**Форма № Н-9.02**

Національний університет «Полтавська політехніка імені Юрія Кондратюка»

(повне найменування вищого навчального закладу)

\_Навчально-науковий інститут інформаційних технологій та робототехніки\_

(повна назва факультету)

\_Кафедра комп’ютерних та інформаційних технологій і систем\_

(повна назва кафедри)

**Пояснювальна записка**

**до дипломного проекту (роботи)**

\_\_\_\_\_\_\_\_\_\_\_бакалавра\_\_\_\_\_\_\_\_\_\_\_

(освітньо-кваліфікаційний рівень)

на тему

Створення гри з доповненою реальністю для мобільних пристроїв за

допомогою середовища розробки ігор Unity

Виконав: студент курсу, групи 401 ТН

спеціальності

122 Комп’ютерні науки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва напряму)

\_\_\_\_\_\_\_\_\_\_\_\_ Бондаренко Ю.С.\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_\_\_\_\_\_\_\_Ляхов О.Л.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Рецензент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Полтава – 2021 року

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА**

**Спеціальність 122 «Комп’ютерні науки»**

**на тему**

**«Створення гри з доповненою реальністю для мобільних пристроїв за допомогою середовища розробки ігор Unity»**

**Студента групи 401-ТН Бондаренка Юрія Сергійовича**

Керівник роботи

старший викладач

Ляхов О.Л.

Завідувач кафедри

кандидат технічних наук,

доцент Головко Г.В.

Полтава – 2021

**РЕФЕРАТ**

Кваліфікаційна робота бакалавра: 61 с., 33 рисунків, 7 таблиць, 10 джерел 1 додаток.

**Об’єкт дослідження:** процес розробки мобільної гри з доповненою реальністю на ігровому двигуні Unity за допомогою Vuforia Engine.

**Мета роботи:** інструментальні засоби для розробки гри з доповненою реальністю.

**Методи:** розроблення і тест мобільної гри з доповненою реальністю.

**Ключові слова:** мобільна гра, доповнена реальність, Unity, Vuforia, 3D-графіка.

**ABSTRACT**

Bachelor's qualification work: 61 pages, 33 drawings, 7 tables, 10 sources, 1 appendix.

**Object of research**: the process of developing an augmented reality mobile game on the Unity game engine with Vuforia Engine.

**Purpose:** augmented reality game development tools.

**Methods**: development and test of a mobile game with augmented reality.

**Keywords:** mobile game, augmented reality, Unity, Vuforia, 3D graphics.

ЗМІСТ

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 6](#_Toc74986565)

[ВСТУП 7](#_Toc74986566)

[РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ 9](#_Toc74986567)

[1.1 Аналіз предметної області 9](#_Toc74986569)

[1.2 Класифікація мобільних ігор 9](#_Toc74986570)

[1.2.1 За категорією 10](#_Toc74986571)

[1.2.2 За жанром 11](#_Toc74986572)

[1.3 Аналіз аналогів програмного продукту 14](#_Toc74986573)

[1.4 Постановка задачі 18](#_Toc74986574)

[РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ 20](#_Toc74986575)

[2.1 Вибір засобів розробки 20](#_Toc74986577)

[2.1.1 Вибір двигуна 21](#_Toc74986578)

[2.1.2 Вибір програмного забезпечення для створення 3D-моделей 23](#_Toc74986579)

[2.2 Принци роботи Vuforia Engine 25](#_Toc74986580)

[2.3 Принцип роботи Unity Engine 26](#_Toc74986581)

[РОЗДІЛ 3 ЕАЛІЗАЦІЯ ПРОЕКТНИХ РІШЕНЬ 28](#_Toc74986582)

[3.1 Створення дизайну рівня 28](#_Toc74986584)

[3.1.1 Моделі ландшафту 29](#_Toc74986585)

[3.1.2 Моделі декорацій 33](#_Toc74986586)

[3.1.3 Моделі інтерфейсу 39](#_Toc74986587)

[3.2 Реалізація механік 41](#_Toc74986588)

[3.2.1 Ефект сцени в зображенні 42](#_Toc74986589)

[3.2.2 Стрільба 42](#_Toc74986590)

[3.2.2 Генерація випадкових маршрутів для качок 42](#_Toc74986591)

[РОЗДІЛ 4 ТЕСТУВАННЯ 43](#_Toc74986592)

[ВИСНОВКИ 46](#_Toc74986594)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 47](#_Toc74986595)

[ДОДАТОК А КОД ПРОГРАМИ 48](#_Toc74986596)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

**AR** (англ. Augmented Reality) – доповнена реальність.

**Ігролад** – процес гри

**Користувач** – особа, що використовує ПЗ з розряду ігор.

**3D** – тривімірний.

**2D** –двовимірний

**[ВСТУП](https://mail.yandex.ua/neo2/" \l "_Toc325035669" \t "_blank)**

Мобільні ігри стали неймовірно популярними, поглинаючи величезну частку ринку ігрової індустрії і до 2021 року мобільні ігри становитимуть сорок п'ять відсотків дохід ігрового ринку. На даний момент дуже перспективним напрямом в мобільних іграх ї ігри з доповненою реальністю.

Доповнена реальність (Augmented Reality, AR) – один із найбільш перспективних напрямів досліджень на сьогоднішній день. Ця технологія вже зарекомендувала себе у багатьох сферах людської діяльності – наприклад, в освіті, медицині, будівництві та ігровій сфері. Головним достоїнством доповненої реальності є наочність, яка сприяє кращому засвоєнню інформації, асоційованої з об'єктом. Завдяки цьому, ця технологія отримала широке поширення в освітній та, зокрема, в музейній сфері. Протягом останніх двадцяти років музеї і комерційні компанії по всьому світу успішно застосовують інформаційні технології для залучення відвідувачів і допомоги їм у сприйнятті експонатів. Використання мобільних додатків з елементами доповненої реальності є одним з найбільш яскравих прикладів таких інновацій. Їх значне поширення в світі частково обумовлено швидким зростанням кількості користувачів смартфонів. які мають можливість отримувати дані про експонатах і товари без надання додаткового обладнання.

Актуальність проекту полягає в тому. що в даний час все більше і більше розвивається світ мобільних пристроїв і відповідно мобільних додатків. Зараз у кожної людини є мобільний телефон і потреба в мобільних додатках с кожним днем тільки збільшується.

Найпопулярнішими жанрами мобільних ігор є ті, у яких бракує складності: головоломки, тривіальні ігри на основі питань та головоломки на відповідність. Ця популярність стосується того, скільки унікальних користувачів грають в місяць, витрачений час гри в місяць та „липкість”, яка визначається як довго гравці будуть продовжувати грати після того, як вони досягли певного рівня майстерності і зазвичай гратимуть за соціальний аспект (змагання або командна гра), а не для того щоб весело провести час [1].

Враховуючи, що майже два мільярди мобільних пристроїв можуть запускати ігри під управлінням Unity, ймовірно, що досить велика кількість нових компаній будуть використовуватимуть Unity для розробки мобільних ігор.

## **РОЗДІЛ 1**

## **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ**

## **1.1 Аналіз предметної області**

Мобільна гра – це відеогра, яку зазвичай грають на мобільному телефоні. Мобільні ігри варіюються від базових (як Snake на старих телефонах Nokia) до складних (3D-ігри та ігри з доповненою реальністю) [1]. Оскільки більшість мобільних пристроїв мають обмежені системні ресурси, функції мобільних ігор не настільки багаті, як ігри, призначені для ПК або ігрових консолей.

Ігри з доповненою реальністю - це остання тенденція мобільних ігор. Ці програми поєднують реальне середовище з вдосконаленою комп’ютерною графікою, щоб забезпечити ефект доповненої реальності. Прикладом може бути Sky Siege, коли гравець стріляє у віртуальні вертольоти, які, здається, літають по кімнаті. Насправді живе зображення кімнати гравця знімається камерою пристрою і подається на екран дисплея, що призводить до доповненої реальності гравця.

Для передових мобільних ігор зазвичай потрібний швидкий центральний процесор (CPU), виділений графічний модуль (GPU), велика кількість оперативної пам’яті (RAM) та екран з високою роздільною здатністю. Більшість розробників використовують безкоштовний ігровий двигун для розробки між платформами, відомий як Unity, для написання ігор з 2D або 3D графікою [2].

## **1.2 Класифікація мобільних ігор**

Сьогодні у нас є широкий вибір мобільних ігор, а також широкий вибір класів ігор. Немає єдиної системи класифікації мобільних ігор. Коріння цієї «жанрової проблеми» полягає у фрагментації та різноманітності мобільного ігрового простору загалом. З такою великою кількістю різних ігрових архетипів, гібридів та постійних інновацій, нелегко класифікувати ігри змістовно та послідовно. Гнучкість. Це єдиний критерій, що вимагається від нової систематики. Оскільки є десятки ігрових архетипів і нових, які з’являються досить регулярно, занадто статичний підхід швидко застаріє. Щоб все було динамічним, додавати нові моделі до моделі має бути порівняно просто. І ця трирівнева ієрархія складається з: категорії, жанру і піджанру [3].

На рисунку 1.1. зображена трирівнева ієрархія.



Рисунок 1.1 – Трирівнева ієрархія

## **1.2.1 За категорією**

**.** Найбільш загальна класифікація розподіляє ігри на категорії.

1. Casual (казуальні) – тип ігор, які орієнтовані на широку аудиторію масового ринку. У казуальних іграх може бути представлений будь-який тип ігрового процесу та жанру. Як правило, вони мають простіші правила, коротші сесії та менш залежні від навичок користувача.
2. Mid-core (середній рівень) – ігри середнього рівня є більш складними, ніж казуальні, вимагаючи від гравців часу на гру, а не на умовно-епізодичну гру. Ігри середнього рівня вимагають майстерності та стратегії для прогресу, і, як результат, вимагають від гравців більших вкладень, ніж у звичайну казуальну гру. Середні ігри часто мають досвід для кількох гравців, побічні квести та управління ресурсами, на відміну від казуальних ігор, де ваша мета - розгадати головоломку або виконати повторювані дії..
3. Sport (спорт) – тип ігор, які намагаються наслідувати гру "традиційного" або "реального" виду спорту, такого як футбол, баскетбол, американський футбол, теніс тощо. Спортивні ігри бувають двох основних різновидів: симулятори та аркадні спортивні ігри. Обидва ці піджанри орієнтовані на імітацію ігрового досвіду спорту як к „реальному світі”. Однак різниця полягає у виконанні [4].
4. Casino – тип ігор, які намагаються наслідувати "реальне" казино. У грі гравці грають в азартні ігри на гроші або фішки в казино на різні можливі випадкові результати або їх комбінації.

## **1.2.2 За жанром**

**.** Найбільш загальна класифікація розподіляє ігри за такими жанрами.

1. Puzzle (головоломки) – жанр ігор спрямованих на логічні та концептуальні завдання. Хоча багато екшн-ігор та пригодницьких ігор включають елементи головоломки у дизайн рівня, справжня гра-головоломка зосереджена на вирішенні головоломки як основний вид геймплейної діяльності [5].

Головоломки можуть включати:

* Відповідність об’єктів на основі категорій (наприклад, візерунки, кольори, фігури, символи тощо)
* Візуалізація та маніпулювання об’єктами в просторі та часі
* Запам'ятовування складних візерунків
* Вирішення зростаючої складності головоломок, що опираються на попередні головоломки
* Розв’язування головоломок в умовах обмеженого часу чи обмеженого життя.
* Застосування раніше відомих знань про світ
* Мислення нестандартно

1. AR (доповнена реальність) – це інтеграція ігрового візуального та аудіовмісту із середовищем користувача в режимі реального часу. Гра з доповненою реальністю часто накладає попередньо створене середовище поверх фактичного середовища користувача. Сама гра може бути настільки простою, як гра у віртуальні шашки, що граються на поверхні столу. Більш просунуті ігри AR можуть насправді створити середовище з середовища користувача. Така гра може включати, наприклад, ігрових персонажів, що піднімаються від журнальних столиків до диванів на віртуальних мостах [6].
2. Arcade (аркада) – жанр ігор котрі вимагають незначного вирішення головоломок або тактичного мислення і покладаються виключно на звичайний геймплей. Простий і зрозумілий ігровий процес - не єдина особливість, в аркадних іграх присутній елемент ігрового процесу на основі циклу (варіації подібних шаблонів повторюються багато разів, і гравець прогресує шляхом повторення та освоєння їх) та ризик / винагорода на основі обмеження за часом або комбінації / оцінка орієнтованого ігрового процесу.
3. Hyper Casual (гіпер-казуальні) – жанр ігор які прості у відтворенні та, як правило, безкоштовні. Вони також мають дуже мінімалістичний інтерфейс користувача. Вони також часто використовують 2D-дизайн з простою кольоровою гамою, що додає простоти. Зазвичай, використовуючи нескінченну петельну механіку, гіпер-казуальні ігри можна грати нескінченно довго. Через відсутність надійної внутрішньоігрової економіки та безкоштовного завантаження більшості гіпер-казуальних ігор, дохід здебільшого приносить реклама [7].
4. RPG (рольова відеогра) – жанр ігор в якому гравець керує головним героєм (або командою), що рухається по вигаданому світі. Часто гравець створює власного персонажа, вказуючи його особливості та зовнішній вигляд. Під час гри гравець можете допомагати NPC і вбивати супротивників, завдяки чому ви перейдете на наступні рівні та отримаєте нові навички. Комп’ютерна рольова гра може розігруватися за різними умовами, наприклад, у фентезі, науковій фантастиці чи стімпанку.
5. Strategy (стратегія) – жанр ігор, в яких для перемоги потрібно, перш за все, навички мислення та планування. Стратегічні ігри підкреслюють тактичні та інколи логістичні завдання. Деякі представники виду також пропонують проблеми, пов'язані з економікою та відкриттям нових територій. Найчастіше стратегічні ігри поділяються на чотири піджанри залежно від того, чи відбувається гра в режимі реального часу чи в покроковій системі, а також від того, чи орієнтована гра на стратегічні чи тактичні завдання [8].
6. Shooter (шутер) – жанр ігор котрий фокусується на впливі гравця на навколишнє середовище за допомогою вогнепальної зброї, а часто і рукопашного бою. Ігровий процес полягає у контролі характеру гравця та усуненні віртуальних супротивників на чолі зі штучним інтелектом або іншими гравцями.
7. Card Games (карткові ігри) – жанр ігор де поєднують привабливість тактичної гри з адреналіном у випадковому виграші здобичі та колоди. В даний час у ніші домінують два варіанти карткової гри: висококонкурентні багатокористувацькі ігри TCG / CCG, це похідні від Hearthstone, і більш звичайні ігри, часто перероблені з фізичних ігор, які вже існують.

