x, vector.y, z);

        elapsed += Time.deltaTime;

        yield return null;
    }

    jumpscareCamera.transform.localPosition = vector;
    IsShakingEnd = true;
}
```

Далі розглянемо метод Update(). Якщо значення isAnimationEnd дорівнює true, то запускається корутина і після неї відтворюється метод Shake(). Далі якщо IsShakingEnd дорівнює true, то задається текст і колір тексту класу EndTextManager, і завантажується меню.

```
void Update()
{
    if (IsAnimationEnd)
    {
        StartCoroutine(Coroutine());
        if (IsCoroutineEnd)
        {
            jumpscareCamera.transform.localPosition = new Vector3(0, 0, 0);
            StartCoroutine(Shake());
        }
    }

    if (IsShakingEnd)
    {
        EndTextManager.Instance.Text = endText;
        EndTextManager.Instance.Color = Color.red;
        GameInfo.Instance.IsGameOver = true;
        SceneManager.LoadScene("Scenes/Menu");
    }
}
```

1.3.6 Створення скрипту для відкриття воріт

До об’єктів Gate1 і Gate2 було додано скрипт OpenGate (див. рис. 1.43) для того, щоб ворота могли зачинитися в певний момент.

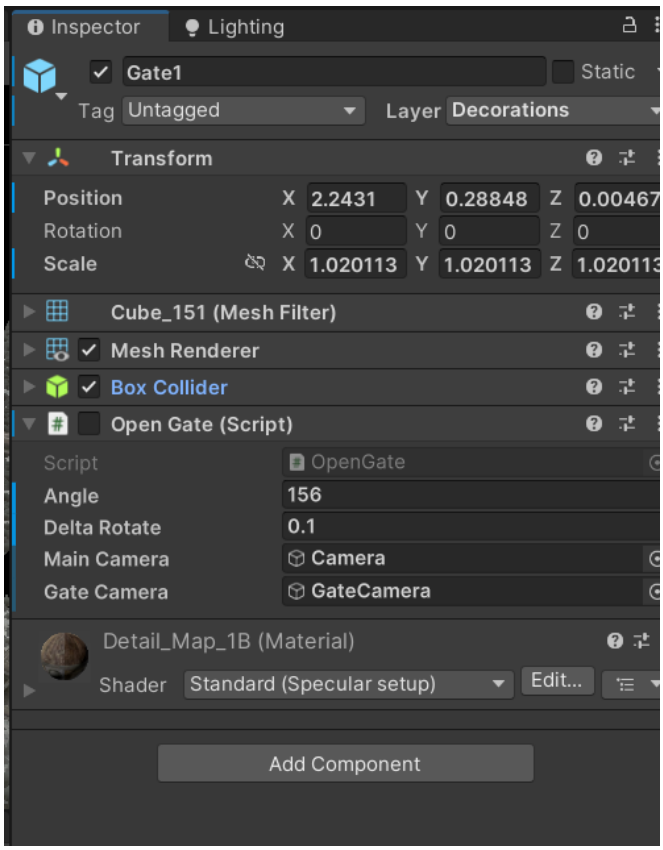


Рисунок 1.43. Inspector об’єкту Gate1

Розглянемо метод RotateGate() скрипта OpenGate. Спочатку створюється змінна з кутами у кватерніонах. Далі об’єкта обертається в напрямку цього кута за допомогою `Quaternion.RotateTowards`, з кроком `deltaRotate`. Після цього запускається корутина методом `StartCoroutine(Coroutine(3))` з затримкою 3 секунди і викликається метод `DeactivateScript()`, що деактивує даний скрипт і об’єкт.

```
private IEnumerator RotateGate()
 {

     Quaternion targetRotation = Quaternion.Euler(0, angle, 0);
     transform.rotation       =       Quaternion.RotateTowards(transform.rotation,
targetRotation, deltaRotate);

     yield return StartCoroutine(Coroutine(3));

     DeactivateScript();
 }
```

1.3.7 Проектування ігрового об’єкту Toy

В процесі гри гравцю потрібно збирати іграшки, тому потрібно було розставити їх по карті. Був створений пустий об’єкт, куди в ієрархію додавались префаби Toy. Координати об’єктів потрібно було десь зберігати, тому було вирішено їх зберігати у json файлі. Це було зроблено таким таким способом як і з древами, тому було створено класи обгортки ToyWrapper і ArrayToyWrapper, а також скрипт ToyJsonConverter, в якому записуються всі координати у json файл toys.json. Після запису файлу всі префаби були видалені.

Так як іграшок на різних рівнях різна кількість у скрипті CreateObjectsManager створено метод CreateToysFromJson(int sumOfToys), який зчитує, записує у масив і створює у сцені лише певну кількість об’єктів ToyWrapper в залежності від значення мінної sumOfToys.

```
private void CreateToysFromJson(int sumOfToys)
{
    ArrayToyWrapper listTrees = new ArrayToyWrapper(sumOfToys);
    listTrees.ReadJson("toys.json");

    float scale = 0.3f;

    Debug.Log(sumOfToys);

    foreach (var toy in listTrees.Array)
    {
        GameObject gameObject = Instantiate(toyPrefab, new Vector3(toy.x, toy.y,
toy.z), new Quaternion(0, 0, 0, 0));
        gameObject.transform.localScale = new Vector3(scale, scale, scale);
        gameObject.transform.Rotate(new      Vector3(toy.xAngle,      toy.yAngle,
toy.zAngle));
    }
}
```

Далі для префаба Rabbit було додано пустий об’єкт з назвою Collider. До цього об’єкту було додано компонент SphereCollider, Player Input і скрипт Toy (див. рис. 1.44). У колайдері встановлено прапорець біля параметру IsTrigger і для Radius встановлено значення 0.75. Компонент SphereCollider із встановленим прапорцем IsTrigger дозволяє реагувати на події входу гравця в зону без фізичної взаємодії. Завдяки радіусу 0.75 зона виявлення є достатньо компактною для точної взаємодії з іграшкою. Скрипт Toy обробляє логіку збору іграшки та може викликати відповідні події, наприклад, збільшення лічильника зібраних об'єктів. Компонент Player Input забезпечує можливість взаємодії з об'єктом через

керування гравця.

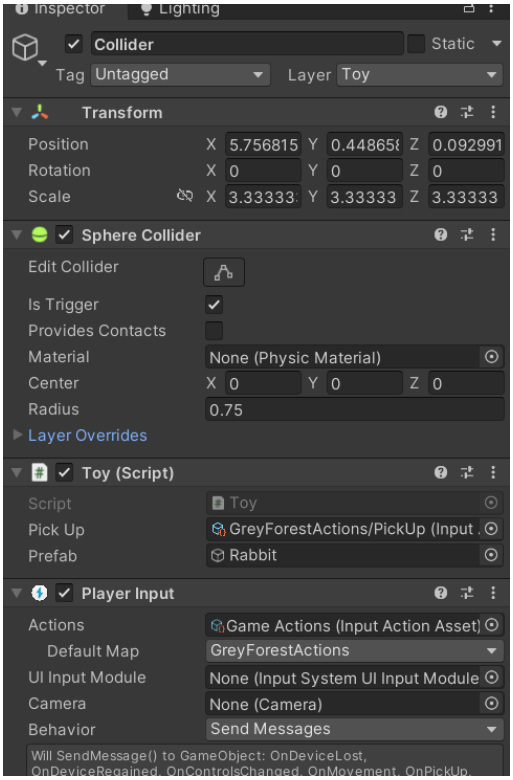


Рисунок 1.44. Inspector об’єкту Collider

Розглянемо доданий до об’єкту Collider скрипт Toy.

У методі Start() виконується пошук об’єкту Fence за допомогою методу GameObject.Find() і потім ініціалізується поле gateCamera.

```
public void Start()
{
     var fence = GameObject.Find("Fence");
     gateCamera = fence.transform.Find("GateCamera").GameObject();
}
```

Так як об’єкт Collider є тригером, в скрипті написан метод OnTriggerEnter(Collider other), який спрацьовує якщо об’єкт Player входить у колайдер.Метод встановлює значення true в полі inArea і робить об’єкт PressC видимим.

```
public void OnTriggerEnter(Collider other)
 {
     inArea = true;
     TextManager.TextManagerInstance.PressC.gameObject.SetActive(true);
 }
```

Також був написаний метод OpenGate() для відкриття воріт.Він деативує головну камеру і активує камеру, яка направлена ворота, і робить активними скрипти OpenGate об’єктів Gate1 і Gate2.

```
private void OpenGate()
{
    gameObject.SetActive(false);
    gateCamera.SetActive(true);

    GameObject gate1 = GameObject.Find("Fence/Gates/Gate1");
    OpenGate openGate1 = gate1.GetComponent<OpenGate>();
    openGate1.enabled = true;

