МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний ігровий додаток «Laser Beam»»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-91/2 Хвайра Таєр Самер Тавфік

Кваліфікаційна робота захищена на засіданні ЕК з оцінкою _____ «__» _____2020 р.

Науковий керівник _____ к.т.н., доц., Федотова Н.А.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2020

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«___» _____ 2020 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові

Хвайрі Таєру Самеру Тавфіку
(прізвище, ім'я, по батькові)

1 Тема проекту *Ігровий мобільний додаток «Laser Beam»*

затверджена наказом по університету від « 26 » листопада 2020 р. № 1824-III

2 Термін здачі студентом закінченого проекту «07» грудня 2020 р.

3 Вхідні дані до проекту *технічне завдання*

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *аналіз предметної області, постановка задачі та методи дослідження, проектування гіпер-казульного додатку, реалізація ігрового мобільного додатку*

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *актуальність роботи, мета та задачі, інфографіка використання ігрових механік у гіпер-казуальних додатках, функціональні вимоги до додатку, проектування гри, діаграма варіантів використання, етапи розробки ігрового додатку, засоби реалізації, процес створення прототипу ігрового додатку, демонстрація меню вибору ігрового рівня, процес розробки ігрового меню, демонстрація ігрового процесу, процес розробки гейдеру лазера, демонстрація функції перегляду результатів, висновки, апробація результатів роботи*

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Отримання завдання на проект	05.09.20-09.09.20	
2	Дослідження предметної області та аналогічних рішень	10.09.20-24.09.20	
3	Формування вимог до проект	25.09.20-09.10.20	
4	Оформлення розділу з планування ІТ-проекту	04.11.20-11.11.20	
5	Проектування додатку	12.11.20-18.11.20	
6	Розробка прототипу додатку	19.11.20-20.11.20	
7	Розробка ігрових інтерфейсів	21.11.20-24.11.20	
8	Розробка спрайтів	25.11.20-27.11.20	
9	Створення шейдерів, та налаштування візуальних ефектів	28.11.20-30.11.20	
10	Налаштування фінального збірки, та компілювання ігрового додатку	01.12.20-02.12.20	
11	Тестування додатка	03.12.20-07.12.20	
12	Представлення роботи	08.12.20-21.12.20	

Магістрант _____ Хвайра Т.С.Т.

Керівник роботи _____ к.т.н., доц. Федотова Н.А.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Ігровий мобільний додаток «Laser Beam»».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 38 найменувань, та 2 додатки. Загальний обсяг роботи − 124 сторінок, у тому числі 90 сторінок основного тексту, 4 сторінки списку використаних джерел, 26 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці ігрового мобільного додатку «Laser Beam» із використанням у ній механіки типу головоломка.

В першому розділі роботи було проведено аналіз предметної області з дослідженням існуючих ігрових механік, що використовуються у подібних проектах, та детальний огляд існуючих гіпер-казуальних ігор.

Визначення мети проекту, задач та засобів реалізації проведено у другому розділі. Проведено вибір засобів реалізація.

Третій розділ присвячено структурно-функціональному моделюванню.

У останньому розділі продемонстрований процес створення прототипу, розробки ігрового інтерфейсу, створення шейдерів оточення, виконане налаштування пост-обробки гри. Після реалізації проекту проведено тестування на працездатність додатка.

Результатом роботи є мобільний ігровий додаток «Laser Beam», представлений у вигляді .apk файлу.

Практичне значення роботи полягає у популяризації ігрової механіки типу головоломка, що допоможе отримувати користь від гри у мобільні додатки.

Ключові слова: мобільний ігровий додаток, гіпер-казуальна гра, ігрова механіка, механіка типу головоломка.

ЗМІСТ

ВСТУП

Ігри випадково перетворились на великий бізнес.

Протягом 50 років відеоігри пройшли шлях від проектів для любителів до проектів що заробляють понад 100 мільярдів доларів на рік. За деякими підрахунками, ігрова індустрія готується подвоїти ці і без того вражаючі цифри до 2023 року [1].

Мобільні ігри є рушійною силою нещодавнього зростання всієї ігрової галузі, а гіпер-казуальні ігри – рушійною силою додатків для мобільних пристроїв, і ці ігри все частіше очолюють чарти Google Play Market та AppStore.

Voodoo, провідний розробник гіпер-казуальних ігор, постійно залучає нових користувачів і підтримує їх. У прес-релізі від травня 2018 року вони повідомили про 150 мільйонів активних гравців щомісяця та в цілому про 300 мільйонів завантажень у 2017 році [2]. Зовсім нещодавно Voodoo повідомив про понад 1,5 мільярда завантажень у своєму портфоліо ігор [3].

Актуальність дипломної роботи зумовлена, великою популярністю гіпер-казуальних ігор, але низькою популярністю механіки головоломки, яка є одною з корисніших у плані розвитку ніж інші.

Предметна область проекту - гіпер-казуальні ігри, через те що саме даний жанр є одним з найголовніших у індустрії на даний момент часу. Виходячи із вибору механіки можна сказати, що зараз її використання дуже лімітоване, але є найбільш корисним при застосуванні, тому що розвивають логічне та критичне мислення у гравців. Саме тому головоломки є найбільш привабливою механікою, у порівнянні з іншими, яка може забезпечити зацікавленість гравця в той же самий час приносячи користь для його розвитку.

Предмет дослідження – ігрова механіка типу головоломка та її користь для гравців.

Об’єкт дослідження – ігрові механіки у рамках жанру гіпер-казуальних ігор.

Мета дипломної роботи: розробка ігрового мобільного додатку «Laser Beam» для підвищення логічної та критичної мозгової активності.

Для досягнення мети дипломної роботи був визначений перелік завдань:

– розглянути основні ігрові механіки сьогодення у рамках гіпер-казуального жанру, та проаналізувати їх особливості;

– проаналізувати стан гіпер-казуальних ігор у світі на даний момент часу;

– на основі даних аналізу спроектувати ігровий мобільний додаток «Laser Beam»;

– на основі створеного проекту, розробити ігровий додаток «Laser Beam».

Методи дослідження:

– теоретичні та емпіричні (аналіз та узагальнення досвіду використання ігрової механіки типу головоломка);

– системно-функціональні та моделювання (створення структурно-функціональної моделі процесу розробки мобільного ігрового додатку; створення моделі ігрового додатку, а саме діаграми варіантів використання).