## **1.3 Аналіз аналогів програмного продукту**

З кожним днем ігрова індустрія розвивається дуже швидко і напевно кожна людина грала, або грає у відеоігри. Серед великої кількості напрямів в відео ігровому світі, останні 5 років набули популярності невеликі проекти так звані інді-ігри. Це відеоігри, котрі, як правило, створюється окремими особами або меншими командами розробників без фінансової та технічної підтримки великого видавця ігор, на відміну від більшості ігор "AAA” класу. Через свою незалежність та свободу розвитку інді-ігри часто зосереджуються на інноваціях, експериментальному геймплеї та ризикуванні, яке зазвичай не передбачено в великих відеоіграх, і можуть досліджувати середовище, щоб отримати унікальний досвід у мистецьких іграх. Тому для того, щоб обрати для свого додатку найкраще з кращих було проведено аналіз найкращих інді-ігор, а саме: Brickscape, Kings of Pool, Knightfall: AR

Brickscape ­– це 3D-головоломка. Переміщайте прямокутні блоки і міняйте їх місцями. Очистіть шлях до виходу з рівня для особливо зазначеного блоку. Мета цієї гри полягає в тому, щоб допомогти одному з блоків втекти з рівня. На кожному рівні ви знайдете кубічну область простору, заповнену блоками. Ви може пересувати будь-який з блоків. Але пам'ятайте, що блоки можуть рухатися тільки в поздовжньому напрямку. Шляхом маніпуляцій з блоками, ви повинні привести головний блок до виходу, розміщеному на одній зі стінок ігрової зони.. Це відносно проста гра. Гра може відтворюватися без AR з звичайним фоном, так і в AR. Версія AR дозволяє грати в гру через камеру з оточенням. Приклад візуального оформлення на рисунку 1.2.

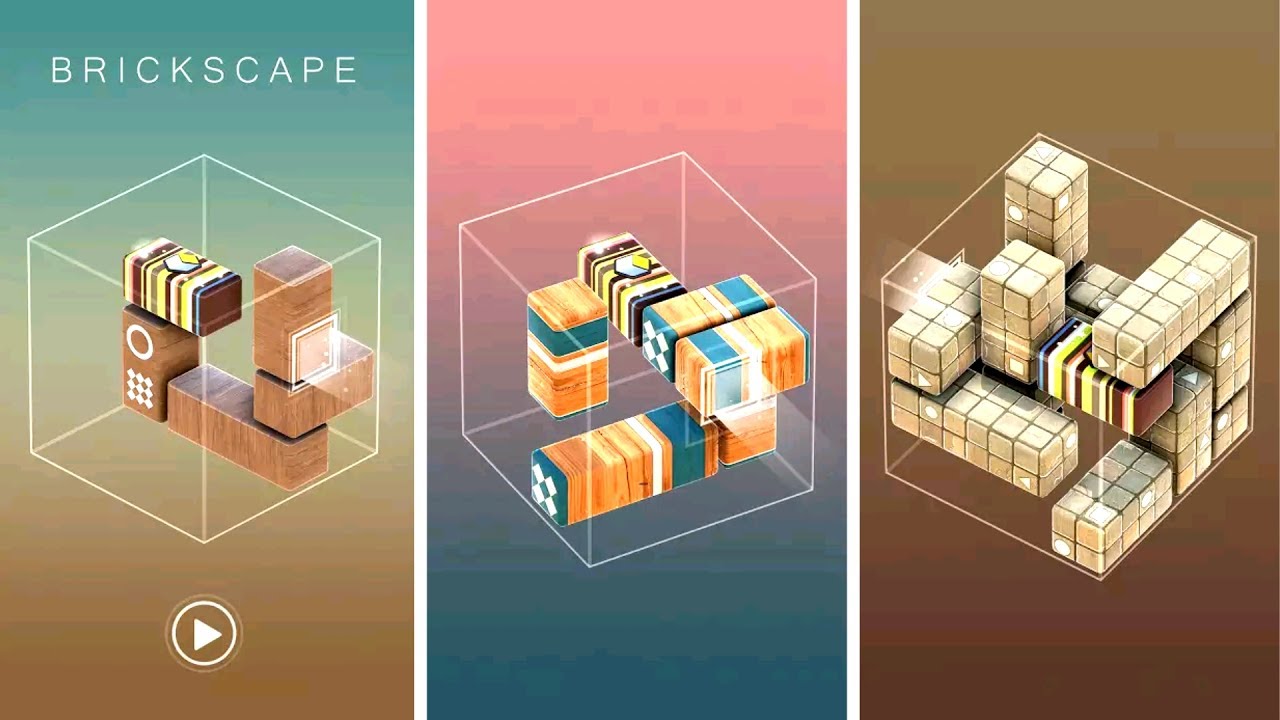


Рисунок 1.2 – Геймплей гри Brickscape

Kings of Pool - гра в більярд з унікальним функціоналом. За допомогою AR ви можете створити стіл у натуральну величину на будь-якій поверхні. Режим має реалістичну фізику кульок та приголомшливу 3D-графіку. Режим AR можна повністю налаштувати, використовуючи ваш спеціальний фетр та наклейки. Додаток накладає більярдний стіл на будь-яку рівну поверхню. Потім ви можете пограти в більярд. У грі представлена багатокористувацька гра в Інтернеті з основним видом гри на 8 куль. Крім того, є глобальний чат, профіль гравця з усіма статистичними даними. Приклад візуального оформлення на рисунку 1.3.



Рисунок 1.3 – Геймплей гри Kings of Pool

Knightfall: AR - ставить гравців на місце лицарів тамплієрів, оскільки вони захищають Акр від ворожих мамелюцьких воїнів. Ви розміщуєте поле бою на будь-якій рівній поверхні поблизу вас, і ваша точка зору тоді діє як прицільна сітка, що дозволяє вам стріляти стрілами та катапультами по ворожих підрозділах, коли вони просуваються до ваших стін. Вбиваючи ворогів, ви отримуєте золото, яке можна витратити на оборону та воїнів, щоб відштовхнути загарбників назад. Подолання рівнів кампанії також розблоковує анімацію для ігрового фоторежиму, що дозволяє виводити персонажів з гри для власного розваги. Приклад візуального оформлення на рисунку 1.4.



Рисунок 1.4 – Геймплей гри Knightfall: AR

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Назва | Дизайн  інтерфейсу | Складна гра | Різноманітність  рівнів | Комерційність |
| Brickscape | Простий | Ні | Велика | Безкоштовна |
| Kings of Pool | Захопливий та цікавий | Ні | Мала | Безкоштовна |
| Knightfall: AR | Простий та привабливий | Ні | Мала | Безкоштовна |

Провівши аналіз по таблиці можна дійти до висновку що гра має бути з простим інтерфейсом, малою різноманітністю рівнів та бажано безкоштовною.

## **1.4 Постановка задачі**

**Об’єкт дослідження:** використання системи доповненої реальності для отримання користувачем нового ігрового досвіду.

**Предмет досліджень:** засоби розробки ігор з доповненою реальністю з використанням ігрового двигуну Unity.

**Мета дипломної** **роботи:** створення гри жанру AR/Arcade з використанням 3D графіки у доповненій реальності. У результаті буде створений самостійний програмний продукт, котрий буде безкоштовний для кінцевого користувача, який слугуватиме для розваги та демонстрації нових технологій.

Основною платформою програмного продукту, є мобільні телефони з операційною системою Android не нижче Android 7.0. Результуючий програмний продукт має бути максимально простим для встановлення на мобільний телефон. Тож враховуючи специфіку мобільних телефонів, будемо мати такі мінімальні вимоги:

1. наявність відео-картки з підтримкою технології Vulkan
2. наявність на телефоні 400 мегабайт вільної оперативної пам’яті
3. наявність процесору з сумарною тактовою частотою не менше ніж 2 гігагерци;

Розробимо план процесу розробки програмного забезпечення.

1. Проектування гри.
   1. Вибір інструментів для доповненої реальності.
   2. Створення геймдизайну.
   3. Створення дизайну рівня.
      1. Дизайн локації.
      2. Дизайн противника.
      3. Дизайн зброї.
2. Програмування.
3. Інтеграція складових частин програмного забезпечення.
4. Тестування та виправлення дефектів.

## **РОЗДІЛ 2**

## **ПРОЕКТНІ РІШЕННЯ**

## **2.1 Вибір засобів розробки**

Ігровий двигун - основна частина коду комп'ютерної гри, доступна з інтегрованим середовищем програмування, призначеним для людей та команд, що створюють комп'ютерні ігри. Він надає функції від анімації до штучного інтелекту. Ігрові двигуни відповідають за візуалізацію графіки, виявлення зіткнень, управління пам’яттю та багато інших опцій.

Ігрові двигуни надають розробникам інструменти для створення численних ігрових додатків. Дизайнери часто використовують ці двигуни для створення інших ігор, що робить це цінною інвестицією.

Ігровий движок містить п’ять компонентів: основна ігрова програма, яка містить ігрову логіку; механізм візуалізації, який можна використовувати для створення 3D-анімованої графіки; аудіосистема, яка складається з алгоритмів, що відносяться до звуків; фізичний двигун для реалізації “фізичних” законів у системі; та Штучний інтелект, модуль, призначений для використання інженерами програмного забезпечення зі спеціальним призначенням.

Для успішного проекту, який зможе довго утримувати гравців, а також приносити прибуток кілька років потрібно створити добрий продукт, а також здійснювати довготривалу підтримку продукту. Для всього цього потрібне використання найбільш придатних засобів розробки. На ринку існує велика кількість движків для реалізації відеоігор. Для розробника важливим завданням є вибір відповідного ігрового движка, кожен з яких є оптимальним вибором для конкретного жанру. Однак існують рішення, які підходять для мультижанрових проектів зі зручними інструментами як розробки, так і підтримки. Визначитися з ігровим движком означає побудувати правильну стратегію розробки та підтримки програми.

## **2.1.1 Вибір двигуна**

**.** На ігровому ринку існують дуже багато двигунів, але з них є три основні, а саме: Unity Engine, Unreal Engine, Godot.

Unity Engine – це мультиплатформенний ігровий движок, який дозволяє легко створювати інтерактивний 3D-вміст. Цей ігровий двигун сьогодні є вибором багатьох великих організацій завдяки його чудовій функціональності, високоякісному контенту та можливості використовувати його для будь-якого типу гри. Він підтримує як 2D, так і 3D вміст. Завдяки своєму універсальному редактору Unity сумісний з Windows, Mac, Linux, IOS, Android, Switch, Xbox, PS4, Tizen та іншими платформами. Зручний інтерфейс полегшує розробку та зменшує потребу в навчанні. Магазин активів Unity зберігає величезну колекцію інструментів та вмісту, що створюється щодня. Використовує мову програмування C#.

Unreal Engine – один з найпопулярніших і широко використовуваних ігрових двигунів є Unreal Engine, який належить Epic Games. По суті, це багатоплатформенний механізм розробки ігор, розроблений для підприємств будь-якого розміру, який допомагає використовувати технологію реального часу для перетворення ідей у залучення візуального вмісту. Unreal Engine пропонує різноманітні навчальні посібники, які допоможуть новачкам почуватись комфортно з двигуном. Як і Unity, Unreal має власний ринок інструментів. Використовує мову програмування C++;

Godot – це двигун з відкритим кодом для крос-платформної розробки ігор. Godot прекрасно слугує для розробки 2D-ігор, і він також добре працює з 3D. Випуск Godot 3.0 приніс значне оновлення 3D-функцій, прискоривши його з іншими сучасними ігровими движками, зберігаючи безкоштовну ціну. Godot також пропонує спеціальний 2D-движок, який працює в піксельних координатах і робить розробку 2D легкою. Завдяки різноманітним доступним мовам, включаючи C ++, C # та GDScript (варіант пітона), Godot легко програмувати та легко вивчати.