    GameObject gate2 = GameObject.Find("Fence/Gates/Gate2");
    OpenGate openGate2 = gate2.GetComponent<OpenGate>();
    openGate2.enabled = true;
}
```

У методі Update() перевіряється чи натиснута кнопка C і чи поле inArea дорівнює true. Якщо так, то викликається метод GameInfo.Instane.CollectToy() і знищує даний перфаб. Також перевіряється чи були зібрані всі іграшки, і якщо так викликається метод OpenGate(). На рисунку 1.45 зображено схему цього методу.

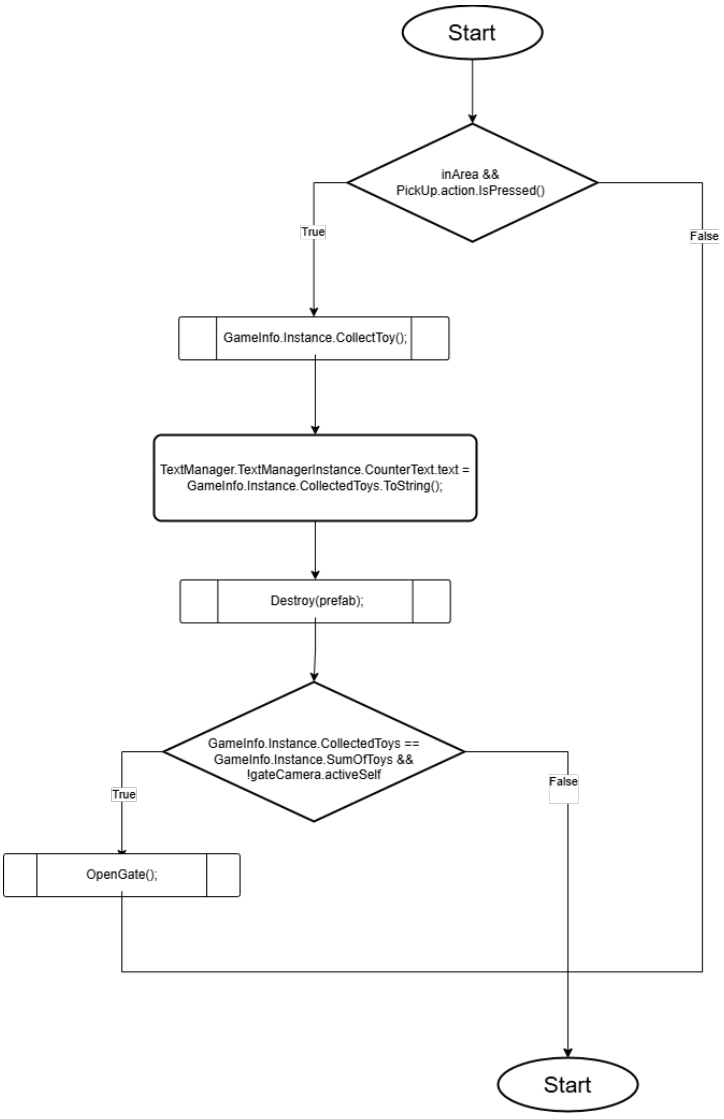


Рисунок 1.45. Схема методу Update()

1.3.8 Створення хорошої кінцівки

Для того щоб гравець міг завершити гру він повинен вийти за ворота. Для цього був створений пустий об’єкт Gates, куди вкладено об’єкти Gate1 і Gate2.

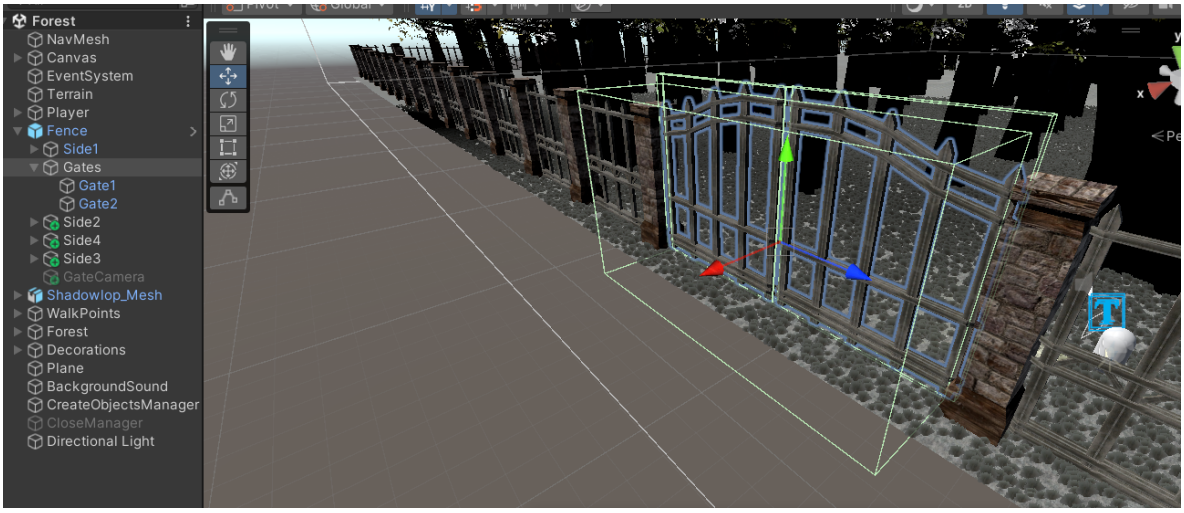


Рисунок 1.46. Вигляд колайдеру об’єкту Gates

До об’єкту було додано компонент BoxCollider (див. рис. 1.46). У колайдері встановлено прапорець біля параметру IsTrigger, щоб колайдером слугував тригером (див. рис. 1.47).

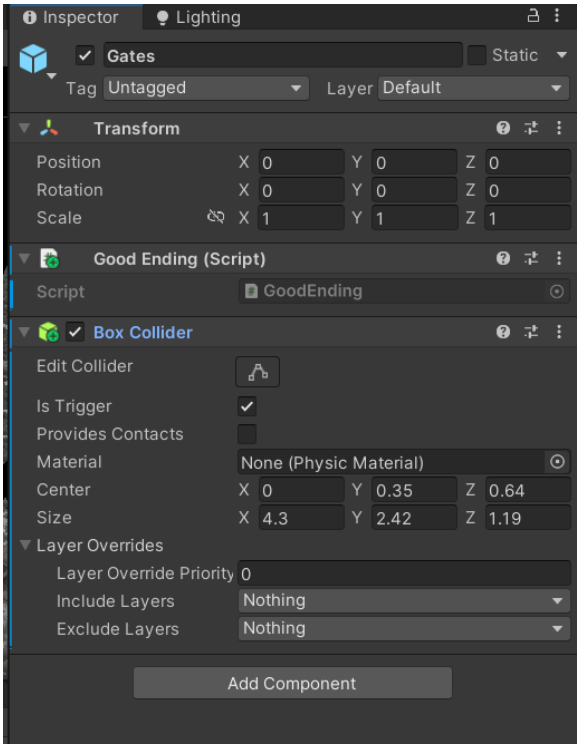


Рисунок 1.47. Inspector об’єкту Gates

Також до об’єкту Gates було додано скрипт GoofEnding. У скрипті є метод OnTriggerExit(Collider other), який викликаються, коли гравець входить у

колайдер. У методі в EndTextManager передається текст і колір тексту, а також завантажується сцена з меню.

```
public class GoodEnding: MonoBehaviour
{
    private readonly string endText = "Win";
    public void OnTriggerExit(Collider other)
    {
        EndTextManager.Instance.Text = endText;
        EndTextManager.Instance.Color = Color.green;
        GameInfo.Instance.IsGameOver = true;
        SceneManager.LoadScene("Scenes/Menu");
    }
}
```

1.4 Створення ігрового інтерфейсу

1.4.1 Створення MainMenu

У головному меню гравець може обрати рівень, почати гру, подивитись клавіші для гри і вийти з гри. Для створення меню у папці проєкту Scenes[6] було створено нову сцену з назвою Menu (див. рис. 1.48).

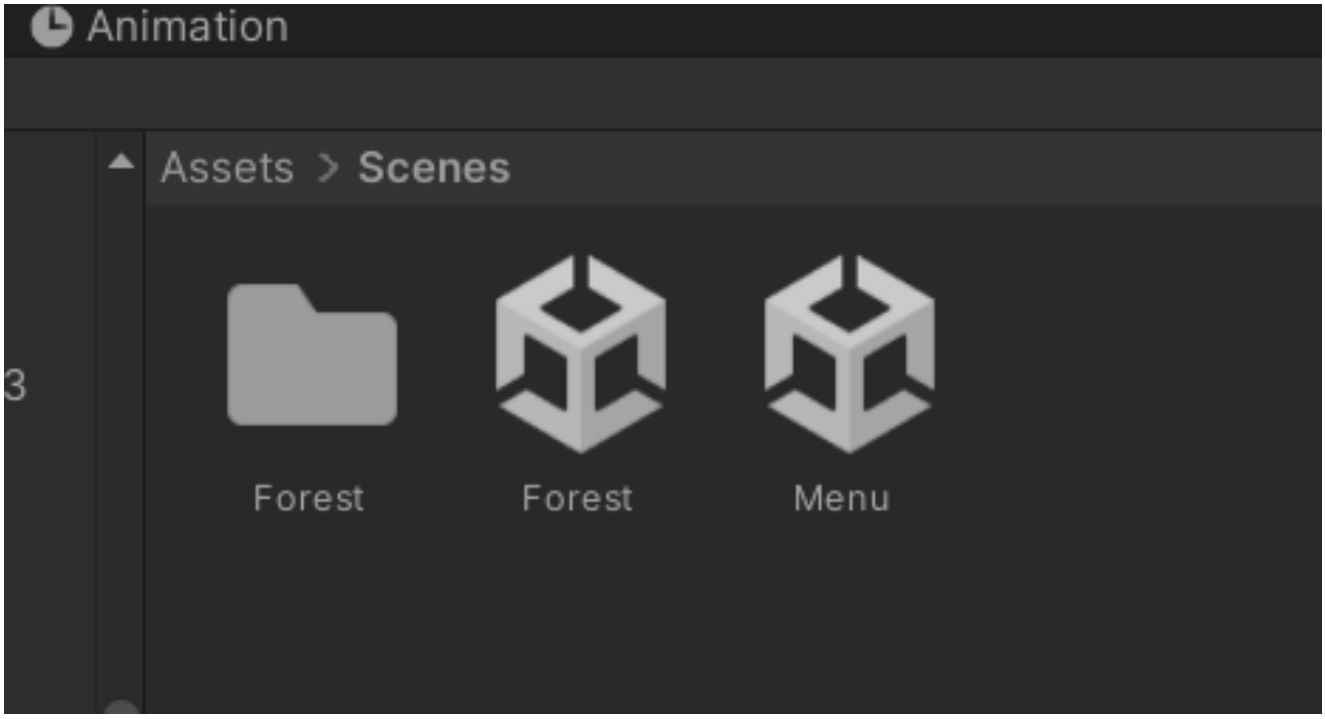


Рисунок 1.48. Папки Scenes

У ієрархії створено UI об’єкт Canvas. В налаштуваннях Render Mode обрано Screen Space – Camera, щоб інтерфейс був прив’язаний до камери і виглядав однаково при будь-якій роздільній здатності. Для цього в поле **Render Camera** була призначена основна камера сцени. Налаштування наведені на рисунку 1.49.

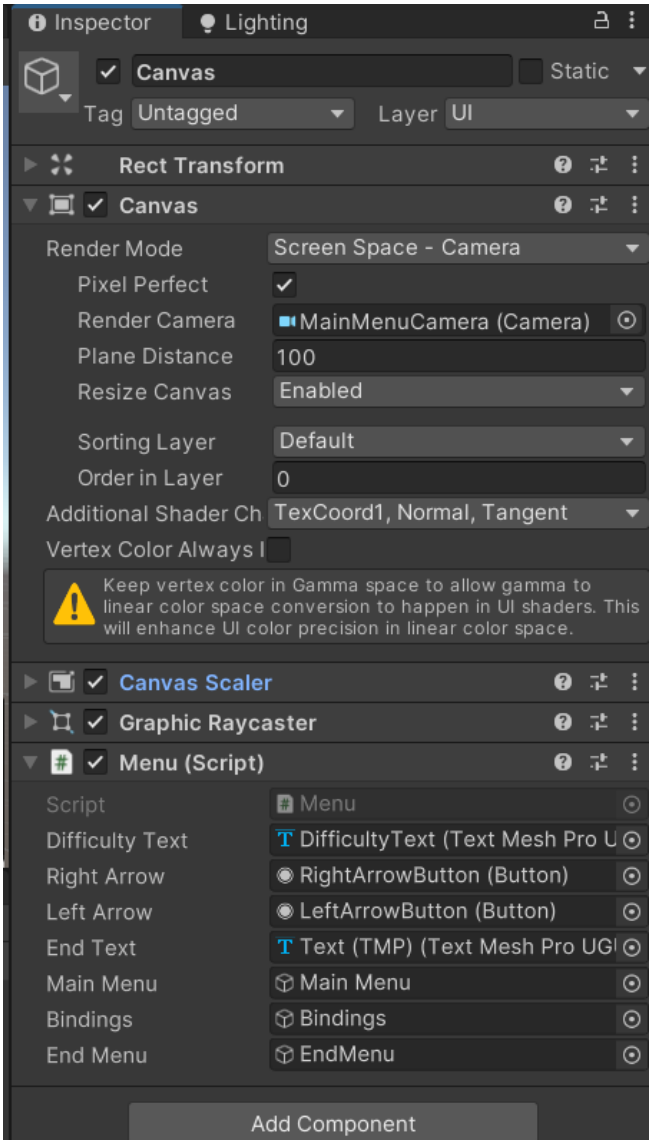


Рисунок 1.49. Inspector об’єкту Canvas

Також для канви був створений скрипт Menu (див. рис. 1.49), де будуть всі методі для роботи меню. У ньому через атрибут [SerializeField][6] задані поля на компоненти інтерфейсу з меню:

- DifficultyText — текстовий елемент, що відображає поточний рівень складності.
- RightArrow і LeftArrow — кнопки для перемикання рівня складності вліво або вправо.
- EndText — текст, який з’являється після завершення гри.
- MainMenu, Bindings, EndMenu — об'єкти меню, які активуються або приховуються.