Практичне значення полягає у розробці мобільного ігрового додатку «Laser Beam» для покращення логічного та критичного типів мислення у гравців.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний аналіз стану гіпер-казуальних ігор у світі

У даний момент участь ігор у житті людей відіграє дуже важливу роль. Домінуючу роль на разі займає жанр гіпер-казуальних ігор. Люди грають, щоб скоротати час, або просто отримати дозу позитиву [4].

Уже зараз зрозуміло, що неможливо забороняти людям грати, але можливо зробити ігри корисними.

Велику кількість цікавих механік була винайдена с 2017 року, року розквіту гіпер-казуальних ігр. Значна кількість із цих механік є веселими, але не несуть ніякої користі для кінцевого користувача [5]. Також можна зазначити, що чомусь саме механіки які є найменш корисними стали найбільш популярними. Таку тенденцію треба змінювати [6].

До основних механік гіпер-казуальних ігор можна віднести:

– механіку спритності (Dexterity);

– механіку укладання (Stacking);

– механіку падіння (Falling).

Всі ці механіки, нажаль, не відповідають потребі зробити ігри корисними, тому необхідність збільшення відсотку використання механіки головоломок неминуче [7].

Так склалося, що життя сучасної людини тісно пов’язано з телефонами та з іграми, саме тому актуальною є задача створення гіпер-казуального ігрового продукту для підвищення логічного та критичного мислення використовуючи механіку головоломки.

1.2 Детальний огляд існуючих гіпер-казуальних проектів та їх механік

Зараз існує велика кількість механік гіпер-казуальних ігор, які використовуються у багатьох компаніях світу таких як: Voodoo, Ketchapp та інших. Ці механіки були розроблені не дуже давно, а тому вони досі виглядають цікаво і ефективно [8].

Нижче будуть розглянуті основні та самі популярні гіпер-казуальні додатки та їх механіки, які є ефективними та використовуються у великій кількості ігор по всьому світу. Серед розглянутих механік оберемо ті які найбільше підійдуть для розробки проєкту. Механіки які дозволять розробити додаток не лише для розваг, а і для розвитку здібностей.

Тап / Таймінг механіка

Тап і таймінг є найпопулярнішими механіками для гіпер-казуальних ігор. Більшість інших механік використовують тап та таймінг як метод введення для свого конкретного ігрового процесу. В той час як у грі, яка є чистим геймплеєм тапу та таймінгу, механіка повністю покладається на точне натискання або точний час цього натискання. Точність є найважливішим аспектом гри, а основним акцентом для користувача є досконаліть. Тільки ідеальний тап принесе максимальний бал. Решта ігрових відчуттів і креативність покладається на використання дрібних неточностей у натисканні, щоб зменшити здатність гравця перемагати, як правило, у формі високого балу. Гра Baseball Boy від Voodoo (рис. 1.1) фокусує увагу гравців на одному ударі бейсбольною битою, як на єдину дію гравця. Кожен удар вражає, але ідеальний удар значно кращий [9].

Рисунок 1.1 – Приклад тап / таймінг механіки

Думаючи про механіку тапу та таймінгу, потрібно позбутися від будь-яких зовнішніх факторів для гравця і забезпечити чітке візуальне завдання для досягнення гравцем поставленої мети. Візуальний зворотний зв’язок тут надзвичайно важливий з чітким відображенням невдалого пострілу, але також великим позитивним відображенням ідеального пострілу. Чим чіткіший гол і чим сильніший ідеальний постріл, тим цікавіше стає при його проведенні.

Механіка укладання (Stacking)

Механіки укладання дозволяє просуватися далі, додаючи результат вашого попереднього відбору до ходу раунду. Гра – це хороший приклад, коли сама вежа складається з раніше складених квадратів. Кожного разу, коли гравцеві не вдається отримати ідеальний стек, сама вежа стискається, роблячи її

складнішою та меншою для наступного стека [10]. Гра The Tower від Ketchapp (рис. 1.2) – це хороший приклад, коли сама вежа складається з раніше складених квадратів. Кожного разу, коли гравцеві не вдається отримати ідеальний стек, вежа стискається, роблячи її складнішою та меншою для наступного стека.

Рисунок 1.2 – Приклад механіки укладання

Думаючи, як розробити механіку укладання, необхідно переконатись, що гравці мають достатньо спроб (5-10) перед тим, як закінчити раунд, але також необхідно переконатись, що складність досить велика, щоб гравці отримували неідеальні спроби принаймні у 20 - 40% випадків. Занадто мало невдалих спроб – гра занадто важка, в той час як занадто багато ідеальних спроб – гра стає занадто простою.

Механіка спритності (Dexterity)

Такі ігри в основному орієнтовані на гравця, який виконує дуже просту і повторювану дію, яку він повинен виконати багато сотень разів. Маючи достатню кількість практики, ці механіки можуть освоїти спритні гравці, і тому найвищий бал – це чесне відображення спритності та майстерності. Щоб ці ігри були веселими, гра, як правило, повинна пришвидшитися [11]. Обравши механіку, яка може бути легкою - дії виконується повільно, але коли тиск через час стає все більшим, імовірність помилки стає дуже великим.

Механіка потребує чіткої жорсткої межі успіху, як правило, одне життя або одна помилка закінчують раунд, і гравець починаєте з початку. Timberman від Digital Melody (рис. 1.3) – чудовий приклад того, як використати увагу, час і спритність гравця, щоб створити складні завдання на основі очок [12]. При розробці управління та чутливість введення є найвищим пріоритетом. Тут не може бути затримок і сірих зон, дія гравців повинна негайно впливати на персонажа. Гравець буде вводити багато сотень натискань за раунд, кожне натискання повинно бути точним, щоб це було весело, будь-які неточності або затримки помножуються на кількість разів.

Рисунок 1.3 – Приклад механіки спритності

Механіка підйому / падіння (Rising / Falling)

Механіка підйому або падіння націлена створити цікаву подорож для користувача. Постійне прогресування рівня призводить до відчуття прогресування без зміни механіки або цілі. Щоб розважати людей, рівень повинен розвиватися сам. Rise Up від Serkan Özyılmaz та Helix Jump від Voodoo (рис. 1.4) показують, як розвивається прогресія коли ви рухаєтеся вгору або вниз по грі [13].