Першим критерієм вибору ігрового двигуна, є його ціна.

1. Unity Engine;
2. Unreal Engine;
3. Godot;

Другим критерієм вибору ігрового двигуна, є підтримка технологій AR. А саме підтримка технологій ARCore для операційної системи Android, підтримка ARKit для операційної системи IOS та часткова підтримка фреймворку Vuforia Engine. Ці технології підтримують такі двигуни:

1. Unity Engine;
2. Unreal Engine;
3. Godot;

Третім критерієм вибору ігрового двигуна, є платформи котрі він підтримує. А саме операційні системи IOS та Android. Ці платформи підтримують такі двигуни:

1. Unity Engine – безкоштовна модель поширення для проектів з оборотом не більше 100 тис. доларів за останні 12 місяців;
2. Unreal Engine – безкоштовна модель поширення для проектів з оборотом не більше 1 міліонна доларів;
3. Godot – повністю безкоштовний;

Четвертим критерієм вибору ігрового двигуна, є мова програмування яку він підтримує. Було визначено такі мови програмування для створення гри:

1. C#;
2. C++;
3. GDScript;

Під поточні потреби було вирішено обрати ігровий двигун Unity Engine з таких причин:

1. підтримка фреймворку для AR Vuforia Engine. Дане рішення – це набір для розробки програм доповненої реальності для мобільних пристроїв, що дозволяє створювати додатки доповненої реальності. Він використовує технологію комп'ютерного зору для розпізнавання та відстеження площинних зображень та 3D-об'єктів у режимі реального часу. Ця можливість реєстрації зображень дозволяє розміщувати та орієнтувати віртуальні об’єкти, такі як 3D-моделі та інші засоби масової інформації, стосовно об’єктів реального світу, коли вони переглядаються через камеру мобільного пристрою;
2. розробка за допомогою мови програмування C#. Дана мова програмування добра тим, що вона має компонентно-орієнтований підхід до програмування, сприяє меншій машинно-архітектурної залежності результуючого програмного коду, гнучкості, переносимості і легкості повторного використання фрагментів коду;
3. Вигідні умови ліцензування. Безкоштовна модель поширення для проектів з оборотом не більше 100 тис. доларів за останні 12 місяців;
4. двигун має зручний інтерфейс з можливістю розставляти об'єкти і в реальному часі тестувати результат;
5. також Unity Engine володіє великим співтовариством розробників, величезною бібліотекою готови рішень і плагінів, що прискорює процес розробки. За допомогою двигуна можна розробляти ігри на всі актуальні ігрові платформа і перенести їх без особливої складності.

## **2.1.2 Вибір програмного забезпечення для створення 3D-моделей**

**.** Для розробки левел дизайну ігри, потрібний засіб комп’ютерного моделювання. Оскільки гра являється тривимірною і користувач зможе розглядати прискіпливо 3D моделі, цей вибір робився виважено.

На сьогоднішній день на ринку програмного забезпечення для створення 3D-моделей, домінують три програми, а саме: Blender, Maya, ZBrush

Blender – програмне забезпечення з відкритим кодом для 3D-моделювання, яке є чудовою альтернативою своїм платним братам. Як і будь-яке програмне забезпечення з відкритим кодом, Blender має велику спільноту розробників, художників та дизайнерів, які завжди готові допомогти. Blender, безкоштовне програмне забезпечення для 3D-моделювання для Mac, працює також у Windows та Linux.

Maya – це програмне забезпечення для 3D-моделювання, анімації, рендерингу та моделювання. Maya використовується у розробці відеоігор, анімаційних фільмів та візуальних ефектів і пропонує вищі інструменти для персонажів та ефектів. Пропонується 30-денна безкоштовна пробна версію, але не можливо комерційно публікуваить жоден з проектів, який створюється в пробному періоді, Тож ціна ліцензії 245 доларів в місяць.

Zbrush – це програма для цифрового ліплення в 3D, розроблена американською компанією Pixologic і поєднує в собі 3D-моделювання, текстурування та фарбування. Вважається одією з найкращих і найпопулярніших програм для створення 3D-моделей. ZBrush не використовує жодних програм САПР для створення своїх моделей. Натомість вона використовує технологію "3D пікселі", схожу на скульптуру. Кожен піксель, який називається «піксол», містить інформацію про глибину, орієнтацію, матеріал та значення кольору, що робить ZBrush одним із найбільш зручних програм для моделювання та текстурування. Ціна ліцензії 895 доларі.

Для моделювання була обрана програма Blender, за її простоту освоєння та умови ліцензування. Вона цілком безкоштовна. Всі моделі розроблені власноруч.

Для створення текстур було використано безкоштовний растровий редактор Gimp.

Отже, були обрані такі інструменти:

1. Unity Engine з мовою програмування C#;
2. Vuforia SDK;
3. Blender;
4. Gimp;

## **2.2 Принци роботи Vuforia Engine**

На сьогоднішній день системи комп’ютерного зору є дуже складними і комплексними, тому ми розглянемо більш детально як комп’ютерний зір може розрізняти 2D об’єкти в контексті Vuforia Engine.

Цільові зображення - це зображення, які Vuforia Engine може виявляти та відстежувати. Vuforia Engine працює по принципу пошуку природних рис, порівнюючи їх із зображення камери з відомою базою даних цільових ресурсів. Після виявлення цільового зображення, Vuforia Engine буде відстежувати зображення.

Пошук природних рис – це техніка відстеження, яка виявляє та відстежує особливості, які природно зустрічаються на зображенні. Це можуть бути кути, краї, краплі, тощо, без використання спеціально розроблених маркерів. Існує кілька функцій відстеження природних ознак таких як SIFT, SURF, Ferns [9]. Вони відрізняються здебільшого особливостями зображення, які є пов'язані між відеозображенням і модель об'єкта чи середовища, що буде відстежуватися. Основний ланцюжок процесів відстеження представлено на рисунку 2.1.

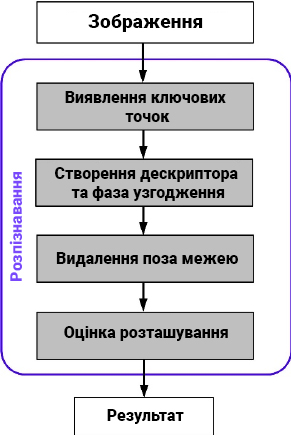


Рисунок 2.1 – Основний ланцюжок процесів відстеження

“Виявлення ключових точок” зазвичай здійснюється кутовим детектором для обчислення точки на зображенні камери.

“Фаза створення та узгодження дескрипторів” залежить від вибору дескриптора. Наприклад, з функцією SIFT, алгоритм оцінює домінуючу ключову точку орієнтації з використанням градієнтів, компенсує для виявлення орієнтації і, нарешті, описує ключові точки з точки зору градієнтів навколо її. Далі файл робиться файл у базі даних із усіма дескрипторами та їх позиціями щодо оригіналу. Потім під час відстеження в реальному часі алгоритм зрівнює дескриптори з відео проти тих, хто знаходиться в базі даних.

"Видалення чужого" складається з каскаду методів для видалення ключових точок, починаючи з найдешевших (прості геометричні тести, такі як лінійний тест), закінчуючи найдорожчими (тести на основі гомографії).

Розташування з гомографія використовується як відправна точка для «Оцінки розташування”. Потім, крок вдосконалення розташування здійснюється на основі метода Гауса-Ньютона для мінімізація повторної проекції помилки ключових точок. Зазвичай 2-4 ітерації досить.

## **2.3 Принцип роботи Unity Engine**

Робочий процес Unity побудований навколо структури компонентів. Компонент може розглядатися як менший шматок більшого механізму. Кожен компонент має свою конкретну роботу, і може загалом (і оптимально) виконує своє завдання або мету без допомоги будь-яких зовнішніх джерел. Крім того, компоненти рідко належать одному механізму і можуть поєднуватися з різними системами щоб виконати своє конкретне завдання, але досягати різних результатів, коли справа стосується загальної картини. Це тому, що компоненти не тільки не дбають про цю загальну картину, але й навіть не знають, що воно існує.

Наприклад контролер PlayStation він має безліч кнопок, але кожна з них цього не уявляє є й інші кнопки, які мають різну поведінку. Натиснуті кнопки завжди роблять свою справу самостійно. Справа в тому, що функція контролера одностороння, і його завдання ніколи не буде змінити через те, до чого він підключений. Це робить його успішним компонентом, оскільки він може зробити свою роботу як самостійний пристрій, але також тому, що він може виконувати свою роботу з кількома пристроями.

Unity дозволяє бачити все, над чим працюєш в реальному часі. Це означає, що можливо переглянути проект, що працює в окремому вікні, внести зміни у свій код або групу об'єктів, і побачити ці зміни.

## **РОЗДІЛ 3**

## **РЕАЛІЗАЦІЯ ПРОЕКТНИХ РІШЕНЬ**

## **3.1 Створення дизайну рівня**

Гра буде мати один рівень, в якому будуть літати качки. Рівень був створений з дотриманням малополігонального стилю, а саме low poly. Основна ідея гри, це полювання на качок, тож було прийнято рішення зробити рівень в декораціях лісу. На рисунку 3.1 представлена модель качки.



Рисунок 3.1 – Модель качки без застосування текстур

## **3.1.1 Моделі ландшафту**

**.** Моделі створюються окремо, а потім з’єднуються в бажаній формі. Використовуються моделі рівнин, рівнин з дорогами, скелі. На рис. 3.2 – 3.10 представлені моделі ландшафту.