```
public class Menu : MonoBehaviour
{
```

```
    [SerializeField]
    private TMP_Text DifficultyText;

    [SerializeField]
    private Button RightArrow;

    [SerializeField]
    private Button LeftArrow;

    [SerializeField]
    private TMP_Text EndText;

    [SerializeField]
    private GameObject MainMenu;

    [SerializeField]
    private GameObject Bindings;

    [SerializeField]
    private GameObject EndMenu;
}
```

Далі у канві було створено UI[6] об’єкт Panel, що слугує заднім фоном. Щоб додати обрану картинку на задній фон її потрібно зробити 2-D спрайтом, після цього картинку можна додати у налаштування Panel у параметр Source Image. Для затемнення зображення параметр **Color** був встановлений у значення **#767676**. Налаштування наведені на рисунку 1.50.

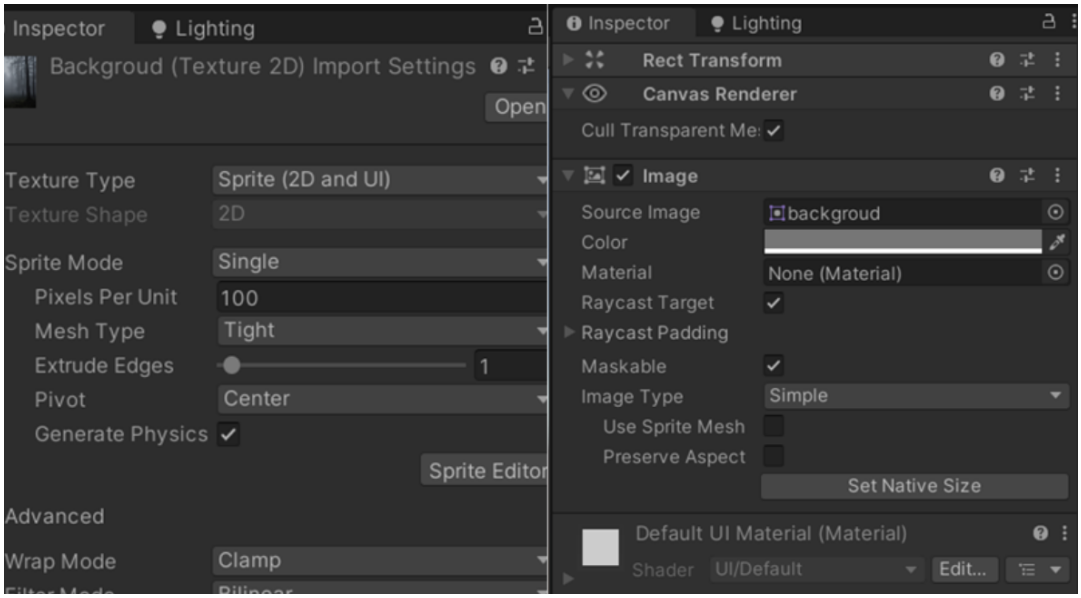


Рисунок 1.50. Inspector об’єкту Background

Потім додається у канву додається пустий об’єкт з назвою MainMenu. Туди додається текст DifficultyText по середині (див. рис. 1.52), який показує складність гри і кнопки RightArrowButton і LeftArrowButton по бокам у вигляді намальованих білих стрілок (див. рис. 1.51). Ці елементи передаються у скрипт Menu.

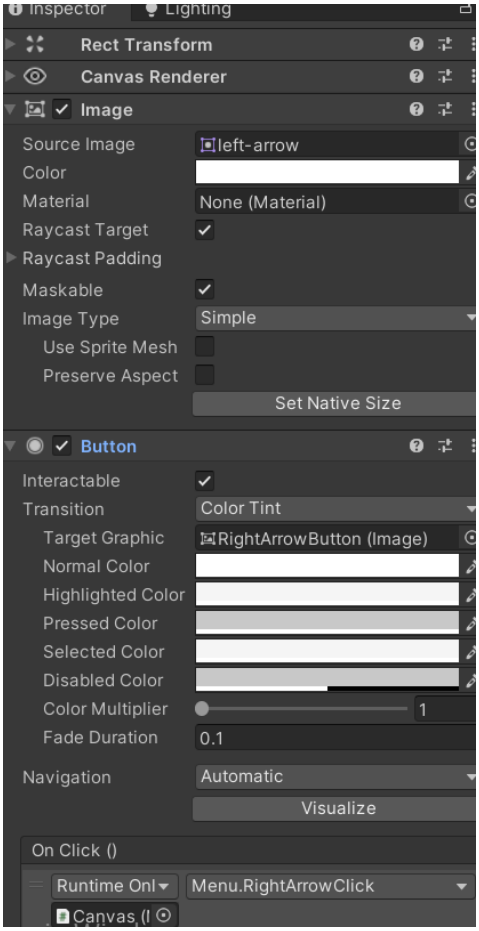


Рисунок 1.51. Inspector об’єкту LeftArrowButton

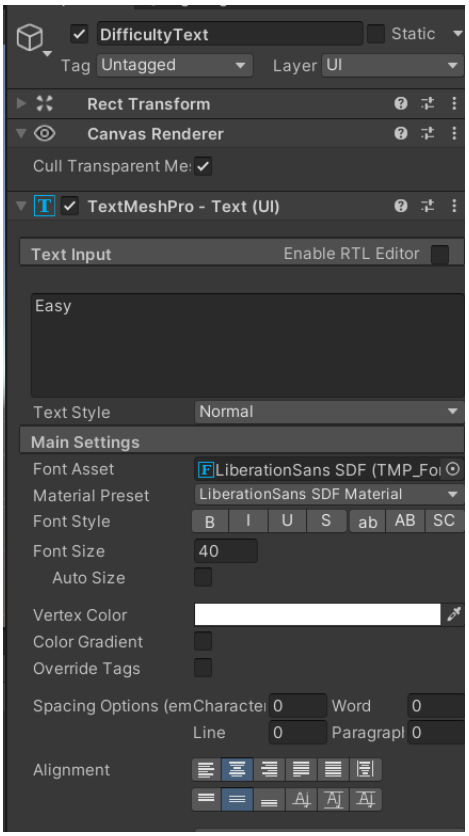


Рисунок 1.52. Inspector об’єкту DifficultyText

При натисненні на стрілки викликається подія OnClick на яку спрацьовує метод LeftArrowClick() або RightArrowClick().

Метод LeftArrowClick спрацьовує на натискання лівої стрілки. Спочатку для змінної difficulty вираховується рівень складності шляхом віднімання одиниці від поточного значення поля DifficultyType класу GameInfo. Потіму у іншій змінній це число конвертується у DifficultyType і його опис виводиться у DifficultyText. Якщо був обраний рівень складності `Easy`, то ліва стрілка деактивується, а права залишається активною. Після цього новий рівень складності зберігається у змінну `DifficultyType` класу GameInfo.

```
public void LeftArrowClick()
    {
        int difficulty = (int)GameInfo.GameInfoInstance.DifficultyType-1;
        DifficultyType difficultyType = (DifficultyType)difficulty;
        DifficultyText.text = difficultyType.ToString();
        if (difficultyType == DifficultyType.Easy)
        {
            RightArrow.interactable = true;
            LeftArrow.interactable = false;
        }
        else
        {
            RightArrow.interactable = true;
        }
        GameInfo.GameInfoInstance.DifficultyType = difficultyType;
    }
```

Метод RightArrowClick() працює навпаки, до DifficultyType додається одиниця і якщо рівень складності Hard, то деактивується права стрілка.

```
    public void RightArrowClick()
    {
        int difficulty = (int)GameInfo.GameInfoInstance.DifficultyType+1;

        DifficultyType difficultyType = (DifficultyType)difficulty;

        DifficultyText.text = difficultyType.ToString();

        if (difficultyType == DifficultyType.Hard)
        {
            RightArrow.interactable = false;
            LeftArrow.interactable = true;
        }
        else
        {
            LeftArrow.interactable = true;
        }

        GameInfo.GameInfoInstance.DifficultyType = difficultyType;
    }
```

Нижче від напису рівня складності додані три кнопки:

1. Play - для початку гри. При натисканні на цю кнопку спрацьовує метод PLayButtonClick(). Цей метод викликає метод UpdateGameInformation з класу GameInfo і завантажує сцену з самою грою.

```
public void PLayButtonClick()
{
     GameInfo.GameInfoInstance.UpdateGameInformation();

     SceneManager.LoadScene("Scenes/Forest");
}
```

2. Options - для налаштування звуку, і Quit, щоб вийти з гри. При натисканні викликається метод BindingsButtonClick(), який деактивує MainMenu і активує меню Bindings.

```
public void BindingsButtonClick()
{
     Bindings.SetActive(true);
     MainMenu.SetActive(false);
}
```

3. Quit – для закриття гри. При натисканні викликається метод QuitButtonClick().

```
public void QuitButtonClick()
{
    Application.Quit();
}
```

На рисунку 1.53 зображено кінцевий результат інтерфейсу головного меню.

Рисунок 1.53. Інтерфейс головного меню

1.4.2 Створення Settings

Для меню Settings у канву було додано пустий об’єкт Settings (див. рис. 1.54), де було створено три слайдера і текст пояснення до них, а також кнопка для ознайомлення з клавішами гри.

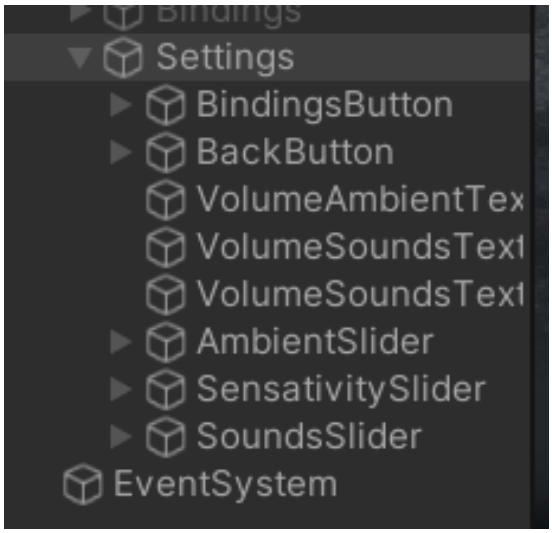


Рисунок 1.54. Об’єкт Settings у ієрархії

Також для цього меню створено скрипт Settings. Приймає у себе об’єкти класу Slider і AudioMixer і має у собі три метода.

При перетягуванні бігунку слайдера AbmientSlider викликається метод ChangeVolumeAmbient(). В ньому отимується поточне значення в змінну volume слайдера і потім викликається метод AudioMixer.SetFloat(), який змінює гучність мікшера “Ambient”.

```
public void ChangeVolumeAmbient()
 {
     float volume = VolumeAmbient.value;
     AudioMixer.SetFloat("Ambient", Mathf.Log10(volume) * 30);
 }
```

При перетягуванні бігунку слайдера SoundsSlider викликається метод ChangeVolumeSounds (). В ньому змінюєnmcz гучність мікшера “Sounds”.

```
public void ChangeVolumeSounds()
 {
     float volume = VolumeSounds.value;
     AudioMixer.SetFloat("Sounds", Mathf.Log10(volume) * 30);
 }
```

При перетягуванні бігунка слайдера SensativitySlider викликається метод ChangeCursorSensitivity(). В ньому значення CursorSensitivity.value передається у поле GameInfo.GameInfoInstance.CursorSensitivity.

```
public void ChangeCursorSensitivity()
 {
     GameInfo.GameInfoInstance.CursorSensitivity = CursorSensitivity.value;
 }
```

Вгорі зліва розташована кнопка-стрілка. При натисненні на неї викликається метод MenuButtonFromOptionsMenuClick(), яка активує MainMenu об’єкт.