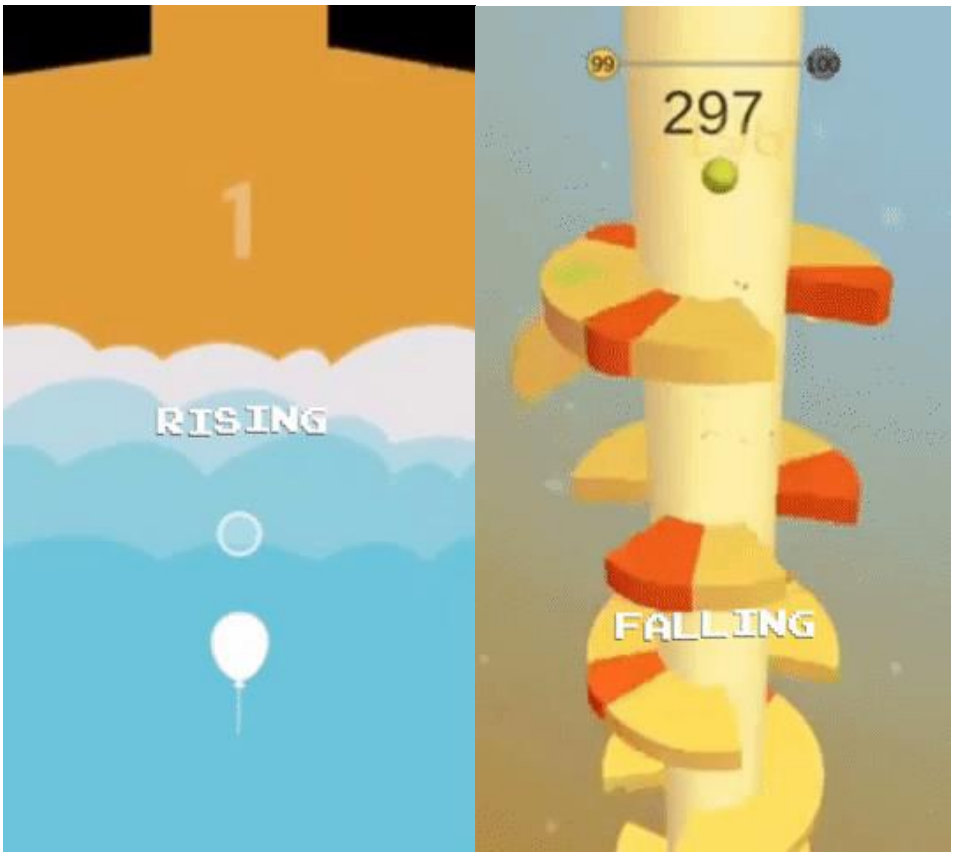

Рисунок 1.4 – Приклад механік падіння та підйому

Фокус гравця зосереджений на вирішенні наступного виклику в ході прогресування, а не на точності. Є багато способів пройти ці рівні, часто потрібно трохи удачі. Ваша єдина мета – захистити об’єкт від ворогів, та досягти поставленої мети.

Подорож створює мале середовище, таким чином гравці самі собі створюють безліч проблем. Невеликі проблеми на початку можуть перейти в набагато складніші проблемі пізніше. Правильний вибором тут є фокусується на гравцях, які одночасно повинні вирішити одну або навіть дві проблеми. Але характер створюваної проблеми змінюється в міру того, як ви піднімаєтеся або падаєте під час ігрового процесу.

Механіка зростання (Growth)

Дана механіка дуже схожа на механіку бездіяльності (idle), оскільки вона, як правило, не залежить від вхідного елемента управління, але формує основну мету ігрового процесу. Переможцем в цьому жанрі ігр завжди є найбільший гравець, і в деяких випадках гравці можуть їсти інших гравців, по суті закінчуючи раунд. Механіка ігрового процесу сама по собі дуже чітка, але все ж створює цікавий ігровий досвід [14].

При розробці потрібно багато думати про щільність гравців, що будуть зростати у процесі гри. Очевидно, що всі гравці хочуть рости, але не всі можуть. Початок із правильною кількістю гравців у правильному просторі та з правильною кількістю їжі – це те, що робить цей жанр цікавим. Дані ігри також стають експоненціально веселішими з реальними ворогами, які грають разом з ними, Такі додатки зараз представляють собою цілий жанр .io (рис. 1.5) ігр у індустрії, хоча кількість ігрових механік для цього жанру обмежена, але у ігор триваліший термін життя, ніж у інших гіпер-казуальних ігор через взаємодію з іншими гравцями [15].

Рисунок 1.5 – Приклад ігрової механіки зростання

Механіка головоломки (Puzzle)

Головоломка – це жанр сам по собі, але гіпер-казуальні головоломки зосереджені на простоті, а не на складності. Якісна гіпер-казуальна головоломка, як правило, не має кінця. Користувача просять продовжувати грати в головоломку якомога довше, і гра не збільшить складності. Сама механіка повинна ускладнюватися через дії користувачів. Хорошими прикладами є 1010! Від компанії Gram Games або 2048 від Ketchapp (рис. 1.6). В обох випадках правила головоломки встановлюються на початку, а ігрова поверхня розвивається під час гри [16]. На відміну від інших настільних ігор, таких як шахи або шашки, які мають чіткі кінцеві цілі, гіпер-казуальні головоломки, як правило, не мають чіткого кінця, і це просто марафон у якому ви повинні протриматися якомога довше.

Рисунок 1.6 – Приклад механіки головоломки

Жанр головоломок вважається найважчим у розробці, оскільки це, як правило, дуже чітка механіка, унікальна для самої гри. Це пов’язано з тим, що дуже важко створити механіку, яка з часом не перетворює гру на щось занадто просте або занадто складне.

Розглянувши найпопулярніші механіки гіпер-казуальних ігор, була обрана основна та допоміжна механіки для майбутнього додатку. У якості основної механіки була обрана головоломка через те що на відміну від інших вона найбільш корисна для користувача. У якості додаткової механіки була обрана механіка спритності. Дана механіка буде імплементована завдяки таймеру та обмеженій кількості натискань.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі

За отриманим результатом аналізу предметної області була сформульована мета дипломної роботи: розробка ігрового мобільного додатку «Laser Beam» для розвитку логічного та критичного типів мислення.

Для досягнення мети дипломної роботи був визначений перелік завдань:

– Проаналізувати предметну область та виконати пошук прототипів

– визначити методи та інструментарій розробки;

– провести планування, документування робіт з IT-проекту;

– спроектувати функціонал мобільного додатку,

– розробити спрайти ігрового продукту;

– розробити механіки головоломки та спритності;

– розроботи інтерфейс ігрового продукту;

– налаштувати сцени та розробити ігрові рівні;

– протестувати ігровий додаток.