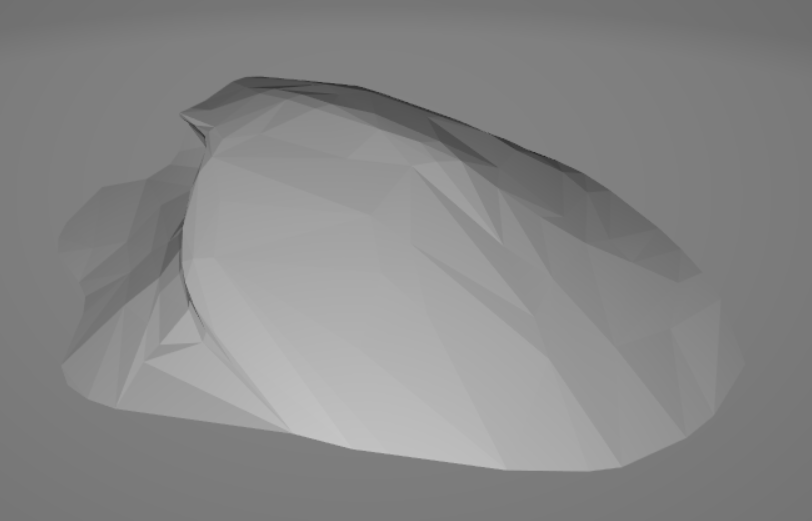
****

Рисунок 3.2 – Модель ландшафту без текстур

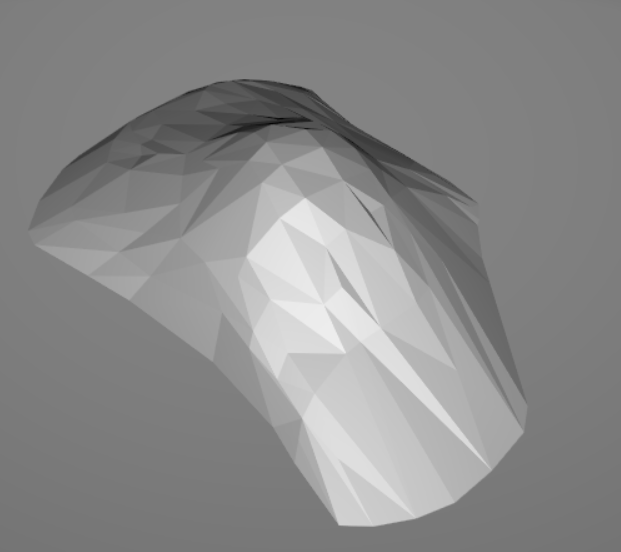
****

Рисунок 3.3 – Модель ландшафту без текстур

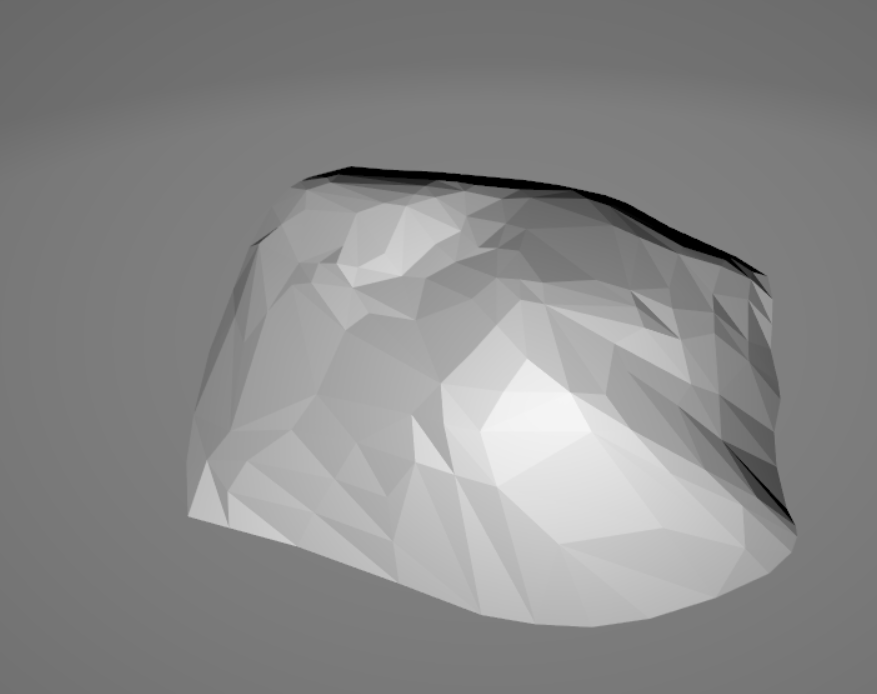


Рисунок 3.4 – Модель ландшафту без текстур

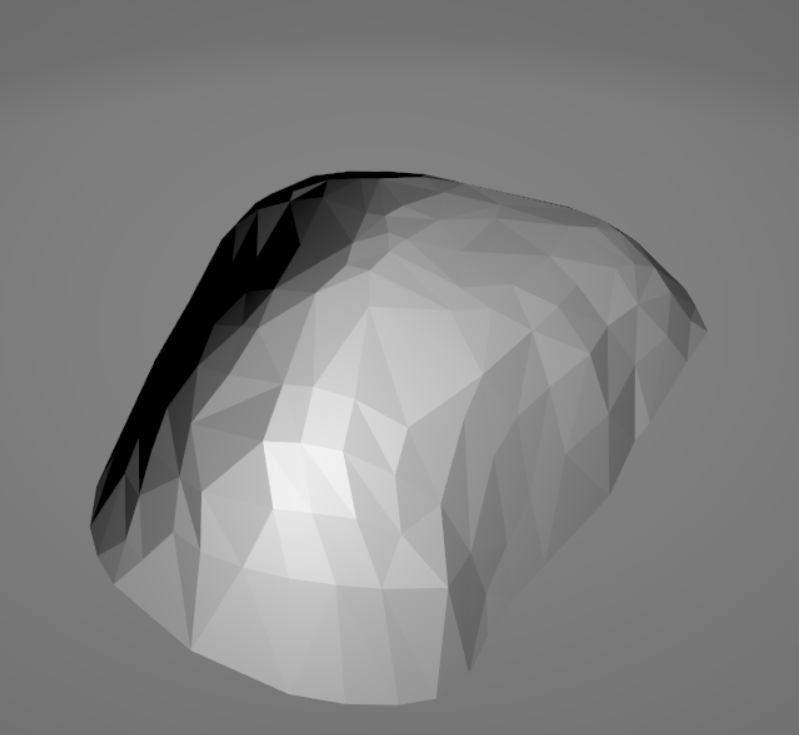


Рисунок 3.5 – Модель ландшафту без текстур

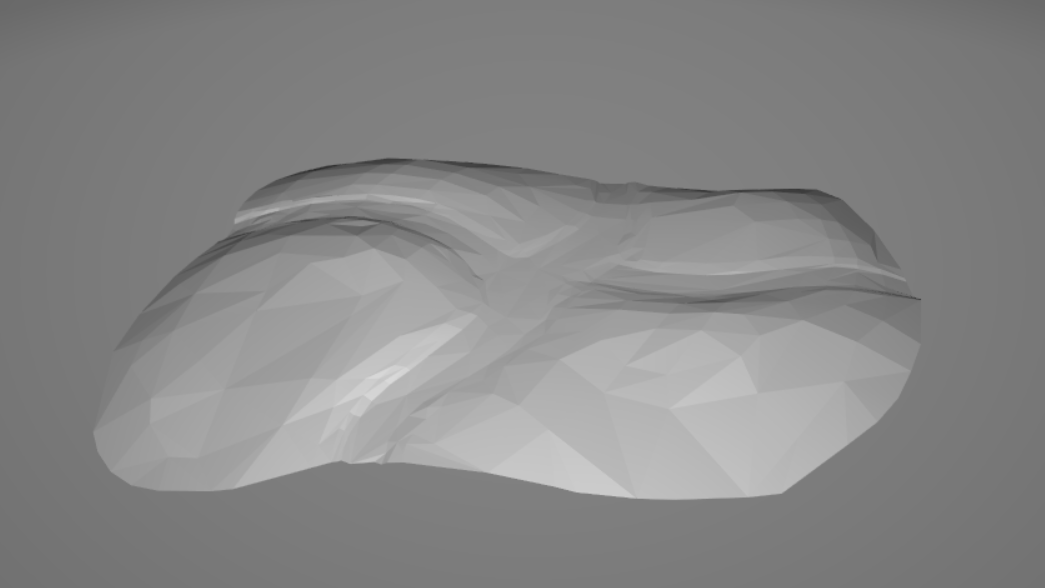


Рисунок 3.6 – Модель ландшафту без текстур

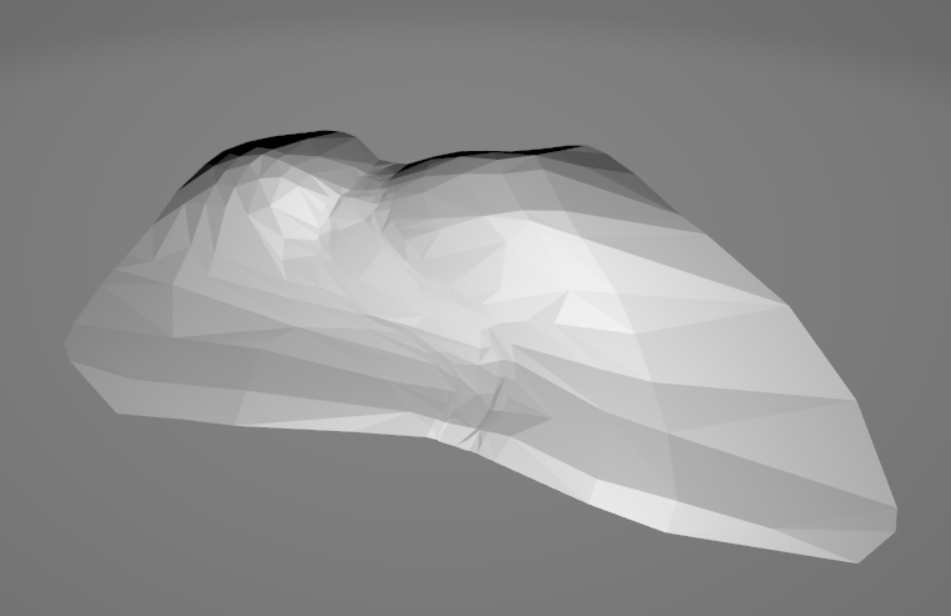


Рисунок 3.7 – Модель ландшафту без текстур

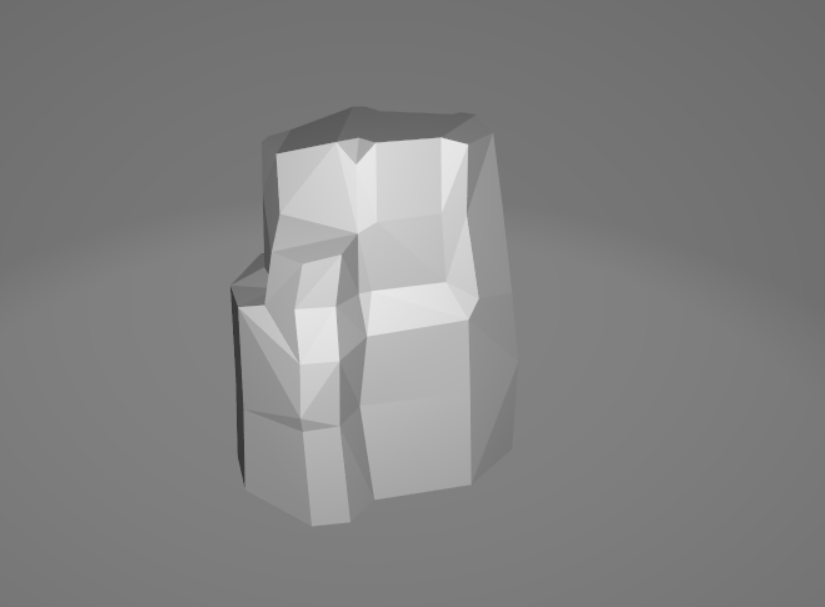
****

Рисунок 3.8 – Модель ландшафту без текстур

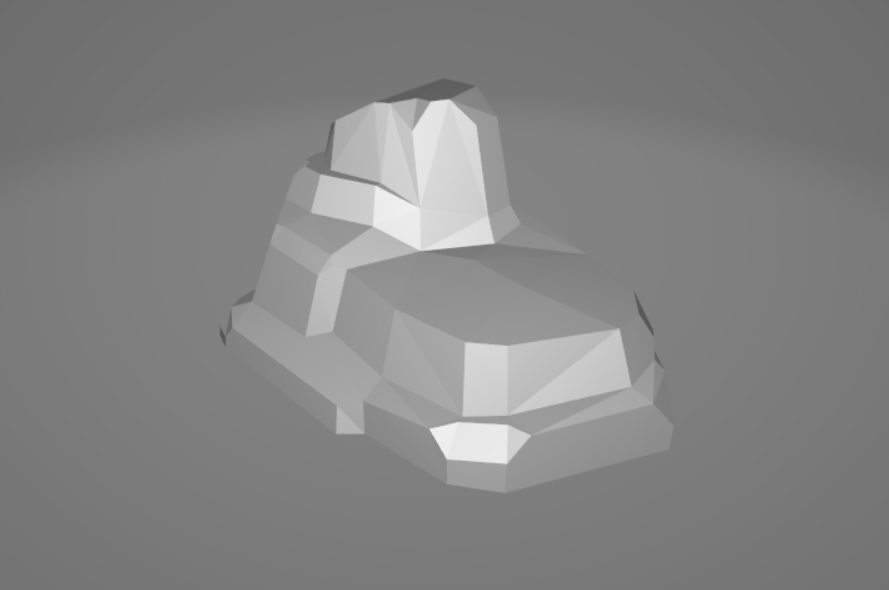
****

Рисунок 3.9 – Модель ландшафту без текстур

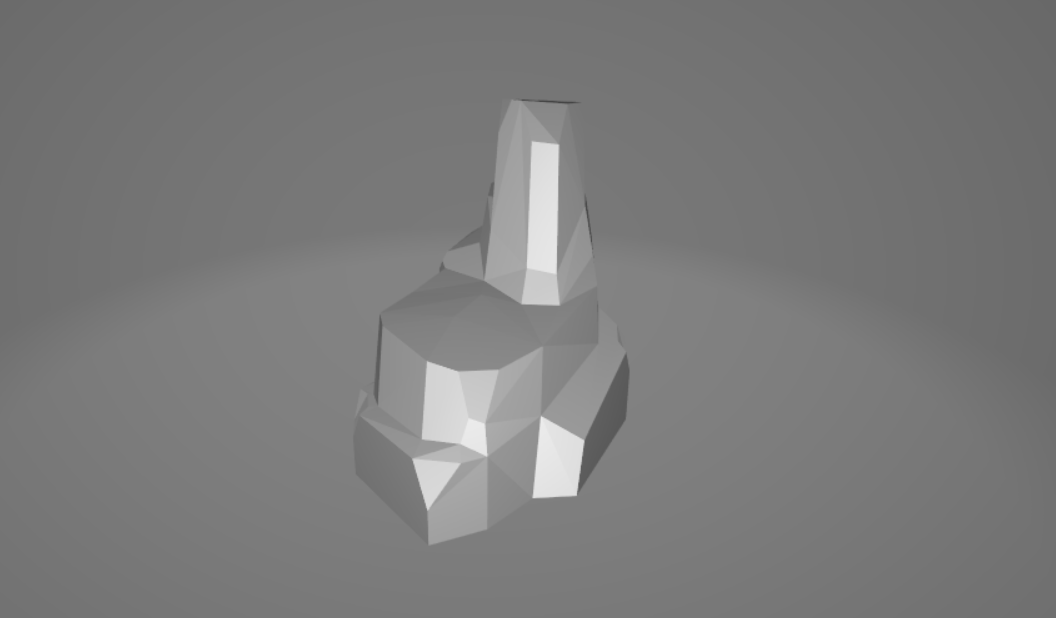


Рисунок 3.10 – Модель ландшафту без текстур

## **3.1.2 Моделі декорацій**

**.** Моделі створенні для прикрашення та надання естетичного вигляду сцені. Використовуються моделі дерев, повалених дерев, містки, дерев’яні паркани, кам’яні паркани, ліхтарі, обрамлення для зображення. На рис. 3.11 – 3.21 представлені моделі декорацій.