```
public void MenuButtonFromOptionsMenuClick()
{
    Options.SetActive(false);
    MainMenu.SetActive(true);
}
```

Нижче на рисунку 1.55 зображено інтерфейс вікна з налаштуваннями.

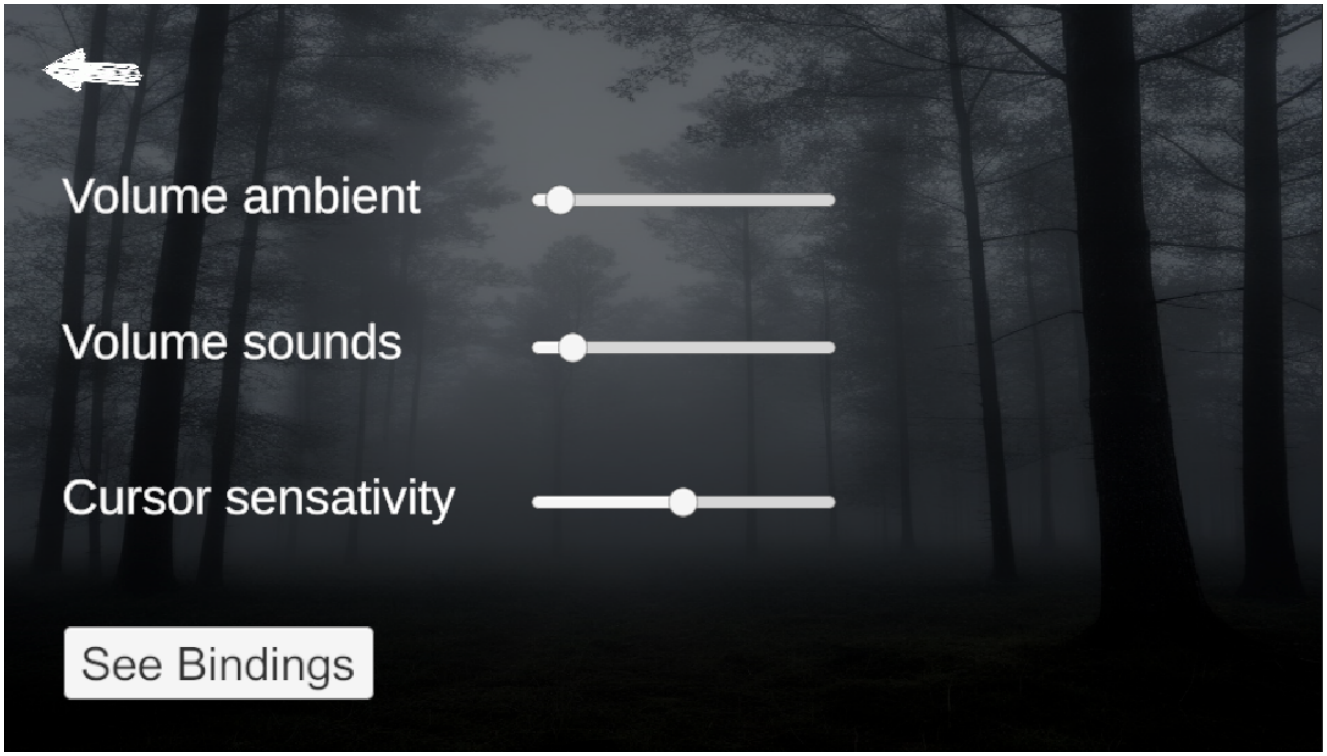


Рисунок 1.55. Інтерфейс меню налаштувань

1.4.3 Створення Bindings

Далі у канві було створено третій пустий об’єкт з назвою Bindings, у який додано усі кнопки і написи. Це меню потрібне для ознаймлення гравця з кнопками, які використовуються у грі. Клавіші оформлені у вигляді білих кнопок, що нагадують справжні клавіші на клавіатурі, а поряд розташований текст з поясненням дії.

Вгорі зліва розташована кнопка-стрілка. При натисненні на неї викликається метод BackToOptionsClick(), яка активує Options меню.

```
public void BackToOptionsClick()
 {
     Bindings.SetActive(false);
     Options.SetActive(true);
 }
```

На рисунку 1.56 зображено інтерфейс вікна Bindings.

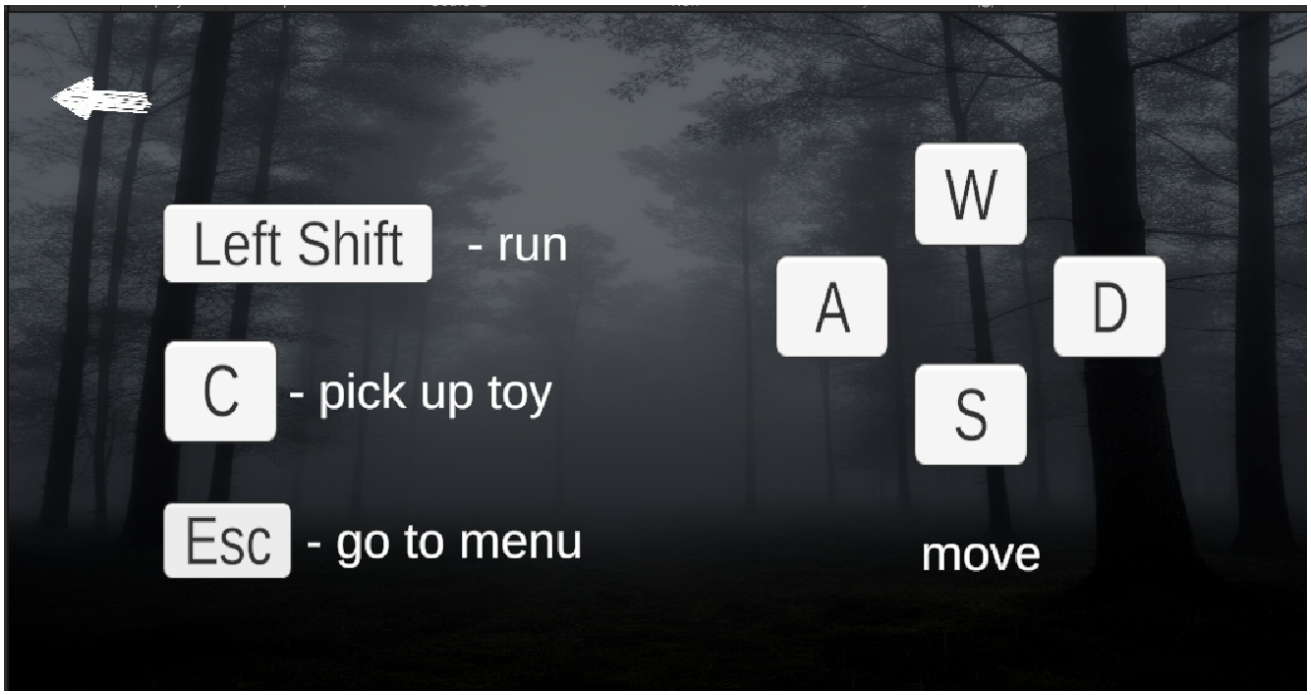


Рисунок 1.56. Інтерфейс Bindings

1.4.4 Створення EndMenu

Далі у канві було створено четвертий пустий об’єкт з назвою EndMenu, у який було додано лише текст EndText, який передається у скрипт Menu, і кнопку MenuButton. Це меню з’являється після завершення гри і показує текст “Win” або “Failed”.

У скрипті Menu у методі Start() спочатку розблоковується курсор, а потім перевіряється поле GameInfo.GameInfoInstance.IsGameOver і якщо значення true, викликається метод ActivateEndMenu().

```
void Start()
{
    GameInfo.GameInfoInstance.DifficultyType = DifficultyType.Easy;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    if (GameInfo.GameInfoInstance.IsGameOver)
    {
        ActivateEndMenu();
    }
}
```

У методі ActivateEndMenu() активується EndMenu і у поле EndText передається текст і колір тексту з класу EndTextManager.

```
public void ActivateEndMenu()
{
    EndMenu.SetActive(true);
    MainMenu.SetActive(false);
    EndText.text = EndTextManager.EndTextInstance.Text;
    EndText.color = EndTextManager.EndTextInstance.Color;
}
```

Внизу тексту розташована кнопка, щоб повернутись до головного меню. При натисненні на неї викликається метод MenuButtonFromEndMenuClick(), яка активує MainMenu.

```
public void MenuButtonFromEndMenuClick()
 {
     EndMenu.SetActive(false);
     MainMenu.SetActive(true);
 }
```

На рисунку 1.57 зображено інтерфейс вікна, яке з’являється після закінчення гри.

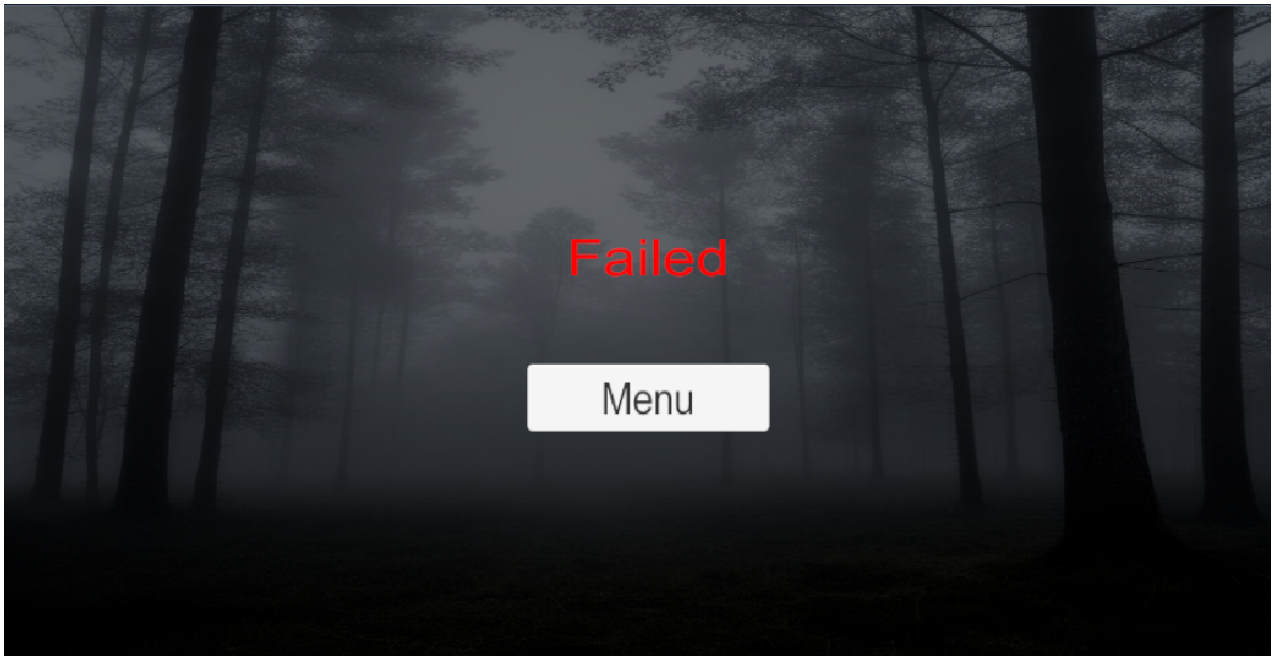


Рисунок 1.57. Інтерфейс екрану завершення гри

1.4.4 Створення інтерфейсу під час гри

Для інтерфейсу під час гри у сцені Forest було створено Canvas (див. рис. 1.58) і налаштовано головну камеру так само як і у сцені Menu. У правий верхній кут додано два TMP_Text з назвою SumText, який відображає суму іграшок, і CountText, який відображає скільки іграшок було зібрано під час гри. Також внизу

канви додано TMP_Text з назвою PressC, але в звичайному стані текст не активний і його не видно на екрані.

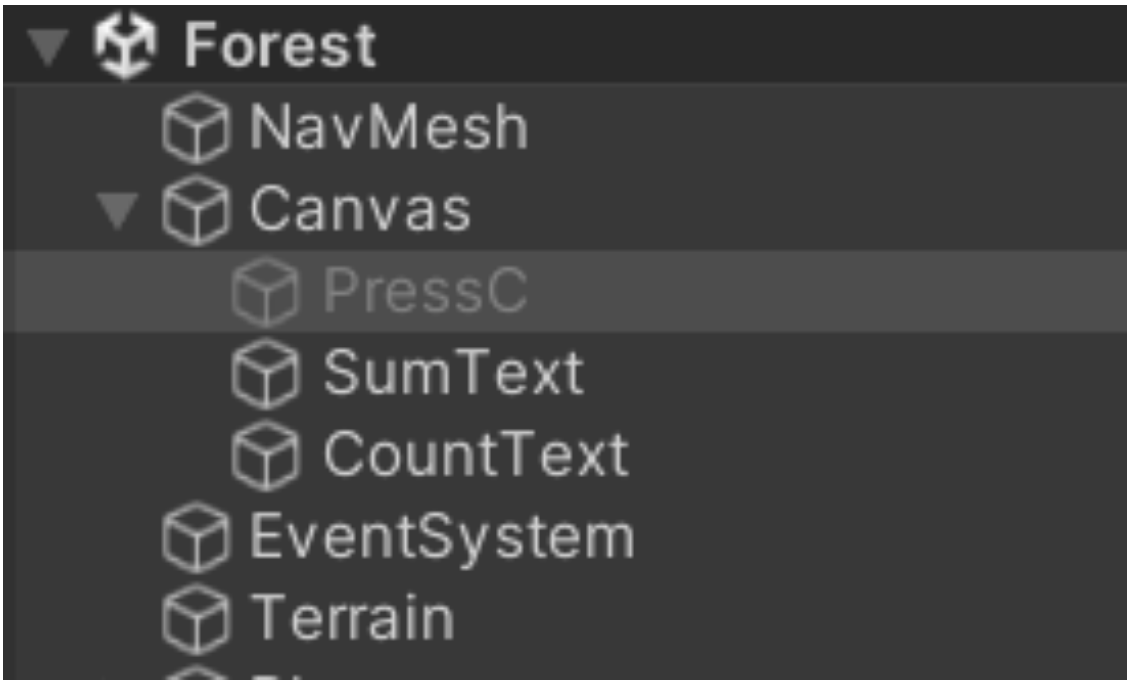


Рисунок 1.58. Об’єкт Canvas у ієрархії

Саме ці три об’єкти передаються у скрипт CreateObjectManager. CountText і PressC записуються у TextManager.TextManagerInstance і використовуються в інших скриптах, які були описані у пункті НОМЕРПУНКТА. В SumText виводиться значення змінної GameInfo.GameInfoInstance.SumOfToys.