Мобільний додаток повинен забезпечувати веселе та корисне проводження часу у грі. Також однією с задач є підвищення швидкості та реакції через встановлені у додатку ігрові обмеження.

Використання ігрового додатку спрямоване на підвищення логічного та критичного видів мислення. Також однією з основних задач таких ігрових додатків є допомога людям зробити цікавішим та кориснішим час їх очікування до будь-якої основної події. Статистично доведено, що найчастіше люди грають в мобільні ігри саме коли чогось ждуть [17].

Впровадження ігрового додатку «Laser Beam» передбачає такі функції:

– вибір рівня гри;

– проходження обраного рівня гри;

– вибір складності гри;

– зміна ассетів (кольорової гамми) гри;

– перегляд рахунку за пройдені рівні.

2.2 Методи дослідження

Для досягнення поставлених задач були виділені основні методи дослідження, а саме:

– теоретичні та емпіричні (аналіз та узагальнення досвіду використання ігрової механіки типу головоломка). Метод аналізу буде використаний для аналізу предметної області, та для більш глибшого розуміння існуючих механік гіпер-казуальних ігр, їх позитивних та негативних сторін;

– системно-функціональні та моделювання (створення структурно-функціональної моделі процесу розробки мобільного ігрового додатку; створення моделі ігрового додатку, а саме діаграми варіантів використання). За допомогою даного методу будуть створенні моделі: IDEF0 [18] – для розуміння предметної області, та більш глибокого розуміння процесу розробки мобільного ігрового додатку; UML діаграм [19], а саме діаграма варіантів використання (Use Case [20]) – для детального розуміння розмежування доступу гравців до ігрового додатку, та можливості використання певними групами користувачів певну кількістю функцій.

Також було виконане планування робіт, створений календарний план та виконана оцінка ризиків які знаходяться у додатку А.

2.3 Вибір засобів реалізації

Для реалізації мобільного ігрового додатку зараз доступні два основні ігрові рушія та прив’язаних до них мов програмування. Тому у якості найбільш простих у використанні, та комфортних у роботі рушіїв гри було обрано два основні:

– Unreal Engine;

– Unity.

Unreal Engine – ігровий рушій, розроблений компанією Epic Games [21], вперше продемонстрований в шутері від першої особи 1998 року Unreal. Хоча спочатку рушій був розроблений для шутерів від першої особи, його також використовували в багатьох інших жанрах, включаючи платформери, файтинги, MMORPG та інші RPG. Написаний на C ++, Unreal Engine відрізняється високим ступенем портативності, підтримуючи широкий спектр платформ. Також рушій підтримує розробку за допомогою блупринтів [22].

Unity – крос-платформний ігровий рушій, розроблений компанією Unity Technologies. У червні 2005 року вперше анонсований та випущений на всесвітній конференції для розробників від компанії Apple Inc. як ексклюзивний ігровий рушій для системи Mac OS X. Станом на 2018 рік підтримка була розширена до більш ніж 25 платформ. Рушій може бути використаний для створення тривимірних, двовимірних ігор, ігор з віртуальною та доповненою реальностями, а також моделювання та іншого досвіду [23]. Рушій також став часто використовуватись за межами відеоігор, а також у кіно, автомобілебудуванні, архітектурі, машинобудуванні та будівництві.

Тому саме Unity був обраний у якості ігрового рушія через його кросплатформеність, та простоту розробки.

Кожен із представлених вище рушіїв підтримує розробку функцій на певній мові програмування:

– C++ в Unreal Engine;

– C# в Unity.

C++ – мова програмування високого рівня, що була розроблена у 1979 році та є похідною від мови програмування C. Синтаксис мови майже повністю запозичений з C. Мова програмування є суворо типізованою [24]. Підтримує такі парадигми програмування:

– процедурна;

– об'єктно-орієнтована;

– узагальнена.

C# – об’єктно-орієнтована мова програмування, яка також відноситься до C-подібних мов. Синтаксис мови дуже схожий з синтаксисом таких мов як C++ та Java. Була розроблена під впливом вказаних вище мов і ввібрала в себе лише найкращі їх сторони та фічі, разом з тим позбулась майже від усіх недоліків своїх попередників. Має строгу статичну типізацію, була створена спеціально для програмування з використанням платформи .Net [25].

Дана мова програмування як ніяка інша підходить для розробки мобільного ігрового додатку.

Через те що у якості мови програмування було обрано C# можна виділити два основних можливих середовища розробки:

– Visual Studio Code;

– Visual Studio.

Visual Studio – кросплатформене інтегроване середовище, що підтримує велику кількість мов програмування [26].

Має велику кількість переваг, таких як:

– значна база плагінів;

– вбудована система контролю версій;

– автоматичне доповнення та виправлення коду;

– розповсюджуєтеся безкоштовно;

– спільнота розробників, які готові допомогти;

– доступна та проста документація;

– можливість використання середовища на різних операційних системах.

Visual Studio Code – безкоштовний редактор вихідних кодів, створений Microsoft для Windows, Linux та macOS [27].

При використанні можна виділити такі переваги:

– дебагінг;

– підсвічування синтаксису;

– інтелектуальне заповнення коду;

– розповсюджуєтеся безкоштовно;

– рефакторинг коду;

– вбудований Git.

Також користувачі можуть змінювати тему, комбінації клавіш, налаштування та встановлювати розширення, що додають додаткову функціональність.

Враховуючи поставлені для розробки задачі та попередній досвід найкращим рушієм для розробки продукту є використання Unity із застосування мови програмування C#. Розробка буде проводитись у середовищі програмування Visual Studio Code. Для розробки спрайтів буде використана програма Adobe Illustrator. Для встроювання реклами буде використаний плагін Google AdMob.

Visual Studio Code – середа розробки програмних засобів, таких як: програмних застосунків, веб додатків, API та мобільних додатків. Використовує платформи розробки компанії Microsoft, такі як Microsoft Silverlight, Windows

Forms, WPF, Магазин Windows та Windows API. Включає в себе редактор коду, засоби тестування, засоби для створення схем та моделей. Підтримує 36 мов програмування в тому числі і C#.