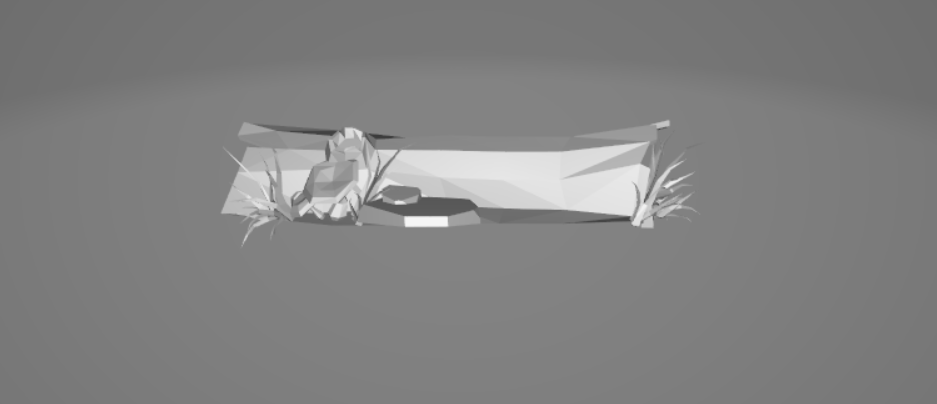


Рисунок 3.11 – Модель колоди дерева без текстур

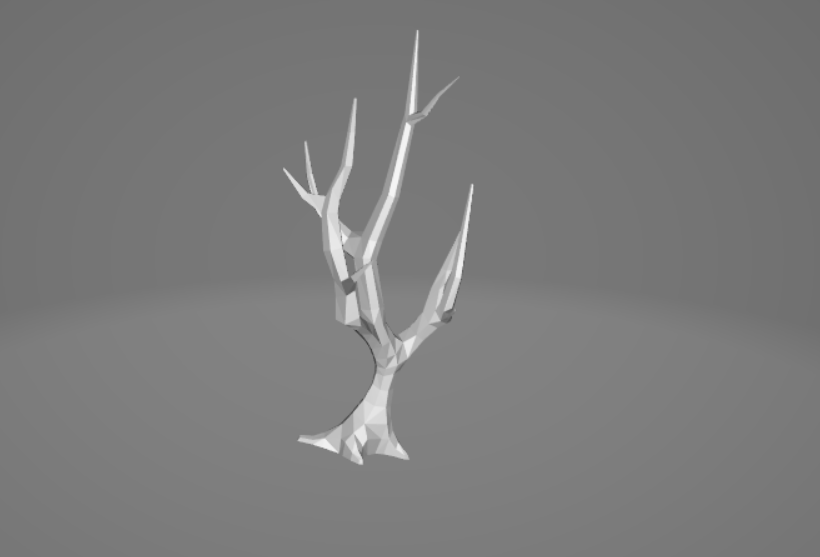


Рисунок 3.12 – Модель мертвого дерева без текстур

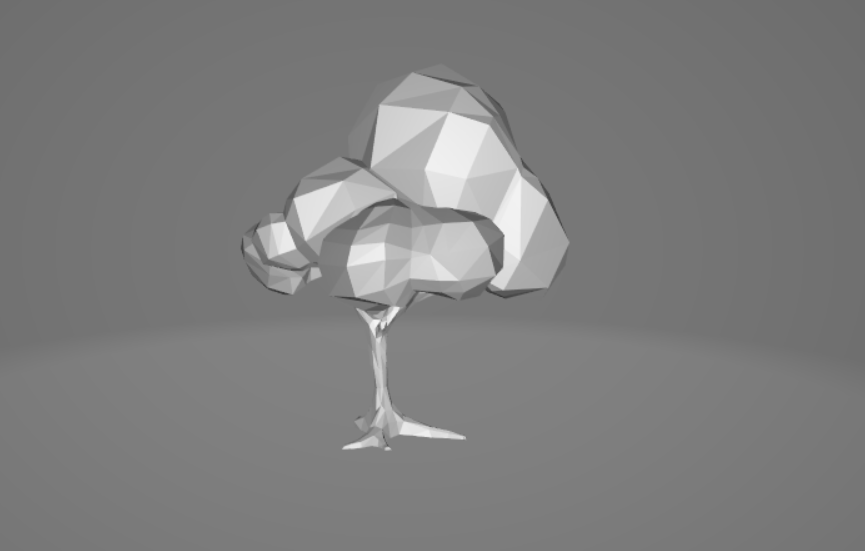


Рисунок 3.13 – Модель дерева без текстур

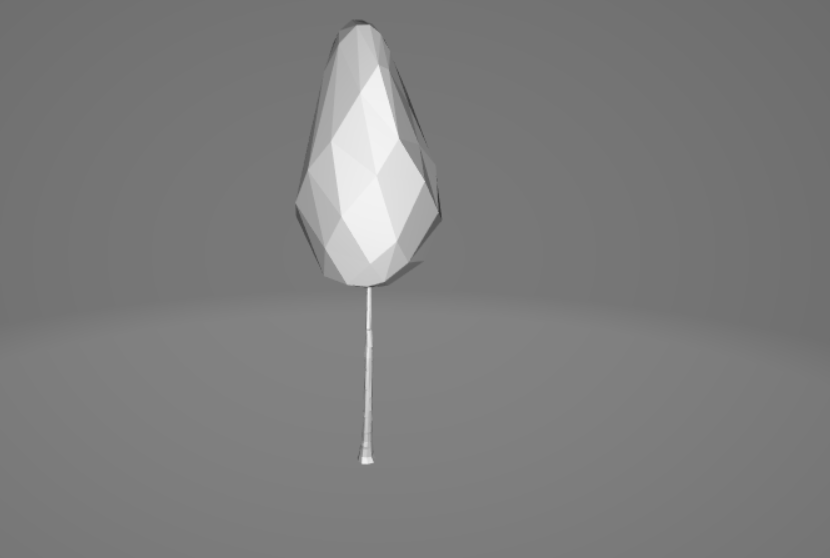


Рисунок 3.14 – Модель ландшафту без текстур

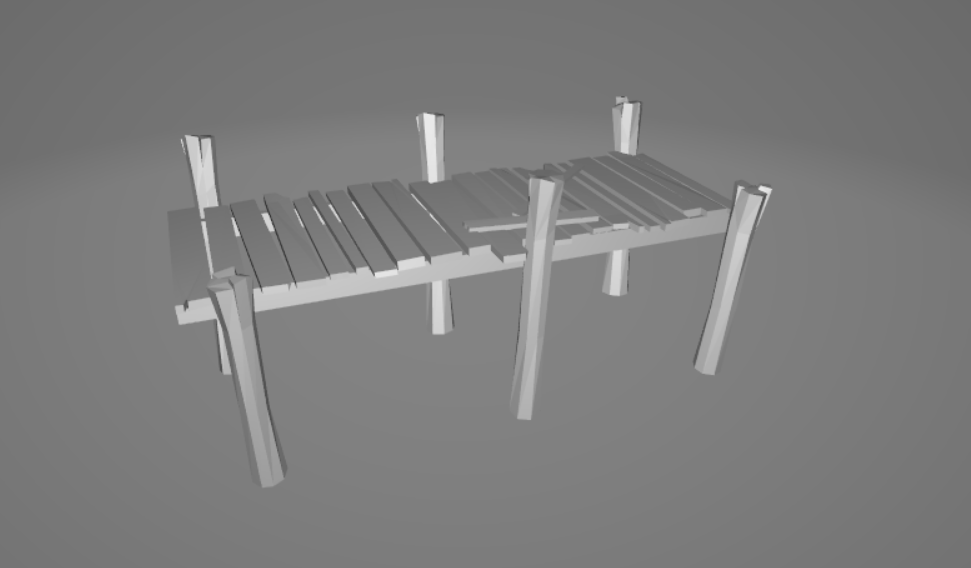


Рисунок 3.15 – Модель містка без текстур



Рисунок 3.16 – Модель містка без текстур



Рисунок 3.17 – Модель паркану с каменів без текстур

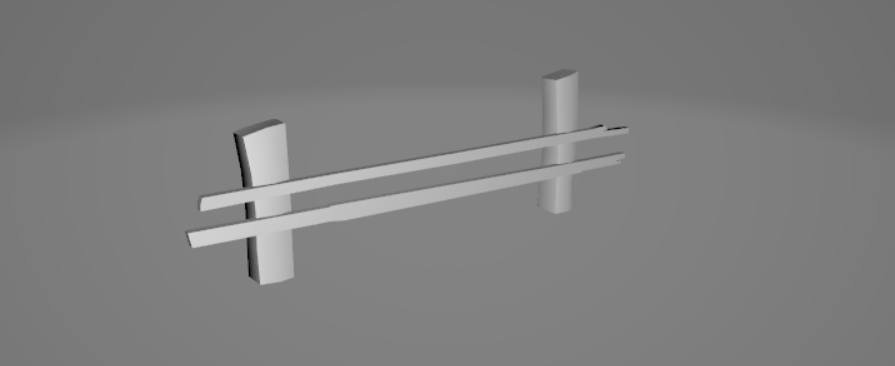


Рисунок 3.18 – Модель паркану з дерева без текстур

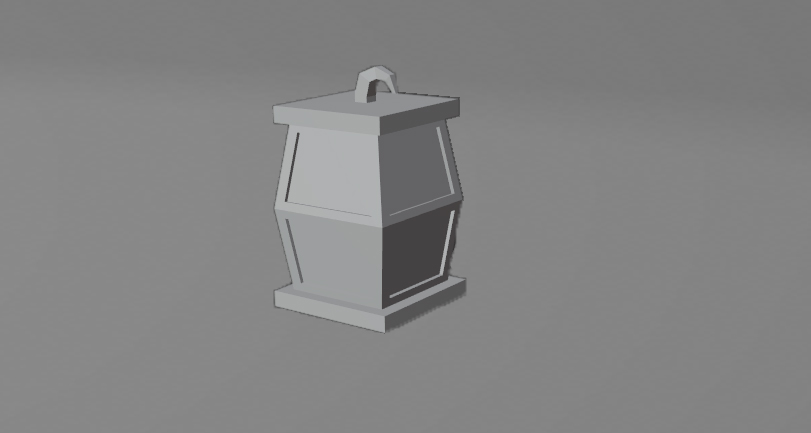


Рисунок 3.19 – Модель ліхтаря без текстур

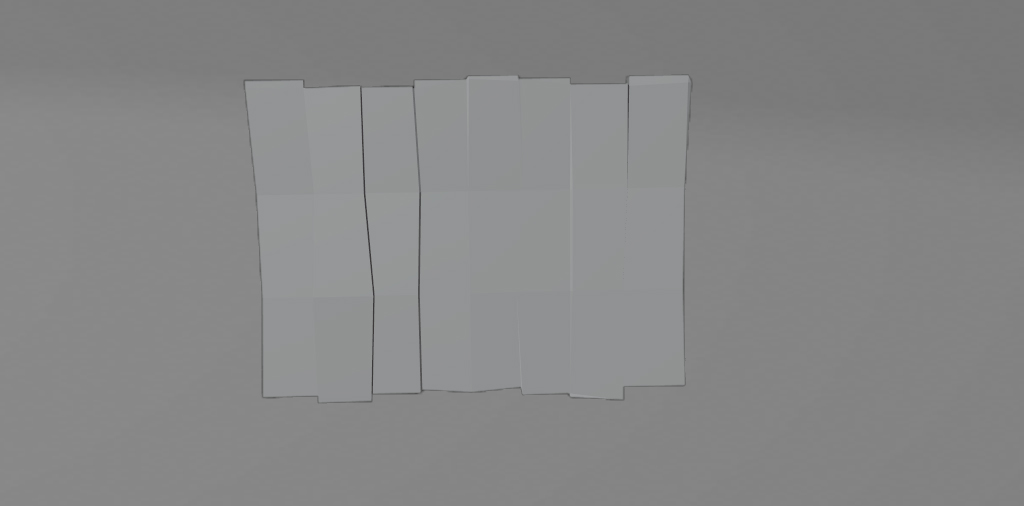


Рисунок 3.20 – Модель дерев’яної стіни без текстур



Рисунок 3.21 – Модель обрамлення для зображення без текстур

## **3.1.3 Моделі інтерфейсу**

**.** Так як ігра в доповненій реальності, то було прийнято рішення дотримуватися дієгєтичного інтерфейсу, а саме інтерфейс має бути органічно вписан в ігролад. Використовуються моделі 3D надписів, гільзи, рушниця та ціль. На рис. 3.22 – 3.28 представлені моделі інтерфейсу.