```
public class CreateObjectsManager : MonoBehaviour
{
    [SerializeField]
    private TMP_Text sumOfToys;

    [SerializeField]
    private TMP_Text counter;

    [SerializeField]
    private TMP_Text pressC;

    [SerializeField]
    private GameObject toyPrefab;

    void Start()
    {
        TextManager.TextManagerInstance.CounterText = counter;
        TextManager.TextManagerInstance.PressC = pressC;
        sumOfToys.text = '/'+GameInfo.Instance.SumOfToys.ToString();
        CreateToysFromJson(GameInfo.Instance.SumOfToys);
   }
}
```

На рисунку 1.59 зображено інтерфейс під час гри.

Рисунок 1.59. Інтерфейс під час гри

Фіналізуючи, можна сказати, що гра була успішно протестована з різних аспектів та довела працездатність реалізованого запланованого функціоналу у повному обсязі, довевши ефективність обраних технологій та підходів.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

У дипломному проєкті створено гру "Grey Forest" у жанрі horror-survival на рушії Unity з реалізованою логікою гравця, ворога, кат-сценами, звуками та інтерфейсом. Якість програмного продукту оцінюється з погляду користувача, ефективності використання ресурсів та відповідності вимогам. Далі буде проведено розрахунок трудомісткості розробки гри.

2.2 Визначення трудомісткості розробки ПЗ

2.2.1 Визначення трудомісткості розробки програмного забезпечення.

Тривалість розробки залежить від обсягу, трудомісткості, кваліфікації виконавців і ринкових термінів. Обсяг програмного забезпечення оцінюється методом структурної аналогії за допомогою таблиці аналогів (табл. 2.1), де всі необхідні значення виділено сірим.

Таблиця 2.1 Каталог аналогів

Найменування ПП	**Обсяг функції ПП – V_o, умов. машинних командах.**
6. ПП автоматизованих розрахунків	1300 – 8600
7. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
8. ПП організації обчислювального процесу	13000 – 10200

Вибравши аналог ПП, що містить V_o в умовних машинних командах, трудомісткості визначати на основі табл. 2.2

Таблиця.2.2. Обсяг умовних команд

Обсяг ПП, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244

На основі отриманих даних за таблицею 2.2 визначається укрупнена норма часу, скоригована поправочним коефіцієнтом Кк=0,7÷0,8 з урахуванням умов розробки:

$$T^a p = \quad 229*0,7 = 160,3 \text{ (люд/годин)} \qquad (2.1)$$

Трудомісткість визначається окремо для кожного етапу розробки з урахуванням складності, новизни та використання стандартних модулів за відповідними формулами:

Розробка технічного завдання

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.2)$$

Розробка технічного проекту

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.3)$$

Розробка робочого проекту

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.4)$$

Для розрахунку необхідні наступні коефіцієнти:

- Li – питома вага і-го етапу розробки (див. табл. 2.3.);
- Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);
- Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5.).

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	**Ступінь новизни**		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	**Ступінь новизни**	**Значення $K_н$**
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Тому що розробка системи є ПО, що має аналоги програмних продуктів, то код ступеня новизни для мого ПО – В, а значення коефіцієнта $К_н$= 0,7. По таблиці 2.4, знаючи код ступеня новизни, тепер можна визначити значення питомих

коефіцієнтів трудомісткості (по таблиці 2.3): L_1=0,12; L_2=0,11; L_3=0,61.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	**Значення $К_т$**
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

У розробленому програмному продукті використовується від 40 до 60 відсотків існуючих функцій, це значить, що $К_т$= 0.7. Тепер потрібно розрахувати трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз}=T^a p *L_1*К_н= 160,3*0,12*0,7=13,47 \text{ (люд/годин)} \quad (2.5)$$

Трудомісткість розробки технічного проекту

$$T_{тп}=T^a p *L_2*К_н = 160.3*0,11*0,7=12,34 \text{ (люд/годин)} \quad (2.6)$$

Трудомісткість розробки робочого проекту

$$T_{рп}=T^a p *L_3*К_н*К_т= 160.3*0,61*0,7*0,7=47,91 \text{ (люд/годин)} \quad (2.7)$$

Для подальших розрахунків необхідно визначити кількість папера, витраченого на кожен етап. $N_{тз}$= 2 (стр), $N_{тп}$= 3 (стр), $N_{рп}$= 51 (стр), $N_{пз}$= 54 (стр) – технічне завдання, розробка технічного проекту, розробка робочого проекту, пояснювальна записка відповідно. Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, години.**		
	Розробка ПП	Контроль керівника	Нормоконтроль
1.ТЗ	$Т_{Ртз}$=13,47	$Т_{кк}$=0,7*$N_{тз}$= 0,7*2=1,4	$Т_{нк}$=0,15*$N_{тз}$= 0,15*2=0,3
2.Розробка ТП	$Т_{Ртп}$=12,34	$Т_{кк}$=0,7*$N_{тп}$= 0,7*3=2,1	$Т_{нк}$=0,15*$N_{тп}$= 0,15*3=0,45
3.Розробка РП	$Т_{Ррп}$=47,91	$Т_{кк}$=0,7*$N_{рп}$= 0,7*51=35,7	$Т_{нк}$=0,15*$N_{рп}$= 0,15*51=7,65
4.Розробка пояснювальної записки	$Т_{рпз}$=1,5*$N_{пз}$= 1,5*54=81	$Т_{кк}$=0,7*$N_{пз}$= 0,7*54=37,8	$Т_{нк}$=0,15*$N_{пз}$= 0,15*54=8,1

Продовження таблиці 2.6

Усього, в т.ч.:	$T_{пп}$=248,22		
- на розробку	$T_р$=154,72		
- контроль керівника		$T_{кк}$=77	
- нормоконтроль			$T_{нк}$=16,5

2.2.2 Розрахунок ціни програмного продукту

У цьому розділі розраховується вартість розробки ПЗ, зокрема заробітна плата виконавців (табл. 6.7), матеріальні витрати та інші витрати з урахуванням мінімальної зарплати 8000 грн і ставки 46 грн згідно з Законом про Держбюджет-2024.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців.

Найменування робіт	**Трудомісткість робіт, роб.години**	**Годинна тарифна ставка,грн..**	**Розрахунок, грн.**
1.Розробка ПП	$T_р$=154,72	48,00	7426,56
2.Контроль керівника	$T_{кк}$=77	60,00	4620,00
3.Нормоконт-роль	$T_{кк}$=16,5	60,00	990,00
Усього ($З_о$)	-	-	Зо= 13036,56

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	**Тип, модель**	**Кількість, шт**	**Ціна одиниці, грн.**	**Вартість, грн.**
Папір	Лист А4	93	5,00	465,00
Разом	-	-	-	$В_{м\,і}$=465,00
Транспортно – заготівельні Витрати 10%				$Втр\_з = 0{,}1 \times Вм1$ = 0,1*465=46,5
Усього				$В_м=В_{м1}+В_{тр\_з}$= 465+46,5=511,5

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	**Значення, грн.**	**Формула розрахунку**
1. Матеріали	511,5	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	13036,56	$З_о$ (див. табл. 2.7)
3.Додаткова заробітна плата	1303,66	$Зд = 0{,}1 \times Зо =$ 0,1*13036,56
4.Відрахування до єдиного фонду соціального внеску	3154,85	$Вс.с.в. = 0{,}22 \times (Зо + Зд) =$ 0,22*(13036,56+1303,66)
5. Накладні витрати	7821,94	$Внак. = 0{,}6 \times Зо =$ 0,6*13036,56
6. Повна собівартість	25828,51	$С_{пов}=В_м+З_о+З_д+Вє.с.в.+Внак.=$ 511,5+13036,56+1303,66+ 3154,85+7821,94

Розмір прибутку, що включається в ціну, визначається по наступній формулі:

$$П=(С_{пов}*Р)/100 = (25828,51*10)/100=2582,85 \text{ (грн)}, \quad (2.8)$$

де Р– плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о=С_{пов}+П=25828,51+2582,85=28411,36 \text{ (грн)}; \quad (2.9)$$

Податок на додану вартість визначається по наступній формулі:

$$ПДВ=0.2*Ц_о=0,2*28411,36=5682,27 \text{ (грн)}; \quad (2.10)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р=Ц_о+ПДВ= 28411,36+5682,27 = 34093,63 \text{ грн}; \quad (2.11)$$

.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Основні положення

Охорона праці потрібна для захисту здоров'я працівників і для створення безпечних умов праці. Розробка даного проєкту проводилася за комп'ютером, тому у цьому розділі розглядаються застосовані норми і рекомендації, що стосуються комп'ютеру, для облаштування комфортного робочого місця. Було розглянуто основні вимоги до освітлення, мікроклімату, ергономіки, електро- та пожежної безпеки, шумового навантаження та інших факторів, які були враховані при організації робочого середовища з комп'ютером.

3.2 Негативні фактори під час роботи за комп'ютером

Робота за комп'ютером має багато негативних наслідків для психіки і для фізіології людини. Сидіння в одній позі тривалий час викликає біль у спині і шиї і проблеми з циркуляцією крові у всьому організмі, що порушує живлення тканин, пошкоджує стінки судин і призводить до їх розширення. Перевантаження одних і тих самих груп м'язів призводить до захворювань суглобів і м'язів, а зниження загальної фізичної активності - до ожиріння та порушення обміну речовин. Монітор викликає напруження очей, під час читання з нього відстань від тексту до людини постійно залишається однаковою, через це м'язи очей перебувають у постійній напрузі. З часом це може призвести до зниження зору.

При тривалій роботі за комп'ютером нерідко розвивається розумова втома і порушення уваги. Крім того, тривале перебування у віртуальному середовищі знижує рівень соціальної взаємодії, що може призводити до емоційного виснаження та відчуття ізоляції. Використання комп'ютера у вечірній час погіршує якість сну через вплив синього світла на вироблення мелатоніну.

3.3 Гігієнічні норми

3.3.1 Вимоги до приміщення

Для комфортного та безпечного розміщення обладнання було влаштоване робоче місце площею 6,0 м².

В приміщеннях, де використовують комп’ютер, не можна використовувати полімерні матеріали для стін, які виділяють шкідливі хімічні речовини, тому стіни у приміщені просто пофарбовані. Колір стін приміщення також має важливу роль. Колір є засобом створення психологічного комфорту та підвищення продуктивності праці. Найбільш сприятливими є світлі, пастельні тони – зеленувато-блакитний, ясно-сірий, золотавий, а яскраві, кольори контрастні поєднання викликають втому і роздратування. У приміщенні з робочим місцем стіни були пофарбовані у пастельний зелений колір.

Покриття підлоги у приміщенні матове, неслизьке і з антистатичними властивостями.

3.3.2 Вимоги до робочого місця

Робочий стіл відповідає вимогам ергономіки і забезпечує оптимальне розміщення обладнання на робочій поверхні. Висота столу з комп’ютером становить 680 мм, а ширина та глибина розраховані таким чином, щоб забезпечити можливість доступу до обладнання у межах досяжності. Під столом було влаштовано простір для ніг з висотою не менше ніж 600 мм, шириною не менше 500 мм і глибиною не менше 450 мм.

Робочий стілець був підібраний з регулюванням за висотою, кутом нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння. Також поверхня сидіння є плоскою, а передній край - заокругленим. Поверхня сидіння і спинки стільця напівм’яка з нековзним, повітронепроникним покриттям, що легко очищається і не електризується.

Монітор на робочій поверхні був розташовуваний на відстані 600-700мм від користувача. Його положення забезпечує зручність зорового споглядання у вертикальній площині під кутом 30° до нормалі.

Клавіатура була розташовувана на відстані 100-300мм від краю. Висота середнього рядка клавіш не перевищує 30мм. Поверхня клавіатури матова з коефіцієнтом відбиття 0,4.

Розташування пристрою введення-виведення інформації організовано так, щоб забезпечити добру видимість екрана персонального комп’ютера, зручність

ручного керування в зоні досяжності моторного поля на висоті 900–1300 мм і в межах ширини 100–500 мм.

Всі параметри можна побачити на рисунку 3.1.

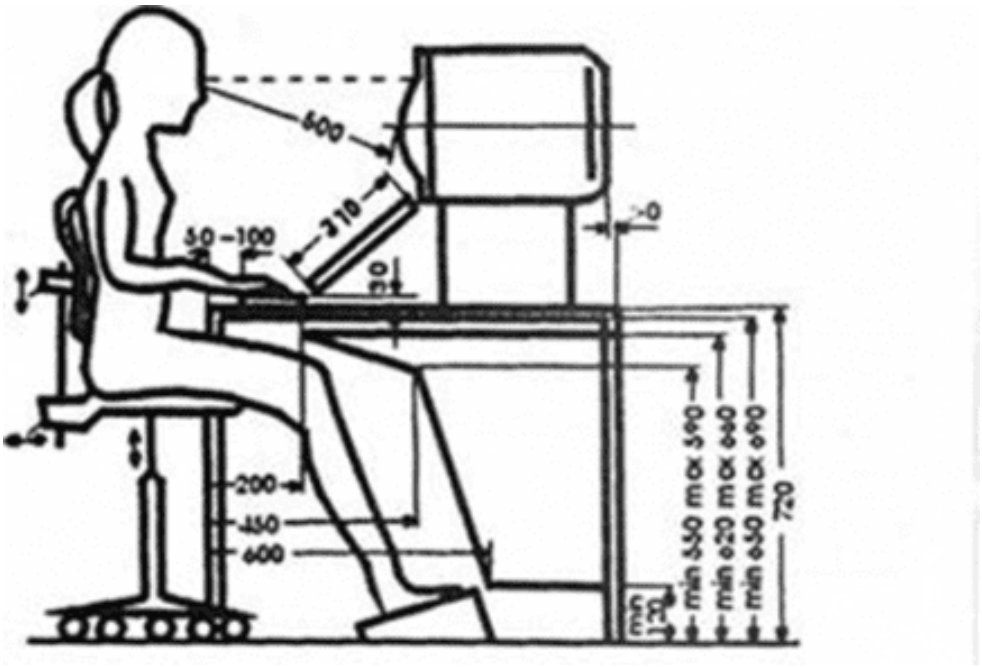


Рисунок 3.1. Параметри для робочого місця

3.3.3 Вимоги до освітлення

Природне освітлення забезпечує коефіцієнт природної освітленості не нижче 1,5%. Для регулювання рівня освітлення природним світлом передбачено використання жалюзі. Робоче місце, обладнане комп'ютером розташовуване так, щоб пряме сонячне світло не попадало в очі. Штучне освітлення приміщення було роблено у вигляді системи загального рівномірного освітлення. Заборонено застосування світильників без розсіювачів та екрануючих сіток. Рівень освітленості на робочому столі становить 300–500 лк.

3.3.4 Вимоги до шуму і вібрацій

Під час роботи рівень звуку не перевищував 50дБА. Такий рівень шуму забезпечує комфортні умови для тривалої розумової праці і не знижує концентрацію уваги. Перевищення допустимих рівнів шуму може знижувати продуктивність праці, роздратування, погіршення пам'яті та підвищення стомлюваності.

3.3.5 Вимоги до мікроклімату

Приміщення з персональними комп'ютерами було обладнане системами опалення і кондиціонування повітря. Оптимальні значення параметрів мікроклімату у приміщенні становить 22–25°С температури повітря , 40–60% вологості повітря, і швидкості руху повітря — не більше 0,1 м/с. Також приміщення провітрюється кожні 2 години.

3.3.6 Електробезпека

Під час роботи з електроприладами необхідно дотримуватися певних правил, щоб уникнути ураження електричним струмом та іншого. Перед початком роботи обов'язкова перевірка справності електроприладів, цілісність їх проводів, вилок, розеток та корпусів приладів. Такі несправності можуть становити серйозну загрозу життю і здоров'ю. Після завершення роботи всі електроприлади вимикаються з мережі, щоб запобігти коротким замиканням або перегріванню. Не допускається дотик до електроприладів вологими або мокрими руками, оскільки вода є провідником електричного струму і може спричинити ураження. У разі виявлення будь-якої несправності електроприладу його негайно вимикають з мережі та віддають на ремонт або вилучають з користування.

3.4 Пожежна безпека

Для запобігання пожежі були проведені відповідні інструктажі з пожежної безпеки і виконувалися правила електробезпеки наведені вище. Також уникалися перевантаження електромережі, не використовувалися несправні або пошкоджені прилади, не залишалося увімкненим обладнання без нагляду, а також регулярно перевірялися справність розеток і подовжувачів.

У приміщенні були встановлені справні первинні засоби пожежогасіння — вогнегасники. Найбільш універсальними є порошкові або вуглекислотні вогнегасники, якими можна гасити електроприлади, тому що вони не проводять струм. Вогнегасник був легкодоступним, справним і розміщеним у зручному місці, подалі від джерел тепла. Біля нього була розміщена інструкція використання, а також перевірявся термін придатності.

Також був розроблений план евакуації працівників, який був розміщений на видимих місцях.

3.5 Основні заходи та засоби щодо збереження здоров'я

Щоб зберегти здоров'я виконувалися всі зазначені у пунктах вище правила з охорони праці. Було дотримано режиму праці і відпочинку і робилися перерви по 5-10 хвилин. Виконувати вправи для очей і розминка для тіла, а у вільний час відвідувався спортивний зал.

Для зменшення впливу синього світла на очі на моніторі комп'ютера використовувався фільтр для зору, що прибирає синє світло і робить його жовтим. На різних пристроях вони можуть називатися по різному. Наприклад, на рисунку 3.2 зображено вигляд фільтру на ноутбуці фірми Acer.

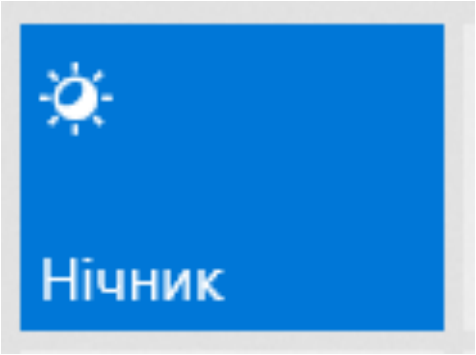


Рисунок 3.2. Нічник на ноутбуці Acer

Також кожен рік передбачено профілактичні медичні огляди у терапевта, офтальмолога, а за потреби — й інших спеціалістів. Це дозволить вчасно виявити можливі порушення здоров'я та запобігти їх подальшому розвитку.

.

ВИСНОВКИ

В результаті розробки була створена 3D гра “Grey Forest” у жанрі horror-survival. Для її створення було досліджено основні елементи даного жанру. Аналіз цих особливостей дозволив краще зрозуміти механіки, що впливають на емоційне сприйняття гри, і використати їх у власному проєкті для покращення ігрового досвіду.

Для створення напруженої хорор атмосфери у грі була створена сцена темного лісу з туманом, дивними звуками, і локаціями, щоб гравець міг орієнтуватись у лісі. Гра ведеться від першої особи для глибшого занурення в ігрове середовище гравця. Під час проходження за гравцем полює монстр, який лякає гравця і змушує його бути обережним і прислухатись до його кроків.

Для розробки використовувався ігровий рушій Unity, який ідеально підходить для подібних невеликих і незалежних проєктів, тому що є безкоштовним і має багато навчальних ресурсів. Для написання коду використовувалася мова C#, яка також має багато навчальних матеріалів і є об’єктно-орієнтованою. Для збереження інформації про гру на час проходження був використаний такий патерн програмування як Singletone, який дав змогу користуватися одної інформацією з різних скриптів проєкту. Також у процесі роботи було досліджено етапи розробки ігор від ідеї до реалізації і були отримані навички пов’язані з використанням компонентів рушія, програмуванням логіки гри, налаштуванням сцени, роботи з анімаціями, звуком та інтерфейсом.

Всі 3D моделі були взяті з unity asset store і були безкоштовні. Це допомагає зосередитись на написанні логіки та технічній реалізації проєкту.

Всі набуті навички і знання в процесі виконання диплому можуть бути використані у майбутніх розробках, вдосконалені і доповнені іншими знаннями. Цей проєкт є важливим на шляху становлення інді-розробником і розвитку у цій сфері.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ґазавей, Дакс. Вступ до дизайну ігрових систем. – Київ: Видавництво «Фабула», 2020. – 67 с.
2. Фрімен, Ерік, Робсон, Елізабет, Бейтс, Берт, Сієрра, Кеті. Head First. Патерни проектування: практичний посібник з об'єктно-орієнтованого програмування. – Київ: Видавництво «Фабула», 2021. – 201 с.
3. Horror-survival [Електронний ресурс] // Wikipedia – The Free Encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Survival_horror – Назва з екрана. – Дата звернення: 15.05.2025.
4. Outlast [Електронний ресурс] // Вікіпедія – Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Outlast – Назва з екрана. – Дата звернення: 20.05.2025.
5. Amnesia: The Dark Descent [Електронний ресурс] // Вікіпедія – Вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Amnesia:_The_Dark_Descent – Назва з екрана. – Дата звернення: 20.05.2025.
6. Unity Technologies. Unity – офіційний сайт ігрового рушія [Електронний ресурс]. – Режим доступу: https://unity.com/en-us – Назва з екрана. – Дата звернення: 04.06.2025.
7. Microsoft Corporation. C# documentation – офіційна довідка з мови програмування C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp – Назва з екрана. – Дата звернення: 06.06.2025.

ДОДАТОК А. Фрагменти коду класів для роботи меню, ворогів та камери персонажу