Adobe Illustrator – це програмний додаток для створення малюнків та ілюстрацій. Illustrator був випущений у 1987 році, і він продовжує регулярно оновлюватися, і тепер включений до складу Adobe Creative Cloud [28]. Illustrator широко використовується графічними дизайнерами, веб-дизайнерами, художниками-візуалістами та професійними ілюстраторами у всьому світі для створення високоякісних ілюстрацій. Illustrator включає багато складних інструментів малювання, які можуть скоротити час, необхідний для створення ілюстрацій.

Google AdMob – це плагін Admob SDK для Unity3d, завдяки якому JavaScript та C# розробники можуть легко додати рекламу Google в свій мобільний додаток на Unity3d. Плагін підтримує IOS та Android, підтримує інтерстицію та банер AdMob.

3 ПРОЕКТУВАННЯ ГІПЕР-КАЗУАЛЬНОЇ ГРИ

3.1 Структурно-функціональне моделювання ігрового процесу у грі із механікою головоломка

Функціональна модель IDEF0 розроблена для відображення функціонування системи, управління системою та функціонального потоку процесів життєвого циклу. Діаграма IDEF0 здатна графічно представити широкий спектр ділових, виробничих та інших активностей підприємства до будь-якого рівня деталізації. Вона забезпечує строгий і точний опис процесу виробництва і сприяє послідовності використання і інтерпретації [29].

Для створення структурно-функціональної моделі був використаний програмний додаток Erwin Process Modeler.

Розглянемо процес проходження гри «Laser Beam» з використанням ігрової механіки типу головоломка. Для відображення, як бізнес-процес буде структурований, було використано модель робіт TO-BE (Як буде). Дана модель буде використана для встановлення правильної «точки нуль», необхідної для розробки ігрового мобільного додатку. А саме для розуміння процесу, який відбувається під час проходження гри з механікою типу головоломка.

Контекстна діаграма являє собою основою всієї структури діаграм в ході структурно-функціонального моделювання і представляє собою найбільш загальний опис досліджуваного процесу та його взаємодії з навколишнім середовищем.

Контекстну діаграму процесу проходження гри «Laser Beam» з механікою типу головоломка представлено на рис. 3.1.

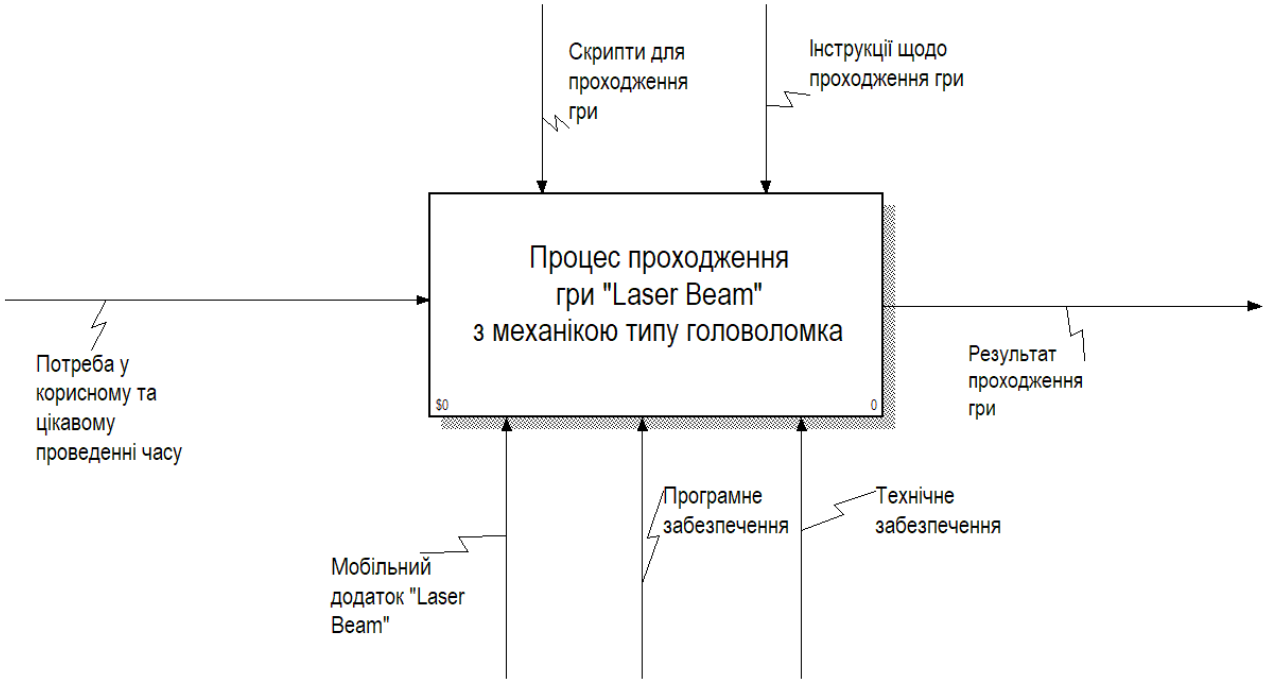

Рисунок 3.1 – Контекстна діаграма

Контекстна діаграма складається лише з одного основного блоку, який чітко описує досліджуваний процес або систему що моделюється, та стрілок або дуг. Вхідна стрілка представляє собою вхідні дані для перетворення системою. Вихідна стрілка представляє собою фінальний результат після закінчення досліджуваного процесу або системи. Стрілки що входять у блок зверху відображають інструменти, які використовуються для перетворення вхідних даних. Стрілки що входять знизу відображають механізми, які сприяють перетворенню вхідних даних [30].

Вхідною стрілкою до функції «Процес проходження гри «Laser Beam» з механікою типу головоломка» є «Потреба у корисному та цікавому проведенні часу»; вихідною – «Результат проходження гри»; стрілками контролю – «Скрипти для проходження гри» та «Інструкції щодо проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», «Програмне забезпечення», та «Технічне забезпечення»

Наступним кроком після опису системи в цілому здійснюється її розбиття на фрагменти. Процес розбиття називається декомпозицією діаграми на функціональні блоки, які описують кожен фрагмент та взаємодію фрагментів між собою і називаються діаграмами декомпозиції.