Рисунок 3.22 – Модель рушниці без текстур

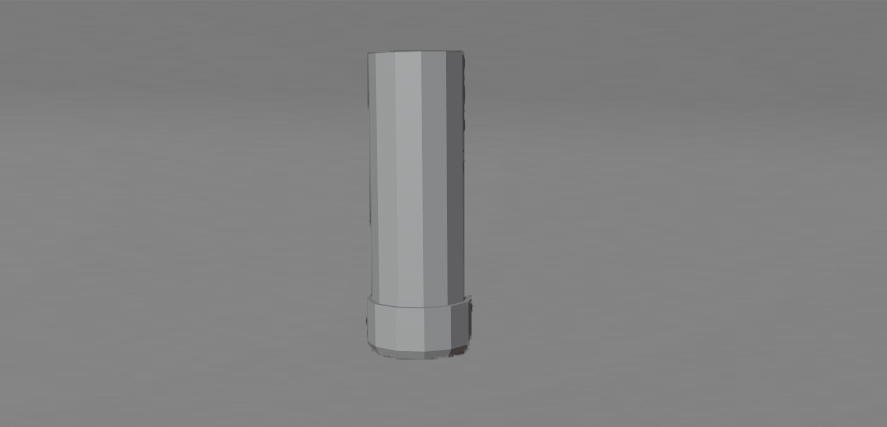


Рисунок 3.23 – Модель гільзи без текстур



Рисунок 3.24 – Модель надпису без текстур



Рисунок 3.25 – Модель надпису без текстур



Рисунок 3.26 – Модель надпису без текстур

****

Рисунок 3.27 – Модель цілі без текстур

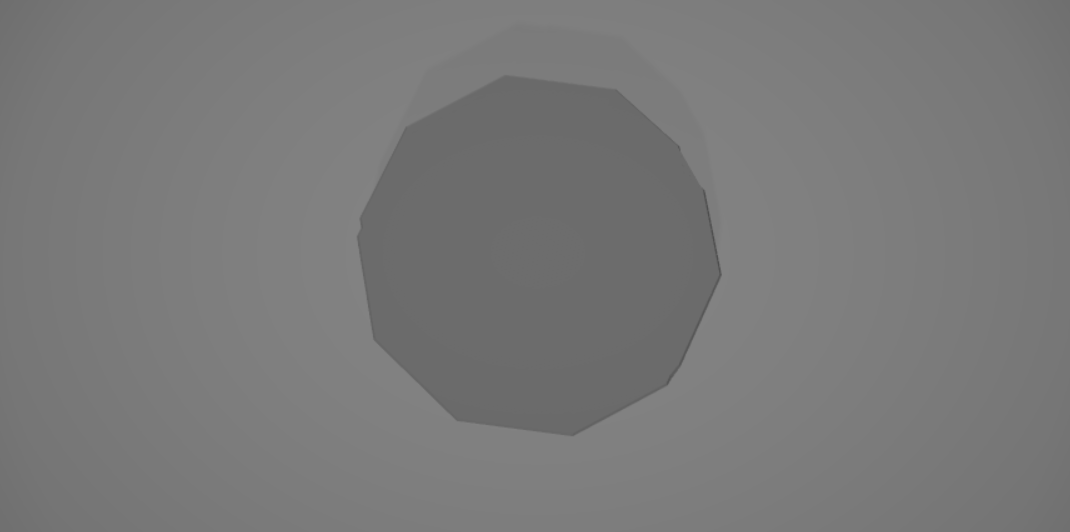


Рисунок 3.28 – Модель цілі без текстур

## **3.2 Реалізація механік**

Після розробки гейм дизайну та левел дизайну, було реалізовані механіки, а саме: ефект сцени в зображенні, стрільба, реєстрація попадань, генерація випадкових маршрутів для качок.

## **3.2.1 Ефект сцени в зображенні**

**.** Для цього ефекта був використаний шейдер написаний на мові програмування HLSL – high level shader language. В цьому шейдері використовувався stencil buffer.

Stencil buffer (буфер трафаретів) – це додатковий буфер даних, на додаток до буфера кольорів та Z-буфера. Буфер розраховує кожен піксель і працює із цілими значеннями, як правило, глибиною один байт на піксель. Z-буфер і буфер трафаретів часто мають однакову область в оперативній пам'яті графічного обладнання. У найпростішому випадку буфер трафарету використовується для обмеження площі візуалізації. Більш розширене використання буфера трафаретів використовує міцний зв’язок між Z-буфером і буфером трафаретів у конвеєрі рендеринга.

## **3.2.2 Стрільба**

**.** Стрільба являється основним способом взаємодії з рівнем. За допомогою стрільби можливо ліквідовувати качок та розпочинати гру. Тож це дуже важливий аспект котрий потрібно проробити детально. Було розроблено список характеристики для рушниці, а саме:

1. кількість набоїв у магазині – характеризує кількість набоїв у користувача.
2. час перезарядки – характеризує час за котрий оновляться набої у користувача.
3. темп стрільби – характеризує час котрий має пройти між пострілами, щоб вистрілити ще раз.

Реєстрація попадання працю за принципом зіткнення коллайдера качки та коллайдера кулі. У випадку зіткнення куля видаляється, а качка програє анімацію падіння. Реєстрацією попадань виконую фізичний двигун Unity Engine, а саме Rigidbody.

## **3.2.2 Генерація випадкових маршрутів для качок**

**.** Для того щоб користувачу швидко не набридло грати в гру, потрібно додати елемент випадковості. Качки літають по випадковим маршрутам, котрі генеруються при початку гри і після відродження качки. Генерування випадкових маршутів виконано за допомогою кривих Безьє.

## **РОЗДІЛ 4**

## **ТЕСТУВАННЯ**

Тестування являється важливим аспектом у створенні успішної відеогри. Щоб тестування було ефективним, потрібно проводити його на протязі всієї розробки відеогри. Таких підхід зменшить затрати на тестування та полегшить роботу Адже саме тестування визначає якість програмного забезпечення.

Для тестування відеогри застосовувався метод play-тестування. Як випливає з назви, в цьому методі тестування тестувальник повинен грати в гру як звичайнин користувач. Тестувальник грає від імені гравця і перевіряє, на некоректне поводження гри. Крім того, кожного разу, коли тестувальник тестує гру як гравець, це допомагає виявляти загальні проблеми, такі як поганий геймдизайн, або поганий дизайн рівнів. У команди також є можливість перевірити всі нефункціональні елементи.

Тож були протестовані такі сценарії:

* Позиціювання моделі на зображенні у статиці;
* Позиціювання моделі на зображенні у русі;
* Програвання звуку при пострілі;
* Реакція кулі на зіткнення;
* Реакція качки на зіткнення з кулею;
* Відновлення качки;

Таблиця 4.1 – Позиціювання моделі на зображенні у статиці

|  |  |  |
| --- | --- | --- |
| Опис | Позиціювання моделі на зображенні у статиці | |
| Передумови | Гра запущена | |
| Дії | | Очікуваний результат |
| 1.Навести камеру на зображення | | На зображенні виникне 3D модель |

Таблиця 4.2 – Позиціювання моделі на зображенні у русі

|  |  |
| --- | --- |
| Опис | Позиціювання моделі на зображенні у русі |
| Передумови | Гра запущена |

Продовження таблиці 4.2

|  |  |
| --- | --- |
| Дії | Очікуваний результат |
| 1.Навести камеру на зображення | На зображенні виникне 3D модель |
| 2. Почати рухати камерою телефона | На зображенні буде залишатися 3D модель |

Таблиця 4.3 – Програвання звуку при пострілі

|  |  |  |
| --- | --- | --- |
| Опис | Програвання звуку при пострілі | |
| Передумови | Гра запущена | |
| Дії | | Очікуваний результат |
| 1.Натиснути на кнопку вогню | | Прозвучить звук пострілу |

Таблиця 4.4 – Реакція кулі на зіткнення

|  |  |  |
| --- | --- | --- |
| Опис | Реакція кулі на зіткнення | |
| Передумови | Гра запущена | |
| Дії | | Очікуваний результат |
| 1.Натиснути на кнопку вогню | | Куля вилітає та при зіткненні з об’єктом зникає |

Таблиця 4.5 – Реакція качки на зіткнення з кулею

|  |  |  |
| --- | --- | --- |
| Опис | Реакція качки на зіткнення з кулею | |
| Передумови | Гра запущена, | |
| Дії | | Очікуваний результат |
| 1.Натиснути на кнопку вогню | | Куля вилітає та при зіткненні з качкою, вона програє анімацію падіння |

Таблиця 4.6 – Відновлення качки

|  |  |  |
| --- | --- | --- |
| Опис | Відновлення качки | |
| Передумови | Гра запущена, качка вбита | |
| Дії | | Очікуваний результат |

Продовження таблиці 4.6

|  |  |
| --- | --- |
| 1.Зачекати 3-5 секунд | Через випадковий проміжок часу, качка має відновитись |

Після проведення тестувань, було встановлено що відеогра не має критичних помилок та дефектів, повністю є працездатною і відповідає очікуваному результату.

## **ВИСНОВКИ**

В ході даної дипломної роботи, було розроблено мобільну гру з елементами доповненої реальності в першому розділі було досліджено розвиток індустрії комп‘ютерних ігор. Були освітленні основні принципи та процеси при розробці мультимедійних ігор. Також була сфорулювана актуальність роботи та визначенні задачі.

В другому розділі було розглянуто найбільш популярні сучасні засоби для розробки ігор. Проведено розбиття їх на класи та проаналізовано їх базові заявлені функціональні можливості. Для більш детального аналізу було обрано по одному найоптимальнішому представнику з кожного класу для розробки гри.

Серед ігрових рушіїв у колі незалежних розробників, останнім часом, найбільшою популярністю користуется Unity. Він повністю задовольняє нас за своїми базовими характеристиками до того ж має потужну підтримку товариства, що значно знижує поріг входження.

Серед додаткових інструментів для AR можна виділити Vuforia Engine котра вміє виявляти та відстежувати зображення по зображенню з камери. Серед систем комп’ютерного зору дуже добре зарекомендовала себе.

У третьому розділі було описано розробку та стиль 3D моделей, а також описанні основні механіки, а саме: ефект сцени в зображенні, стрільба, реєстрація попадань, генерація випадкових маршрутів для качок.

Проаналізувавши ринок, приходимо до того що на даний момент не існує схожих мобільних ігор, тож вона є унікальною.

Розроблений програмний продукт сумісний тільки з операційними системами Android 7.0 та вище. Форма поширення: безкоштовна.

## **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Що таке мобільна гра [Електронний ресурс]: [Веб-сайт] – Електронні дані. – доступу: https://en.wikipedia.org/wiki/Mobile\_game
2. Мобільна гра [Електронний ресурс]: [Веб-сайт] – Електронні дані. – доступу: https://www.techopedia.com/definition/24261/mobile-games
3. Таксономія нового жанру та навіщо вона нам потрібна [Електронний ресурс]: [Веб-сайт] – Електронні дані. – доступу: https://www.gamerefinery.com/new-genre-taxonomy-and-why-we-need-it/
4. Визначення спортивних відеоігр [Електронний ресурс]: [Веб-сайт] – Електронні дані. – доступу: https://engagedfamilygaming.com/parent-resources/video-game-definition-week-sports-games/
5. Rollings, Andrew; Ernest Adams (2006). Fundamentals of Game Design. Prentice Hall. –193 с.
6. Геймінг в доповненій реальності [Електронний ресурс]: [Веб-сайт] – Електронні дані доступуhttps://whatis.techtarget.com/definition/augmented-reality-gaming-AR-gaming
7. "Hyper-Casual Games: Mobile Gaming's Greatest Genre". CleverTap. Retrieved 2020-06-08.
8. Andrew Rollings, Ernest Adams: Andrew Rollings and Ernest Adams on Game Design. New Riders Publishing, 2003, ISBN 1-59273-001 – 267 с.
9. D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, D. Schmalstieg, “Real-time detection and tracking for augmented reality on mobile phones”, IEEE Trans. Vis. Comp. Graph., 16(3), 2010. – 368 с.
10. Mayra F. An Introduction to Game Studies / Frans Mayra. – Thousand Oaks: SAGE Publications Ltd, 2008. – 208 с.