```
// Menu.cs
using Assets.Enumerables;
using UnityEngine.UI;
using UnityEngine;
using UnityEngine.SceneManagement;
using Assets.Logic.Scripts;
using TMPro;

namespace Assets.Scripts
{
    public class Menu : MonoBehaviour
    {
        [SerializeField]
        private TMP_Text DifficultyText;

        [SerializeField]
        private Button RightArrow;

        [SerializeField]
        private Button LeftArrow;

        [SerializeField]
        private TMP_Text EndText;

        [SerializeField]
        private GameObject MainMenu;

        [SerializeField]
        private GameObject Bindings;

        [SerializeField]
        private GameObject Options;

        [SerializeField]
        private GameObject EndMenu;

        void Start()
        {
            GameInfo.Instance.DifficultyType = DifficultyType.Easy;
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;

            if (GameInfo.Instance.IsGameOver)
            {
                ActivateEndMenu();
            }
        }

        public void LeftArrowClick()
        {
            int difficulty = (int)GameInfo.Instance.DifficultyType-1;

            DifficultyType difficultyType = (DifficultyType)difficulty;

            DifficultyText.text = difficultyType.ToString();

            if (difficultyType == DifficultyType.Easy)
            {
```

```
        RightArrow.interactable = true;
        LeftArrow.interactable = false;
    }
    else
    {
        RightArrow.interactable = true;
    }

    GameInfo.Instance.DifficultyType = difficultyType;
}

public void RightArrowClick()
{
    int difficulty = (int)GameInfo.Instance.DifficultyType+1;
    DifficultyType difficultyType = (DifficultyType)difficulty;
    DifficultyText.text = difficultyType.ToString();

    if (difficultyType == DifficultyType.Hard)
    {
        RightArrow.interactable = false;
        LeftArrow.interactable = true;
    }
    else
    {
        LeftArrow.interactable = true;
    }