Відповідно до [31] діаграма декомпозиції складається з блоків діяльності, що відображають одну чи кілька функцій, та стрілок, що являють собою ресурси, потрібні для забезпечення діяльності. Діяльність представляє собою процес, функцію або задачу, що виконується протягом певного інтервалу часу та результатом якої є досягнення мети проекту. Блок діяльності зображується прямокутником, у середині якого зазначається назва функції та номер блока [32]. Блоки на діаграмі розташовують у порядку зменшення важливості з позицій автора діаграми. Найбільш важливий блок розташовують у верхньому лівому, а найменш – в правому нижньому куті. Таким чином, структура діаграми показує, які функції мають вплив на інші [33]. Всі блоки нумеруються в порядку впливу.

На першому рівні декомпозиції моделі головна функція деталізується на наступні блоки представлені на рис. 3.2.:

− авторизація гравця;

− вибір рівня гри;

− проходження ігрового рівня;

− отримання результату проходження.

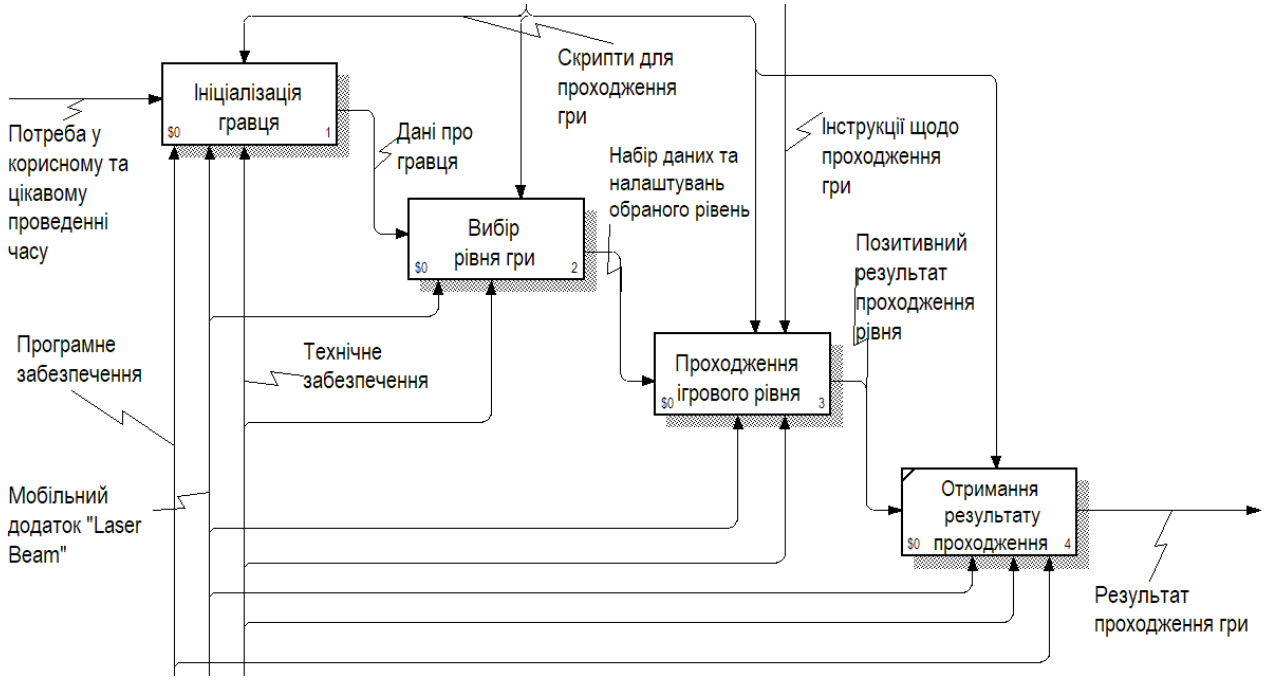

Рисунок 3.2 - Перший рівень декомпозиції IDEF0

Вхідними стрілками до діяльності «Ініціалізація гравця» є «Потреба у корисному та цікавому проведенні часу»; вихідною – «Дані про гравця»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», «Програмне забезпечення», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Вибір рівня гри» є «Дані про гравця»; вихідною – «Набір даних та налаштувань обраного рівня»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Проходження ігрового рівня» є «Набір даних та налаштувань обраного рівня»; вихідною – «Позитивний результат проходження рівня»; стрілками контролю – «Скрипти для проходження гри» та «Інструкції щодо проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Отримання результату проходження» є «Позитивний результат проходження рівня»; вихідною – «Результат проходження гри»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», «Програмне забезпечення», та «Технічне забезпечення».

Всього в розробленій структурно-функціональній моделі два рівні декомпозиції. Діаграму декомпозиції процесу «Вибір рівня гри», що відноситься до другого рівня, зображено на рис. 3.3.