## ДОДАТОК А

## КОД ПРОГРАМИ

1. Duck.cs

using System.Collections;  
using PathCreation.Examples;  
using UnityEngine;  
  
public class **Duck** : MonoBehaviour  
{  
 [SerializeField] [Range(0f, .1f)] private float **fallDownTimeOffset**;  
 [SerializeField] private float **distanceToFallDown**;  
 [SerializeField] private float **timeToFallDown**;  
 [SerializeField] private RandomPathCreator **randomPathCreator**;  
 private Animator \_animator;  
 private PathFollower \_pathFollower;  
  
 private static readonly int *DeathBlend* = Animator.StringToHash("DeathBlend");  
  
 *// Start is called before the first frame update* void **Awake**()  
 {  
 \_animator = GetComponent<Animator>();  
 \_pathFollower = GetComponent<PathFollower>();  
 }  
   
  
 public void DuckShooted(float timeToRespawn, float timeToPlayDeathAnimation)  
 {  
 \_pathFollower.enabled = false;  
 StartCoroutine(PlayDeathAnimationCoroutine(timeToPlayDeathAnimation));  
 StartCoroutine(RespawnCoroutine(timeToRespawn));  
 }  
  
 private void FallDuck()  
 {  
 LeanTween.cancel(gameObject);  
 LeanTween.moveY(gameObject, distanceToFallDown, timeToFallDown).setEase(LeanTweenType.*easeInOutSine*);  
   
 }  
   
 private IEnumerator RespawnCoroutine(float timeToRespawn)  
 {  
   
 randomPathCreator.CreateRandomPath();  
 yield return new WaitForSeconds(timeToRespawn);  
 \_animator.SetFloat(*DeathBlend*,0f);  
 \_pathFollower.enabled = true;  
 }  
  
 private IEnumerator PlayDeathAnimationCoroutine(float timeToPlayDeathAnimation)  
 {  
 FallDuck();  
   
 yield return new WaitForSeconds(fallDownTimeOffset);  
   
 float multiplier = (float) .8f / timeToPlayDeathAnimation;  
   
 float time = 0;  
   
 while (time <= timeToPlayDeathAnimation)  
 {  
 \_animator.SetFloat(*DeathBlend*, time\*multiplier);  
 time += Time.deltaTime;  
   
 yield return null;   
   
 }   
 }   
}

1. DuckCollision.cs

using System;  
using UnityEngine;  
  
public class **DuckCollision** : MonoBehaviour  
{  
 public Duck **duckInfo**;  
 public Action<Duck> **duckShooted**;  
  
 *// Start is called before the first frame update* public void DuckShooted()  
 {  
 duckShooted?.Invoke(duckInfo);  
 }  
   
   
}

1. Bullet.cs

using System.Collections;  
using UnityEngine;  
using System;  
  
public class **Bullet** : MonoBehaviour  
{  
   
 [SerializeField] private Rigidbody **bulletRigidbody**;  
 public Action<Vector3, string> **onBulletCollisionAction**;  
 private float \_bulletSpeed;  
 private Coroutine \_deactivateBullet;  
 private Vector3 \_startPoint= Vector3.zero;  
 private bool \_bulletColided;  
 private Quaternion \_savedCameraRotation;  
 private bool \_gameObjectActive = false;  
  
 private void **Update**()  
 {  
 if (\_gameObjectActive)  
 {  
 BulletMove();  
 }  
 }  
   
 public void Setup(Vector3 startPoint, float bulletSpeed, Transform cameraTransform)  
 {  
 gameObject.SetActive(true);  
 \_gameObjectActive = true;  
 \_bulletColided = false;  
   
 bulletRigidbody.velocity = Vector3.zero;  
 bulletRigidbody.angularDrag = 0;  
 bulletRigidbody.angularVelocity = Vector3.zero;  
  
 \_savedCameraRotation = cameraTransform.rotation;  
   
 \_bulletSpeed = bulletSpeed;  
   
 \_startPoint = startPoint;  
   
 transform.rotation = cameraTransform.rotation;  
 transform.position = startPoint;  
   
 \_deactivateBullet = StartCoroutine(DeactivateBullet(5f));  
 }  
   
 private void BulletMove()  
 {  
 bulletRigidbody.velocity = transform.forward \* \_bulletSpeed \* Time.deltaTime;  
 }  
  
 private void **OnCollisionEnter**(Collision other)  
 {  
 if (!\_bulletColided)  
 {  
 GameObject tempGameObject = other.gameObject;  
  
 if (tempGameObject.CompareTag("Joint"))  
 {  
 \_bulletColided = true;  
 \_gameObjectActive = false;  
 gameObject.SetActive(false);  
 StopCoroutine(\_deactivateBullet);  
 return;  
 }  
 else if (tempGameObject.CompareTag("Target"))  
 {  
 \_bulletColided = true;  
 \_gameObjectActive = false;  
 gameObject.SetActive(false);  
 StopCoroutine(\_deactivateBullet);  
 return;  
 }  
   
 transform.rotation = \_savedCameraRotation;  
 Ray ray = new Ray(\_startPoint, transform.forward);  
   
 if (Physics.Raycast(ray))  
 {  
 Debug.Log(tempGameObject.name);  
 tempGameObject.GetComponent<DuckCollision>().DuckShooted();  
 \_bulletColided = true;  
 \_gameObjectActive = false;  
 gameObject.SetActive(false);  
 onBulletCollisionAction?.Invoke(tempGameObject.transform.position, tempGameObject.tag);  
 StopCoroutine(\_deactivateBullet);  
 }  
 }  
 }  
   
  
 private IEnumerator DeactivateBullet(float time)  
 {   
 yield return new WaitForSeconds(time);  
 \_gameObjectActive = false;  
 gameObject.SetActive(false);  
 }  
}

1. GunFire.cs

using System;  
using System.Collections;  
using UnityEngine;  
  
[RequireComponent(typeof(InstantieateParticles))]  
[RequireComponent(typeof(Animator))]  
public class **GunFire** : MonoBehaviour  
{  
 *// Start is called before the first frame update* [SerializeField] private Transform **arCamera**;  
 [SerializeField] private float **bulletSpeed**;  
 [SerializeField] Transform **bulletStartPosition**;  
 [SerializeField] Bullet [] **bulletsList**;  
 [SerializeField] private AnimationClip **reloadAnimation**;  
  
 private float \_reloadTime;  
   
 public Action<float> **GunReloadingAction**;  
 public Action **GunFireAction**;  
  
 private InstantieateParticles \_instantiateParticles;  
 private Animator \_animator;  
   
 private bool \_isReloading, \_isFiring;  
 private int \_currentRoundsAmmount;  
   
 private static readonly int *IsFire* = Animator.StringToHash("IsFire");  
 private static readonly int *IsReload* = Animator.StringToHash("IsReload");  
  
 private void **Awake**()  
 {  
 \_animator = GetComponent<Animator>();  
 \_instantiateParticles = GetComponent<InstantieateParticles>();  
 }  
 void **Start**()  
 {  
 GunFireAction += GunShoot;  
 GunReloadingAction += ReloadGun;  
   
 \_reloadTime = reloadAnimation.length;  
  
 \_currentRoundsAmmount = bulletsList.Length;  
  
 \_isReloading = false;  
 \_isFiring = false;  
   
 }  
  
 public void **Fire**()  
 {  
 if (!\_isFiring && !\_isReloading && \_currentRoundsAmmount > 0)  
 {  
 GunFireAction?.Invoke();  
 \_currentRoundsAmmount--;  
 \_instantiateParticles.Instantiate();*//instantiate smoke particles* }  
 }  
 *// Update is called once per frame* void **Update**()  
 {  
 *//invoking actions on FireAction (it`s method GunShoot). Instantiating smoke particles* if (Input.GetKeyDown(KeyCode.*Space*))  
 {  
 if (!\_isFiring && !\_isReloading && \_currentRoundsAmmount > 0)  
 {  
 GunFireAction?.Invoke();  
 \_currentRoundsAmmount--;  
 \_instantiateParticles.Instantiate();*//instantiate smoke particles* }  
 }  
 }  
  
 public void EndFireAnimationEvent()*//shows when animation end. When bullets == 0, fire action GunRealoading. Event calls from animation* {  
   
 *//Makes animation going to reload. If bullets == 0, invoking actions on ReloadingAction (it`s method GunShoot, and will be method connected with UI)* if (\_currentRoundsAmmount == 0)  
 GunReloadingAction?.Invoke(\_reloadTime);  
   
 *// makes animation going to idle, if not reloading* \_isFiring = false;  
 \_animator.SetBool(*IsFire*, false);  
   
 }  
   
 private void GunShoot()  
 {  
   
 bulletsList[\_currentRoundsAmmount - 1].Setup(bulletStartPosition.position,  
 bulletSpeed, arCamera);  
   
 \_isFiring = true;  
 \_animator.SetBool(*IsFire*, true);  
 }  
   
  
 private void ReloadGun(float reloadTime)  
 {  
 \_animator.SetBool(*IsReload*, true);  
 \_currentRoundsAmmount = bulletsList.Length;  
 \_isReloading = true;  
  
 StartCoroutine(StopReloadAnimationCoroutine(reloadTime));  
 }  
  
 private IEnumerator StopReloadAnimationCoroutine(float reloadTime)  
 {  
 yield return new WaitForSeconds(reloadTime - .05f);  
   
 \_animator.SetBool(*IsReload*, false);  
 \_isReloading = false;  
 }   
}

1. BulletCollisionHandler.cs

using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.Events;  
  
public class **BulletCollisionHandler** : MonoBehaviour  
{  
 [SerializeField] List<Bullet> **bullets**;  
 [SerializeField] UnityEvent <Vector3,string> **OnBulletCollision**;  
   
   
 *// Start is called before the first frame update* void **Start**()  
 {  
 foreach (var item in bullets)  
 {  
 item.onBulletCollisionAction += OnBulletCollisionEnter;  
 }  
 }  
  
 void OnBulletCollisionEnter(Vector3 collisionPosition, string colliderTag)  
 {  
 OnBulletCollision?.Invoke(collisionPosition, colliderTag);  
 }  
  
 *// Update is called once per frame*}

1. DucksHandler.cs

using System.Collections.Generic;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
  
public class **DucksHandler** : MonoBehaviour  
{  
   
 [SerializeField] private List<GameObject> **ducksPathCreator**;  
 [SerializeField] private List<GameObject> **ducks**;  
 [SerializeField] private List<DuckCollision> **duckCollisions**;  
 [SerializeField] private UIHandler **uiHandler**;  
 [SerializeField] private int **minShotedDuck**;  
 [SerializeField] private int **maxShotedDuck**;  
 [SerializeField] private float **minimalTimeDuckRespawn**;  
 [SerializeField] private float **maximalTimeDuckRespawn**;  
 [SerializeField] private float **timeToPlayDeathAnimation**;  
 [SerializeField] private AudioSource **duckSound**;  
   
 private int activeDuckAmmount;  
 *// Start is called before the first frame update* private void **Awake**()  
 {  
 activeDuckAmmount = Random.Range(minShotedDuck, maxShotedDuck);  
 *//uiHandler.InitializeDucksAmmount(activeDuckAmmount);* foreach (var item in duckCollisions)  
 {   
 item.duckShooted += HandleShootedDuck;  
 }  
 }  
  
 public void HandleShootedDuck(Duck duckInfo)  
 {  
 duckSound.pitch = Random.Range(.9f, 1f);  
 duckSound.Play();  
 uiHandler.AddShotedDuck();  
 duckInfo.DuckShooted(Random.Range(minimalTimeDuckRespawn, maximalTimeDuckRespawn), timeToPlayDeathAnimation);  
 }  
  
 public void **ActivateDucks**()  
 {  
 activeDuckAmmount = Random.Range(minShotedDuck, maxShotedDuck);  
 uiHandler.InitializeDucksAmmount(activeDuckAmmount);  
   
 for (int i = 0; i < activeDuckAmmount; i++)  
 {  
 ducks[i].SetActive(true);  
 ducksPathCreator[i].SetActive(true);  
 }  
 }  
   
 public void **DeactivateDucks**()  
 {  
 for (int i = 0; i < activeDuckAmmount; i++)  
 {  
 ducks[i].SetActive(false);  
 ducksPathCreator[i].SetActive(false);  
 }  
 }  
   
}

1. GunHandler.cs

using UnityEngine;  
using UnityEngine.Events;  
  
public class **GunHandler** : MonoBehaviour  
{  
 [SerializeField] private GunFire **gun**;  
 [SerializeField] private UnityEvent<float> **gunReload**;  
 [SerializeField] private UnityEvent **gunFire**;  
  
 [SerializeField] private AudioSource **gunShotAudio**;  
 *// Start is called before the first frame update* private void **Awake**()  
 {  
 gun.GunFireAction += GunShooted;  
 gun.GunReloadingAction += GunReloading;  
 }  
  
 *// Update is called once per frame* private void GunShooted()  
 {  
 gunShotAudio.Play();  
 gunFire?.Invoke();  
 }  
  
 private void GunReloading(float reloadTime)  
 {  
 gunReload?.Invoke(reloadTime);  
 }  
}

1. UIHandler.cs

using System;  
using System.Collections;  
using TMPro;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
public class **UIHandler** : MonoBehaviour  
{  
 [SerializeField] private StartEndGame **\_startEndGame**;  
 [SerializeField] private TextMeshProUGUI **shotedDucksAmmountText**;  
 [SerializeField] private GameObject **reloadingText**;  
 [SerializeField] private RectTransform **maskingPanel**;  
  
 private int \_ducksAmmountToShot;  
 private int \_shotedDucks;  
 *// Start is called before the first frame update* public void InitializeDucksAmmount(int ducksAmmount)  
 {  
 \_ducksAmmountToShot = ducksAmmount;  
 \_shotedDucks = 0;  
   
 shotedDucksAmmountText.text = $"{\_shotedDucks}/{\_ducksAmmountToShot}";  
 shotedDucksAmmountText.enabled = true;  
 }  
  
   
   
 public void **DeactivateDuckAmmountText**()  
 {  
 shotedDucksAmmountText.enabled = false;  
 }  
 public void **AddShotedDuck**()  
 {  
 \_shotedDucks++;  
  
 if (\_shotedDucks == \_ducksAmmountToShot)  
 {  
 \_startEndGame.ActivateEndGame();  
 }  
   
 shotedDucksAmmountText.text = $"{\_shotedDucks}/{\_ducksAmmountToShot}";  
 }  
   
 public void **ShowReloadUI**(float reloadTime)  
 {  
 reloadingText.SetActive(true);  
 StartCoroutine(ReloadAnimationCoroutine(reloadTime));  
 }  
   
 private IEnumerator ReloadAnimationCoroutine(float reloadTime)  
 {  
 float time = 0;  
 float lerpXScale = 0;  
  
 while (time <= reloadTime)  
 {  
 time += Time.deltaTime;  
 lerpXScale = Mathf.Lerp(0f, 1, (float) time / reloadTime);  
 maskingPanel.localScale = new Vector3(lerpXScale, 1f, 1f);  
 yield return null;  
 }  
  
 reloadingText.SetActive(false);  
 }  
}

1. CameraDuckClipPlane.cs

using UnityEngine;  
using Vuforia;  
  
public class **CameraDuckClipPlane** : MonoBehaviour  
{  
 [SerializeField] private TrackableBehaviour **trackableBehaviour**;  
 [SerializeField] private Camera **arCamera**;  
 [SerializeField] private Camera **mainCamera**;  
 [SerializeField] private Transform **stencilPlane**;  
  
  
 private bool \_imageTracked = false;  
 *// Start is called before the first frame update* void **Start**()  
 {  
 trackableBehaviour.RegisterOnTrackableStatusChanged(OnTrackableStateChanged);  
 }  
  