    GameInfo.Instance.DifficultyType = difficultyType;
}

public void PLayButtonClick()
{
    GameInfo.Instance.UpdateGameInformation();
    SceneManager.LoadScene("Scenes/Forest");
}

public void QuitButtonClick()
{
    Application.Quit();
}

public void MenuButtonFromEndMenuClick()
{
    EndMenu.SetActive(false);
    MainMenu.SetActive(true);
}

public void BackToOptionsClick()
{
    Bindings.SetActive(false);
    Options.SetActive(true);
}

public void MenuButtonFromOptionsMenuClick()
{
    Options.SetActive(false);
    MainMenu.SetActive(true);
}

public void ActivateEndMenu()
{
    EndMenu.SetActive(true);
    MainMenu.SetActive(false);
    EndText.text = EndTextManager.Instance.Text;
```

```
            EndText.color = EndTextManager.Instance.Color;
        }

        public void BindingsButtonClick()
        {
            Bindings.SetActive(true);
            Options.SetActive(false);
        }

        public void OptionsButtonClick()
        {
            Options.SetActive(true);
            MainMenu.SetActive(false);
        }
    }
}

//Enemy.cs

using Assets.Logic.Scripts.Wrappers;
using Assets.Scripts;
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

public class Enemy : MonoBehaviour
{
    [SerializeField]
    private Transform player;

    [SerializeField]
    private LayerMask whatIsPlayer;

    private Animator animator;

    private AudioSource audioSource;

    private NavMeshAgent agent;

    private float speed = GameInfo.Instance.EnemyWalkSpeed;

    private float runSpeed = GameInfo.Instance.EnemyRunSpeed;

    private float sightRange = GameInfo.Instance.SightRange;

    private bool playerInSightMesh;

    private Vector3 walkPoint;

    private bool walkPointIsSet;

    private bool isCoroutineEnd;

    private ArrayWalkPointWrapper arrayWalkPointWrapper = new ArrayWalkPointWrapper(29);

    void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        agent.speed = speed;
        animator = GetComponent<Animator>();
        audioSource = GetComponent<AudioSource>();
        arrayWalkPointWrapper.ReadJson("walkPoints.json");
    }
    void Update()
```

```
    {
        animator.SetFloat("Moving", 1);
        playerInSightMesh = Physics.CheckSphere(transform.position, sightRange,
whatIsPlayer);


        if (playerInSightMesh)
        {
            ChasePlayer();
        }

        else
            Patroling();
    }

    private void Patroling()
    {
        audioSource.pitch = 1.2f;
        agent.speed = speed;

        if (!walkPointIsSet)
        {
            audioSource.Pause();
            animator.SetFloat("Moving", 0);
            StartCoroutine(Wait());

            if (isCoroutineEnd)
            {
                SetWalkPoint();
                isCoroutineEnd = false;
            }
        }
        else
        {
            agent.SetDestination(walkPoint);
            animator.SetFloat("Moving", 1);
        }

        if (!audioSource.isPlaying)
        {
            audioSource.Play();
        }

        var enemyPosition = transform.position;
        enemyPosition.y = 0;

        if (Vector3.Distance(walkPoint, enemyPosition) < 1)
        {
            walkPointIsSet = false;
        }
    }

    private void ChasePlayer()
    {
        agent.speed = runSpeed;
        audioSource.pitch = 1.5f;

        if (!audioSource.isPlaying)
        {
            audioSource.Play();
        }

        transform.LookAt(player);
        agent.SetDestination(player.position);
```

```
        animator.SetFloat("Moving", 2);
    }

    private void SetWalkPoint()
    {
        animator.SetFloat("Moving", 1);
        int walkPointIndex = Random.Range(0, arrayWalkPointWrapper.WalkPoints.Length -
1);

        walkPoint.x = arrayWalkPointWrapper.WalkPoints[walkPointIndex].x;
        walkPoint.y = 0;
        walkPoint.z = arrayWalkPointWrapper.WalkPoints[walkPointIndex].z;
        walkPointIsSet = true;
    }

    private IEnumerator Wait()
    {
        animator.SetFloat("Moving", 0);
        yield return new WaitForSecondsRealtime(5);
        isCoroutineEnd = true;
    }
}

// PlayerCamera.cs

using Assets.Scripts;
using UnityEngine;

public class PlayerCamera : MonoBehaviour
{
    [SerializeField]
    private GameObject flashlight;

    [SerializeField]
    private Transform orientation;

    private float cursorSensitivity = GameInfo.Instance.CursorSensitivity;

    private float rotationX = 1.0f;
    private float rotationY = 1.0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    // Update is called once per frame
    void Update()
    {
        var mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * cursorSensitivity;
        var mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * cursorSensitivity;

        rotationY += mouseX;
        rotationX -= mouseY;

        rotationX = Mathf.Clamp(rotationX, -90f, 90f);

        orientation.rotation = Quaternion.Euler(0, rotationY, 0);

        transform.rotation = Quaternion.Euler(rotationX, rotationY, 0);
        flashlight.transform.rotation = Quaternion.Euler(rotationX, rotationY, 0);

        transform.position = orientation.position;
```

```
    }
}
using Assets.Scripts;
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

namespace Assets.Logic.Scripts
{
    class Settings : MonoBehaviour
    {
        [SerializeField]
        private Slider VolumeAmbient;

        [SerializeField]
        private Slider VolumeSounds;

        [SerializeField]
        private Slider CursorSensitivity;

        [SerializeField]
        private AudioMixer AudioMixer;

        private void Start()
        {
            ChangeVolumeAmbient();
            ChangeVolumeSounds();
        }

        public void ChangeVolumeAmbient()
        {
            float volume = VolumeAmbient.value;
            AudioMixer.SetFloat("Ambient", Mathf.Log10(volume) * 30);
        }

        public void ChangeVolumeSounds()
        {
            float volume = VolumeSounds.value;
            AudioMixer.SetFloat("Sounds", Mathf.Log10(volume) * 30);
        }