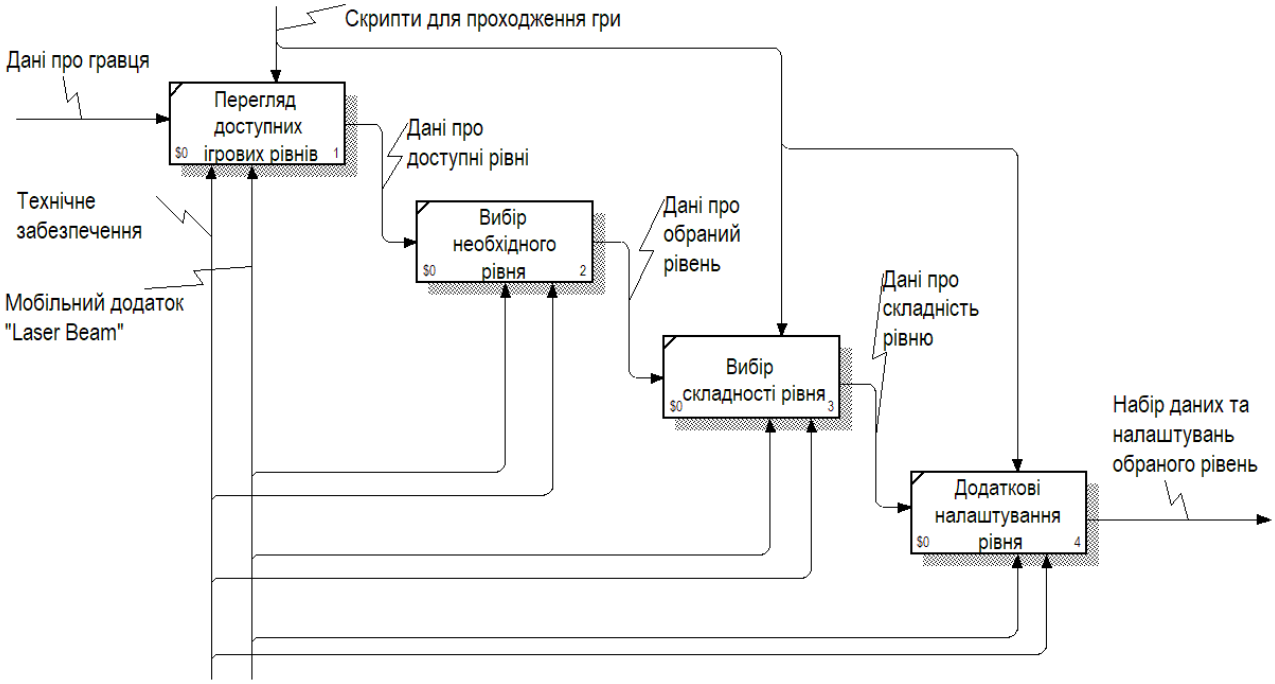

Рисунок 3.3 - Діаграма декомпозиції IDEF0 «Вибір рівня гри»

Вхідною стрілкою до діяльності «Перегляд доступних ігрових рівнів» є «Дані про гравця»; вихідною – «Дані про доступні рівні»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Вибір необхідного рівня» є «Дані про доступні рівні»; вихідною – «Дані про обраний рівень»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Вибір складності рівня» є «Дані про обраний рівень»; вихідною – «Дані про складність рівню»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Додаткові налаштування рівня» є «Дані про складність рівню»; вихідною – «Набір даних та налаштувань обраного рівня»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Діаграму декомпозиції процесу «Проходження ігрового рівня», що також відноситься до другого рівня, зображено на рис. 3.4.

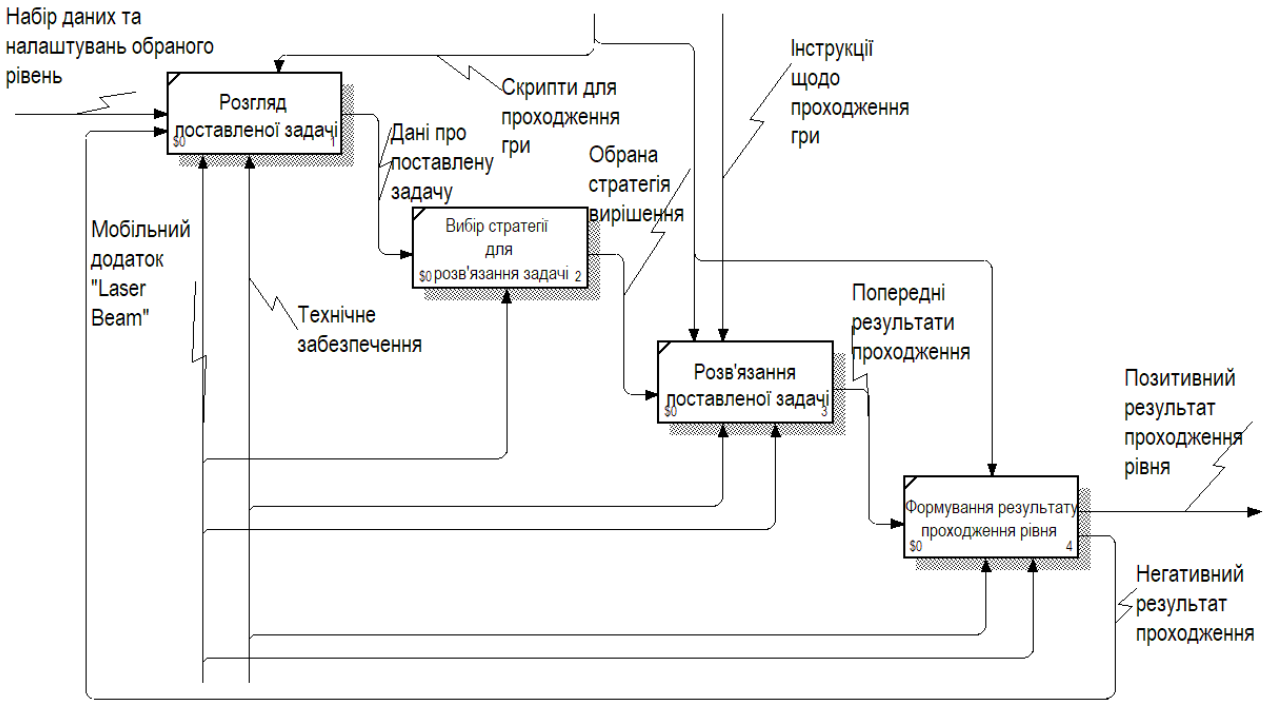

Рисунок 3.4 - Діаграма декомпозиції IDEF0 «Проходження ігрового рівня»

Вхідною стрілкою до діяльності «Розгляд поставленої задачі» є «Набір даних та налаштувань обраного рівня» та стрілка зворотного зв’язку «Негативний результат проходження» ; вихідною – «Дані про поставлену задачу»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Вибір стратегії для розв’язання задачі» є «Дані про поставлену задачу»; вихідною – «Обрана стратегія вирішення»; стрілкою механізмів – «Мобільний додаток «Laser Beam»».

Вхідною стрілкою до діяльності «Розв’язання поставленої задачі» є «Обрана стратегія вирішення»; вихідною – «Попередні результати проходження»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

Вхідною стрілкою до діяльності «Формування результату проходження» є «Попередні результати проходження»; вихідними – «Позитивний результат проходження рівня», або стрілка зворотного зв’язку «Негативний результат проходження»; стрілкою контролю – «Скрипти для проходження гри»; стрілками механізмів – «Мобільний додаток «Laser Beam»», та «Технічне забезпечення».

3.2 Створення діаграми варіантів використання

Для глибокого розуміння функціонування ігрового додатку необхідно використати модель варіантів використання, яка складається із опису наявних акторів, варіантів використання додатку, та взаємодії між ними у формі Use Case-діаграми (діаграми прецедентів) [34].

Найпоширенішим інструментом для документування проекту сьогодення є мова UML, а саме Unified Modelling Language. Дана мова включає в себе велику кількість діаграм [35], але нас цікавить саме діаграма прецедентів (Use Case).

Діаграма прецедентів повинна бути простою і містити лише невелику кількість варіантів використання. Якщо діаграма містить більше 20 фігур, то це є прикладом неправильного її використання [36].