 *// Update is called once per frame* void **Update**()  
 {  
 if (\_imageTracked)  
 {  
 mainCamera.farClipPlane = Vector3.Distance(stencilPlane.transform.position, mainCamera.transform.position);  
 }  
 }  
  
 public void OnTrackableStateChanged(TrackableBehaviour.StatusChangeResult obj)  
 {  
 if (obj.NewStatus == TrackableBehaviour.Status.*TRACKED*)  
 {  
 \_imageTracked = true;  
 mainCamera.fieldOfView = arCamera.fieldOfView;  
 }  
 }  
}

1. InstantiateAfterDeath.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class **InstantiateAfterDeath** : MonoBehaviour  
{  
 [SerializeField] private float **secondsToDisable**;  
 [SerializeField] private float **headYTitleOffset**;  
 [SerializeField] private float **valuesYTitleOffset**;  
 [SerializeField] private List<GameObject> **headshotTitle**;  
 [SerializeField] private List<GameObject> **fiftyPointsTitle**;  
 [SerializeField] private List<GameObject> **hundredPointsTitle**;  
   
   
 *// Start is called before the first frame update  
   
  
 // Update is called once per frame* public void **InstantiateObjects**(Vector3 collidedPosition, string coliderTag)  
 {  
   
  
 if (coliderTag == "Head")  
 {  
 foreach (var item in headshotTitle)  
 {  
 if (!item.activeSelf)  
 {  
 item.transform.position = collidedPosition + Vector3.up \* headYTitleOffset;  
 item.SetActive(true);  
 StartCoroutine(DisableTitle(item));  
 break;  
 }  
 }  
   
 foreach (var item in hundredPointsTitle)  
 {  
 if (!item.activeSelf)  
 {  
 item.transform.position = collidedPosition + Vector3.up \* valuesYTitleOffset;  
 item.SetActive(true);  
 StartCoroutine(DisableTitle(item));  
 break;  
 }  
 }  
   
 }  
 else  
 {  
 foreach (var item in fiftyPointsTitle)  
 {  
 if (!item.activeSelf)  
 {  
 item.transform.position = collidedPosition + Vector3.up \* valuesYTitleOffset;  
 item.SetActive(true);  
 StartCoroutine(DisableTitle(item));  
 break;  
 }  
 }  
 }  
 }  
  
 private IEnumerator DisableTitle(GameObject title)  
 {  
 yield return new WaitForSeconds(secondsToDisable);  
 title.SetActive(false);  
 }  
   
   
}

1. InstantieateParticles.cs

using System.Collections;  
using UnityEngine;  
  
public class **InstantieateParticles** : MonoBehaviour  
{  
 *// Start is called before the first frame update* [SerializeField] float **secondsToContinue**;  
 [SerializeField] Transform **instantiatePosition**;  
 [SerializeField] GameObject **particle**;  
   
  
 *// Update is called once per frame* public void Instantiate()  
 {  
 StartCoroutine(SpawnFireParticle());  
 }  
  
 IEnumerator SpawnFireParticle()  
 {  
 Destroy(Instantiate(particle, instantiatePosition.position, instantiatePosition.rotation),secondsToContinue);  
 yield return new WaitForSeconds(secondsToContinue);  
   
  
 }  
}

1. MaterialQueue.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class **MaterialQueue** : MonoBehaviour  
{  
 [SerializeField] private List<Material> **materials**;  
   
 [SerializeField] private int **materialQueue**;  
 *// Start is called before the first frame update* void **Start**()  
 {  
 foreach (var item in materials)  
 {  
 item.renderQueue = materialQueue;  
 }  
 }  
  
 *// Update is called once per frame*}

1. ObjectFollow.cs

using UnityEngine;  
  
public class **ObjectFollow** : MonoBehaviour  
{  
 [SerializeField] private Transform **objectToFollow**;  
 [SerializeField] private Vector3 **positionOffset**;  
  
 *// Start is called before the first frame update  
   
 // Update is called once per frame* void **Update**()  
 {  
 transform.position = objectToFollow.position + positionOffset;  
 transform.rotation = objectToFollow.rotation;  
 }  
}

1. RandomPathCreator.cs

using System;  
using PathCreation;  
using System.Collections.Generic;  
using System.Linq;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
  
public class **RandomPathCreator** : MonoBehaviour  
{  
 [SerializeField] private List<Transform> **regularPositions**;  
 [SerializeField] private List<Transform> **startPositions**;  
 [SerializeField] private int **minPositions**;  
 [SerializeField] private int **maxPositions**;  
 [SerializeField] private PathCreator **\_pathCreator**;  
  
 private List<Transform> tempRegularPositions;  
  
   
  
 private void **OnEnable**()  
 {  
 tempRegularPositions = new List<Transform>();  
 CreateRandomPath();  
 }  
  
 public void CreateRandomPath()  
 {  
 tempRegularPositions = regularPositions.ToList();  
 int positionsAmmount = Random.Range(minPositions, maxPositions);  
   
  
 List<Transform> tempList = new List<Transform>();  
  
 tempList.Add(startPositions[Random.Range(0, startPositions.Count - 1)]);  
  
 int tempIndex = 0;  
   
 for (int i = 0; i < positionsAmmount; i++)  
 {  
 tempIndex = Random.Range(0, tempRegularPositions.Count - 1);  
 tempList.Add(tempRegularPositions[tempIndex]);  
 tempRegularPositions.RemoveAt(tempIndex);  
 }  
  
 BezierPath bezierPath = new BezierPath(tempList, true, PathSpace.*xyz*);  
 bezierPath.GlobalNormalsAngle = 270f;  
 bezierPath.ControlPointMode = BezierPath.ControlMode.*Free*;  
   
 \_pathCreator.bezierPath = bezierPath;  
   
 }  
  
}

1. ShakeRotation.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class **ShakeRotation** : MonoBehaviour  
{  
 [SerializeField] Vector3 **rotationAxis**;  
 [SerializeField] float **turnDegree**;  
 [SerializeField] float **periodRotationTime**;  
 [SerializeField] float **overallRotationTime**;  
 [SerializeField] float **tweenDelay**;  
 private void **OnEnable**()  
 {  
 LTDescr rotateAnimation = LeanTween.rotateAroundLocal(gameObject, rotationAxis, turnDegree, periodRotationTime)  
 .setEaseShake()  
 .setRepeat(-1)  
 ;  
  
 LeanTween.value(gameObject, turnDegree, 0f, overallRotationTime).setDelay(tweenDelay).setOnUpdate(  
 (float value) => {  
 rotateAnimation.setTo(Vector3.right \* value);  
 }  
 ).setEase(LeanTweenType.*easeOutQuad*);  
 }  
 *// Start is called before the first frame update*}

1. StartEndGame.cs

using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.Events;  
  
public class **StartEndGame** : MonoBehaviour  
{  
   
 [SerializeField] private List<TweenMoveObject> **moveGates**;  
 [SerializeField] private TweenRotateObject **rotateGate**;  
 [SerializeField] private Target **startTarget**;  
 [SerializeField] private UnityEvent **onGameStart**;  
 [SerializeField] private UnityEvent **onGameEnd**;  
 *// Start is called before the first frame update* private void **Awake**()  
 {  
 startTarget.onTargetCollision += OnGameStart;  
 startTarget.onTargetCollision += StartGame;  
   
 }  
 *//  
 // Update is called once per frame* public void StartGame()  
 {  
 startTarget.ActiveMovement(false);  
   
 foreach (var item in moveGates)  
 {  
 item.MoveObject(false);  
 }  
   
 rotateGate.RotateObject(false);  
 }  
   
 public void EndGame()  
 {  
 startTarget.ActiveMovement(true);  
   
 foreach (var item in moveGates)  
 {  
 item.MoveObject(true);  
 }  
   
 rotateGate.RotateObject(true);  
 }  
  
 private void OnGameEnd()  
 {  
   
 onGameEnd?.Invoke();  
 }  
  
 private void OnGameStart()  
 {  
 onGameStart?.Invoke();  
 }  
  
 public void ActivateEndGame()  
 {  
 EndGame();  
 OnGameEnd();  
 }  
}

1. Target.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class **Target** : MonoBehaviour  
{  
 [SerializeField] private TweenMoveObject **targetTweenMoveObject**;  
 [SerializeField] private GameObject **title**;  
 public Action **onTargetCollision**;  
 *// Start is called before the first frame update* private void **OnCollisionEnter**(Collision other)  
 {  
   
 onTargetCollision?.Invoke();  
 StartCoroutine(DisableObjects());  
 }  
  
 public void ActiveMovement(bool inversion)  
 {  
 title.SetActive(true);  
 gameObject.SetActive(true);  
 targetTweenMoveObject.MoveObject(inversion);  
 title.GetComponent<TweenMoveObject>().MoveObject(inversion);  
 }  
  
   
   
 private IEnumerator DisableObjects()  
 {  
 yield return new WaitForSeconds(.5f);  
 title.SetActive(false);  
 gameObject.SetActive(false);  
 }  
  
 public void Reset()  
 {  
   
 }  
}

1. UIShellsFall.cs

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class **UIShellsFall** : MonoBehaviour  
{  
   
 [SerializeField] private Rigidbody [] **shells**;  
 [SerializeField] private float **secondsToStopShellFall**;  
  
 private int \_fallShellsCount = 0;  
  
 private List<Bounds> \_shellsBounds;  
 private List<Vector3> \_startShellsPosition;  
 *// Start is called before the first frame update* void **Start**()  
 {  
 \_startShellsPosition = new List<Vector3>();  
 \_shellsBounds = new List<Bounds>();  
  
 foreach (var item in shells)  
 {  
 \_startShellsPosition.Add(item.position);  
 \_shellsBounds.Add(item.GetComponent<MeshCollider>().bounds);  
 }  
   
 }  
  
 *// Update is called once per frame* public void **Reload**(float reloadTime)  
 {  
 \_fallShellsCount = 0;  
 StartCoroutine(ReloadCoroutine(reloadTime));  
 }  
   
 public void **FallShell**()  
 {  
 float yOffset = Random.Range(-\_shellsBounds[\_fallShellsCount].extents.y, \_shellsBounds[\_fallShellsCount].extents.y);  
 float xOffset = Random.Range(-\_shellsBounds[\_fallShellsCount].extents.x, \_shellsBounds[\_fallShellsCount].extents.x);  
 float zOffset = Random.Range(0f, \_shellsBounds[\_fallShellsCount].extents.z);  
 *// float yOffset = Random.Range(-\_shellsBounds[\_fallShellsCount].extents.y, \_shellsBounds[\_fallShellsCount].extents.y);  
 // float xOffset = Random.Range(-\_shellsBounds[\_fallShellsCount].extents.x, \_shellsBounds[\_fallShellsCount].extents.x);  
 // float zOffset = Random.Range(0f, \_shellsBounds[\_fallShellsCount].extents.z);* Vector3 hitPosition = \_shellsBounds[\_fallShellsCount].center + new Vector3(xOffset, yOffset, 0f);  
  
 Vector3 hitForce = new Vector3( Random.Range(0, 150f),Random.Range(-50f,50f),100f);*//Random.Range(0, 150f), Random.Range(-50f,50f), Random.Range(-150f, 150f)* shells[\_fallShellsCount].useGravity = true;  
 shells[\_fallShellsCount].isKinematic = false;  
  
 shells[\_fallShellsCount].AddForceAtPosition(hitForce, hitPosition);  
  
 StartCoroutine(StopShellFromFalling(shells[\_fallShellsCount]));  
   
 \_fallShellsCount++;  
   
 }  
  
 private IEnumerator StopShellFromFalling(Rigidbody rigidbody)  
 {  
 yield return new WaitForSeconds(secondsToStopShellFall);  
 rigidbody.useGravity = false;  
 rigidbody.velocity = Vector3.zero;  
 rigidbody.angularVelocity = Vector3.zero;  
 }  
  
 private IEnumerator ReloadCoroutine(float reloadTime)  
 {  
 yield return new WaitForSeconds(reloadTime);  
   
 for (int i = 0; i < shells.Length; i++)  
 {  
 shells[i].isKinematic = true;  
 shells[i].useGravity = false;  
 shells[i].velocity = Vector3.zero;  
 shells[i].angularVelocity = Vector3.zero;  
 shells[i].transform.localPosition = Vector3.zero;  
 shells[i].transform.localRotation = Quaternion.identity;  
 shells[i].transform.eulerAngles = new Vector3(90f, 0f, 0f);  
 }  
 }  
}