        public void ChangeCursorSensitivity()
        {
            GameInfo.Instance.CursorSensitivity = CursorSensitivity.value;
        }
    }
}
```

ДОДАТОК Б. Слайди мультимедійної презентації

Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності

ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТКИ ГРУПИ 4РП-08: ФРОЮК ДАРИНА МАКСИМІВНИ

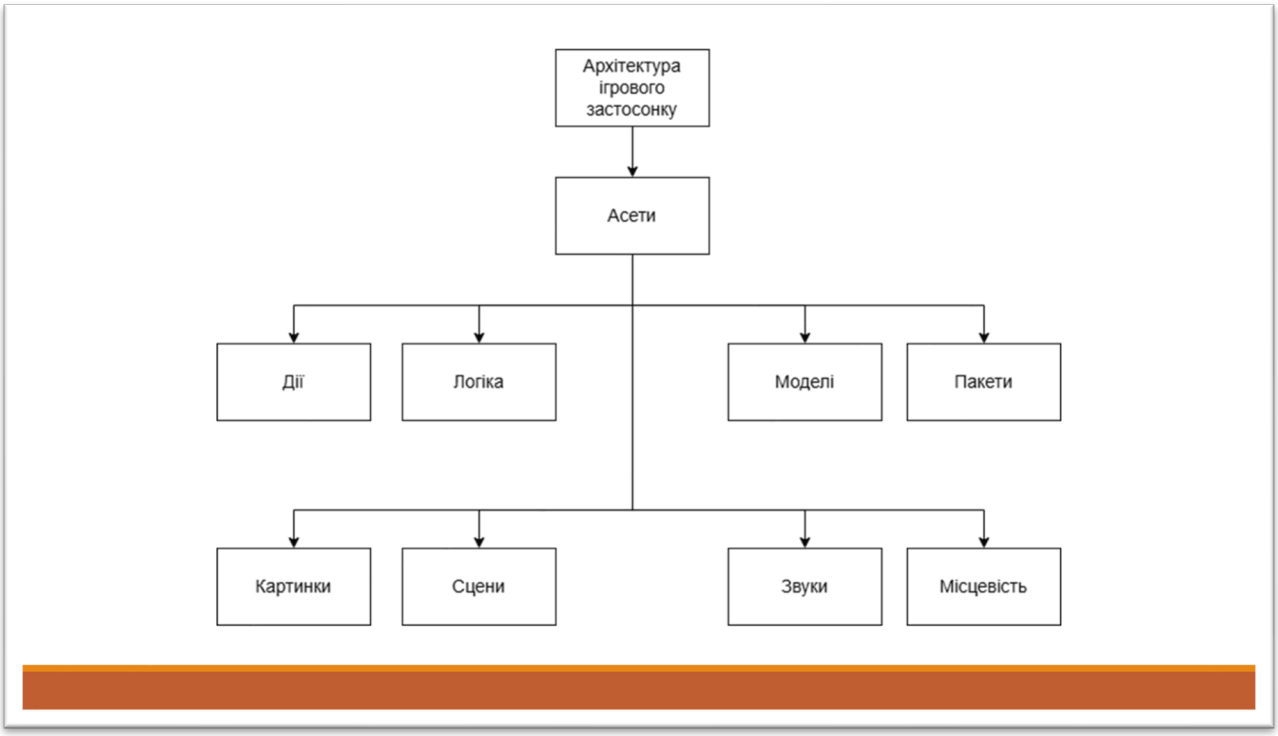
КЕРІВНИК: ЖАДАН А.С.

Використані ознаки жанру Horror-Survival у проєкті:

- ✓ Напружена атмосфера (приглушене світло, туман, локація лісу, звуки);
- ✓ Головна мета гри – вижити;
- ✓ Ворог полює на гравця протягом гри;
- ✓ Для закінчення гри гравець має зібрати певну кількість іграшок і вийти з лісу;

Етапи розробки гри:

1. Розробка ідеї гри і аналіз наявних аналогів(The eyes, Outlast, Amnesia);
2. Пошук асетів;
3. Створення сцени лісу;
4. Написання скриптів і логіки проєкту;
5. Створення інтерфейсу;


Архітектура ігрового застосонку
Асети
Дії
Логіка
Моделі
Пакети
Картинки
Сцени
Звуки
Місцевість

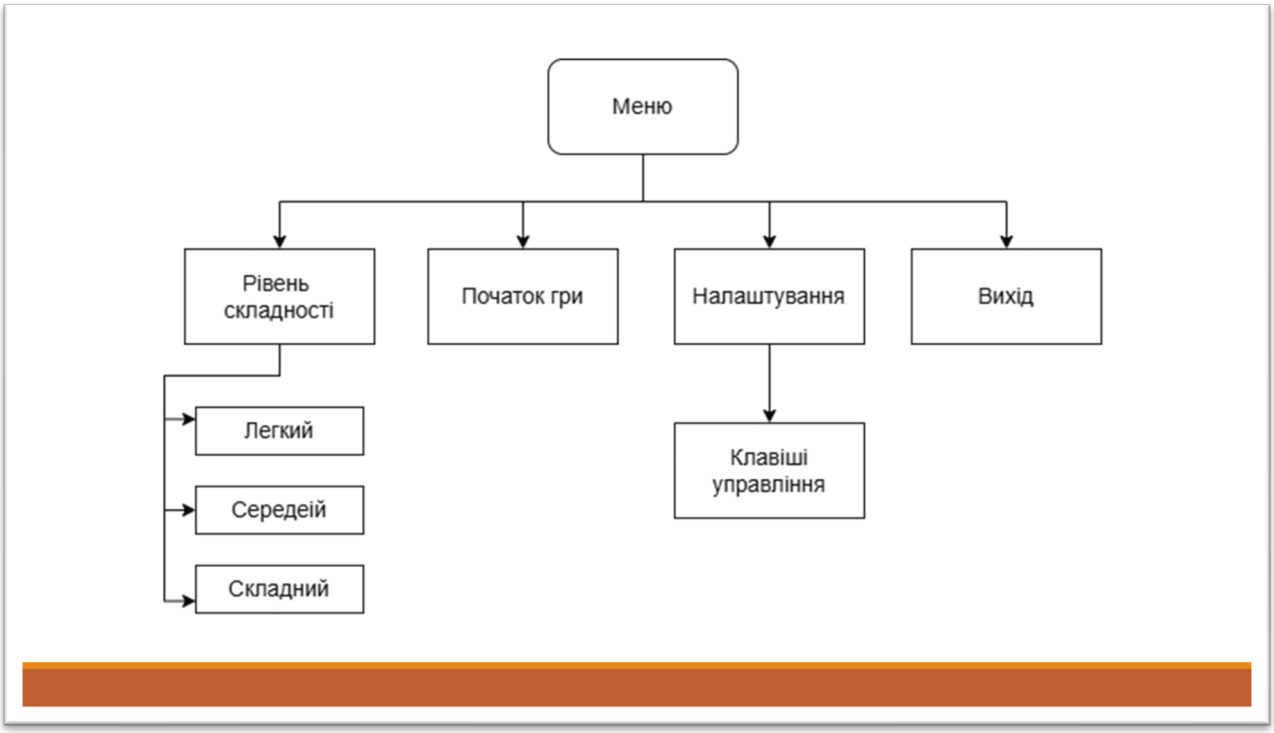
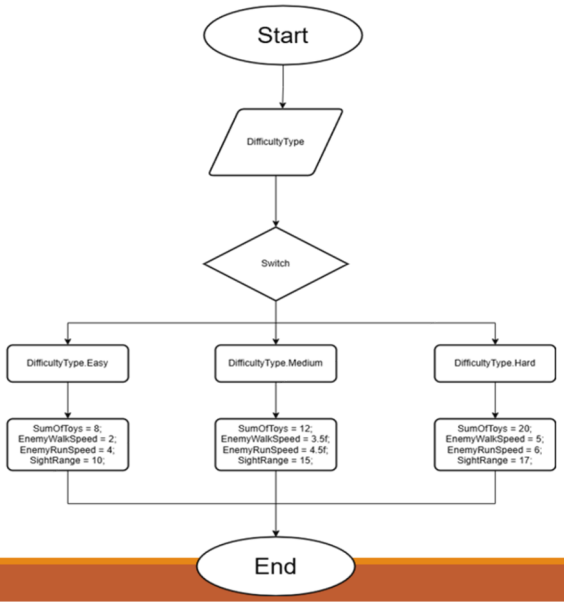
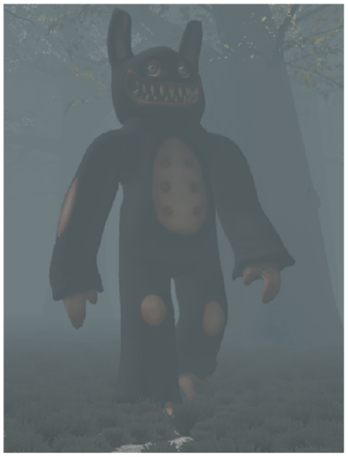

Меню
Рівень складності
Початок гри
Налаштування
Вихід
Легкий
Середеій
Складний
Клавіші управління

Схема методу UpdateGameInformation():

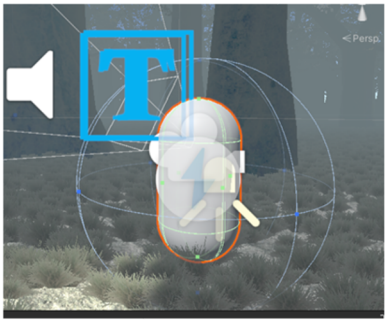


Вигляд гравця, ворога і іграшки



Вигляд ворога у грі

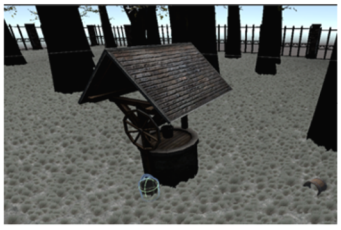
Іграшка кролик



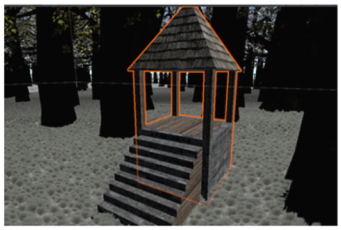
Вигляд гравця при розробці

Вигляд лісу з камери гравця

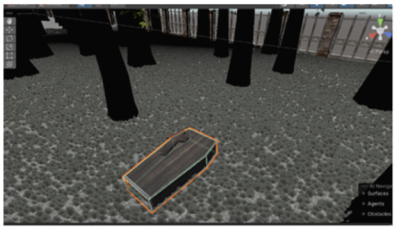
Створені локації у лісі:



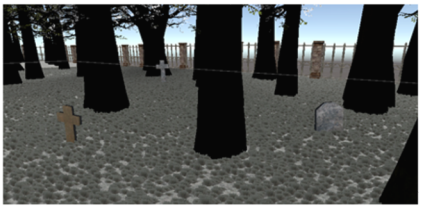
Колодязь



Каплиця



Домовина



Локація з хрестами

Тестування гри

Гра тестувалася протягом всієї розробки, тому було знайдено і виправлено дуже багато помилок. Було виправлено 3D звук кроків ворога, неправильний лічильник у верхньому кутку екрану, зникнення іграшки після її підбору гравцем, довго підбиралися правильні показники для рівнів складності, тощо.

Висновки

Під час розробки було отримано навички використання ігрового рушія Unity, його компонентів, бібліотек і асетів. Було досліджено етапи розробки ігор від ідеї до реалізації, що допоможе у майбутньому при розробці інших ігрових проєктів. Також були дослідженні основні елементи жанру horror-survival, в якому можна розватись і надалі.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Фроюк Дарини Максимівни

(прізвище, ім`я та по батькові)

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітня програма *«Розробка програмного забезпечення»*

Керівник дипломного проекту (роботи) *Жадан Артур Сергійович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка веб-застосунку контролю енергоспоживання в побутових умовах*

Обсяг розрахунково-пояснювальної записки *87* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений ігровій тематиці та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 12 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Наявність системи налаштувань, яка включає рівні гучності і демонстрацію клавіш керування.*

Реалізовано збереження і завантаження даних через JSON дерев, точок патрулювання. Розділення логіки гри по компонентах відповідає принципам ООП.

д) основні недоліки дипломного проекту

В пояснювальній записці не наведено інформації про застосування мультиплатформності рушія Unity. Недостатньо уваги приділено оптимізації (розрахунок пам'яті, швидкість зчитування, частота оновлень). Деякі недоліки оформлення пояснювальної записки та графічної частини проекту.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис

«*20*» *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проєкт здобувача (здобувачки) освіти

відділення комп'ютерних систем

Фроюк Дарина Максимівна

(прізвище, ім`я та по батькові)

Спеціальність: *121 " Інженерія програмного забезпечення"*

Освітньо-професійна програма: *«Розробка програмного забезпечення»*

Тема дипломного проєкту: *Розробка 3D-гри у жанрі survival-horror з налаштуванням рівнів складності*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЄКТУ

а) обсяг і якість виконання проєкту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проєкт виконано відповідно технічному завданню. Пояснювальна записка містить 87 сторінки. У пояснювальній записці виконано опис етапів розробки програмного забезпечення для 3D-гри у жанрі survival-horror з налаштуванням рівнів складності, а також його програмного забезпечення. Графічна частина складається з 12 слайдів мультимедійної презентації, які також містять схеми і скріншоти, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.*

б) самостійність роботи над проєктом: *Протягом всього строку дипломного проєктування та переддипломної практики здобувачка освіти Фроюк Д.М. поступово та послідовно виконувала всі етапи розробки. Всі роботи здобувачка освіти виконувала самостійно, з оглядом на рекомендації керівника*

в) теоретична підготовка випускника (випускниці): *Здобувачка освіти Фроюк Д.М. під час роботи над дипломним проєктом вивчила достатню кількість літературних джерел та матеріалів за даною тематикою.*

Вважаю, що теоретична підготовка дипломниці добра і вона готова до захисту дипломного проєкту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проєктування здобувачка освіти мала змогу самостійно приймати рішення з реалізації скриптів і елемнетів гри, що розробляла, та показала вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проєкту

Жадан Артур Сергійович

Місце роботи і посада керівника дипломного проєкту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії

Підпис

«16» грудня 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Фроюк Дарина Максимівна,
здобувачка освіти гр. 4РП-08, та

Жадан Артур Сергійович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності» (авторка роботи – Фроюк Д.М., керівник роботи – Жадан А.С.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Фроюк Д.М. /

Керівник _____ / Жадан А.С. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4РП-08

Фроюк Дарини Максимівни

на тему *Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 12.06.2025 р. значення коефіцієнту подібності в роботі становить 17,65%, коефіцієнт цитування – 0,97%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Фроюк Д.М.* (П.І.Б.)

проведена « *16* » *червня* 20*25* р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/12/2025
Дата редагування 6/12/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності

Автор
Фроюк Дарина Максимівна
Науковий керівник / Експерт
Жадан Артур Сергійович

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

25	12087	102936
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	21
Інтервали	A→	0
Мікропробіли	·	55
Білі знаки	ß	0
Парафрази (SmartMarks)	a	124

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	126 1.04 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	76 0.63 %
3	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	68 0.56 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	68 0.56 %
5	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	64 0.53 %

6	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	63 0.52 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	36 0.30 %
8	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	32 0.26 %
9	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	32 0.26 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	31 0.26 %

з домашньої бази даних (0.06 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	7 (1) 0.06 %

з програми обміну базами даних (0.40 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2021_KRB_SNs-42_Bahrii_OO_antiplagiat.doc [redacted] 6/18/2021 Ternopil Ivan Pul'uj National Technical University (кафедра комп'ютерних наук)	19 (2) 0.16 %
2	Диплом Полтарескул.docx [redacted] 6/17/2021 Odessa National Polytechnic University (ІКС, кафедра інформ. систем)	18 (2) 0.15 %
3	Розробка VR гри в середовищі Unity Engine [redacted] 6/9/2023 National University "Zaporizhzhia Polytechnic" (Кафедра "Програмні засоби")	11 (1) 0.09 %

з Інтернету (17.19 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content**	**523 (13) 4.33 %**
2	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	343 (35) 2.84 %
3	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	289 (33) 2.39 %
4	**https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download**	**117 (5) 0.97 %**
5	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	86 (8) 0.71 %
6	https://oppb.com.ua/news/gigiyenichni-vymogy-do-organizaciyi-i-obladnannya-robochyh-misc	78 (6) 0.65 %
7	https://qna.habr.com/q/1391502	74 (7) 0.61 %
8	**https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download**	**73 (2) 0.60 %**
9	**https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download**	**65 (3) 0.54 %**
10	**https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download**	**44 (2) 0.36 %**

11	https://apau.org.ua/2021/07/06/%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0-%D0%B7%D0%B0-%D0%BA%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%BC-%D0%BD%D0%B0%D1%81%D0%BB%D1%96%D0%B4%D0%BA%D0%B8-%D1%82%D0%B0-%D0%BF%D0%BE%D1%80%D0%B0/	42 (3) 0.35 %
12	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	33 (2) 0.27 %
13	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	32 (4) 0.26 %
14	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	31 (2) 0.26 %
15	**https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download**	**29 (1) 0.24 %**
16	https://docplayer.net/82533729-Aktualni-pitannya-rozvitku-ekonomiki-harchovih-tehnologiy-ta-tovaroznavstva.html	28 (4) 0.23 %
17	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	26 (2) 0.22 %
18	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	25 (2) 0.21 %
19	http://lib.kart.edu.ua/bitstream/123456789/1523/1/%D0%9E%D0%9F%20%D1%96%20%D0%9D%D0%A1%20209.pdf	23 (3) 0.19 %
20	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a05c07c5-bf65-4cb0-bdfa-e28694707551/content	22 (2) 0.18 %
21	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	22 (3) 0.18 %
22	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	22 (2) 0.18 %
23	https://studfile.net/preview/9328989/page:5/	18 (1) 0.15 %
24	https://smekni.com/a/20191-12/oblk-analz-kontrol-opodatkuvannya-pributku-pdprimstva-12/	13 (1) 0.11 %
25	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	7 (1) 0.06 %
26	https://card-file.ontu.edu.ua/server/api/core/bitstreams/fc8a1853-39fc-4671-8807-2fd27ddb0779/content	7 (1) 0.06 %
27	https://card-file.ontu.edu.ua/server/api/core/bitstreams/6eb6bf1c-5813-45e6-93c5-25539b4709d3/content	6 (1) 0.05 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: **«Розробка програмного забезпечення» Група:** 4РП- 08

Дипломний проект
здобувачки **освіти денної форми навчання РП.** 08.24.000.ДП

ФРОЮК
ДАРИНИ МАКСИМІВНИ

м. Одеса
2025 р. **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

Спеціальність: 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма: «Розробка **програмного забезпечення»**
Група: 4 РП-08