Метою діаграми використання є відображення динамічного аспекту системи. Однак це визначення є занадто загальним, щоб описати мету, оскільки інші чотири діаграми (діяльність, послідовність, співпраця та діаграма стану) також мають те саме призначення [37].

Діаграми варіантів використання використовуються для збору вимог системи, включаючи внутрішні та зовнішні впливи. Ці вимоги в основному є вимогами розробки. Отже, коли система аналізується для збору її функціональних можливостей, виділяють прецеденти та визначаються актори [38].

Для побудови діаграми необхідно визначити діючих акторів та варіанти використання.

Отже, у якості основного актора додатку був виділений лише гравець – актор, що використовує мобільний ігровий додаток для проходження ігрових рівнів без можливості їх редагування.

Далі визначені основні прецеденти проекту:

– вибір ігрового рівня – вибір доступного ігрового рівня у меню гри (для гравця та розробника);

– авторизація гравця – авторизація гравця у системі для отримання доступних йому рівнів та його результатів проходження;

– проходження ігрового рівня – виконання поставленого перед гравцем завдання (для гравця та розробника);

– перегляд результату проходження – перегляд результату гри (кількість балів за проходження) (для гравця та розробника).

Готова діаграма прецедентів, що представляє собою зв'язки акторів з варіантами використання представлена на рис. 3.5.

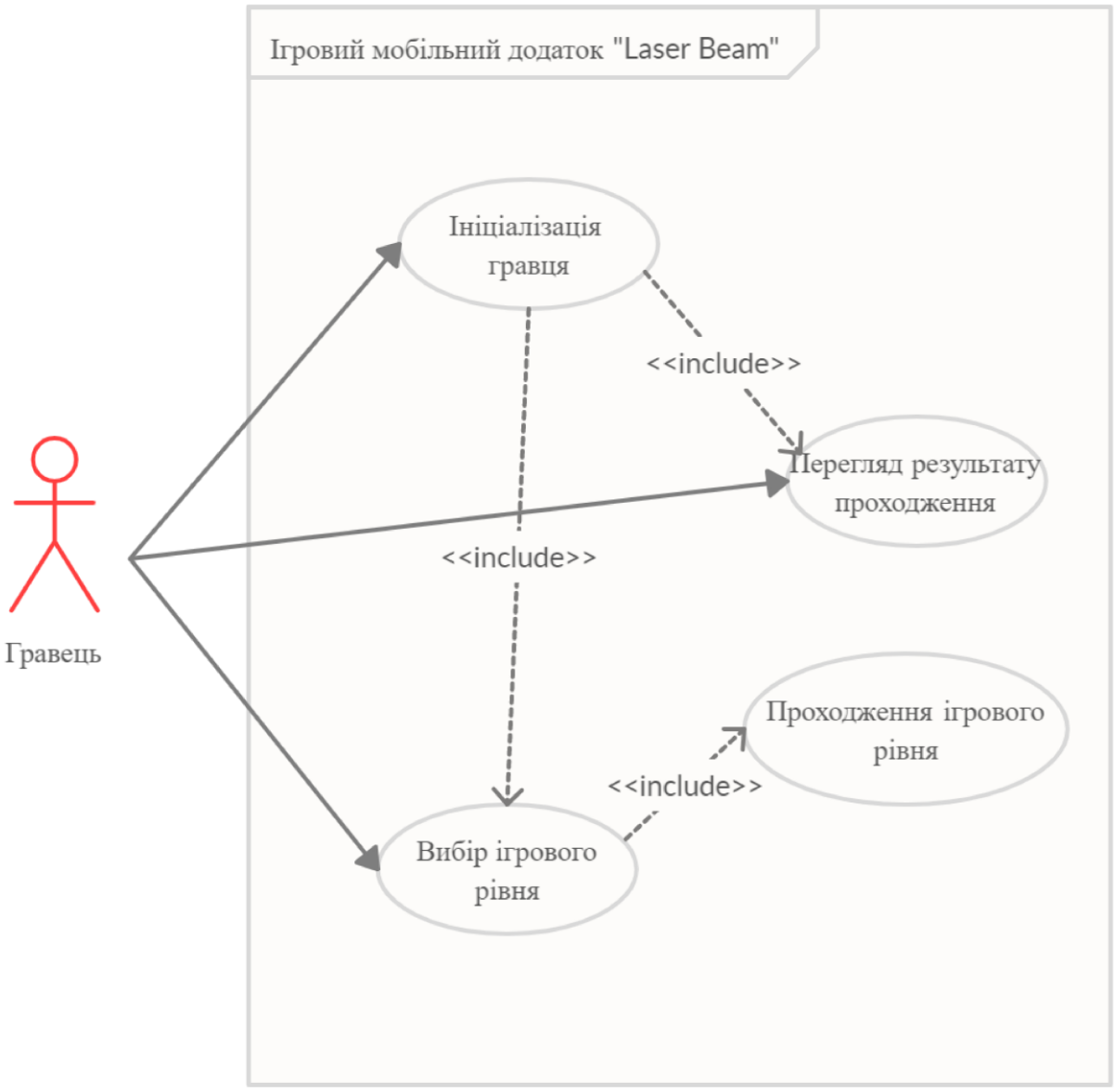

Рисунок 3.5 – Діаграма варіантів використання ігрового мобільного додатку

4 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

4.1 Розробка прототипу ігрового додатку

Не всі розробники розуміють, що необхідним і важливим етапом розробки є створення прототипу гри. Ігровий прототип потрібен для тесту механіки і геймплея гри, на основі якого можна формувати всю подальшу роботу і мати хорошу опорну точку для всього проекту.

Основною механікою додатку була обрана механіка головоломки, виходячи з цього треба реалізувати гру у вигляді задач які буде вирішувати гравець. У якості задачі гравцю необхідно правильно налаштувати лазер за допомогою дзеркал, щоб він влучив у ціль.

Спланувавши основну геймплейну механіку можна переходити до її розробки. Під час розробки прототипу необхідно використовувати лиже примітиви. Ніяких ефектів, інтерфейсів, лише геймплей.

У проекті можна виділити дві основні механіки, які необхідно реалізувати у рамках прототипу:

– механіка лазеру, та його відбивання від дзеркал;

– механіка ротації дзеркал для відбивання лазеру.

Для розробки даних механік першим чином необхідно створити проект і виконати налаштування збірки (рис. 4.1), а саме змінити платформу розробки на Android.

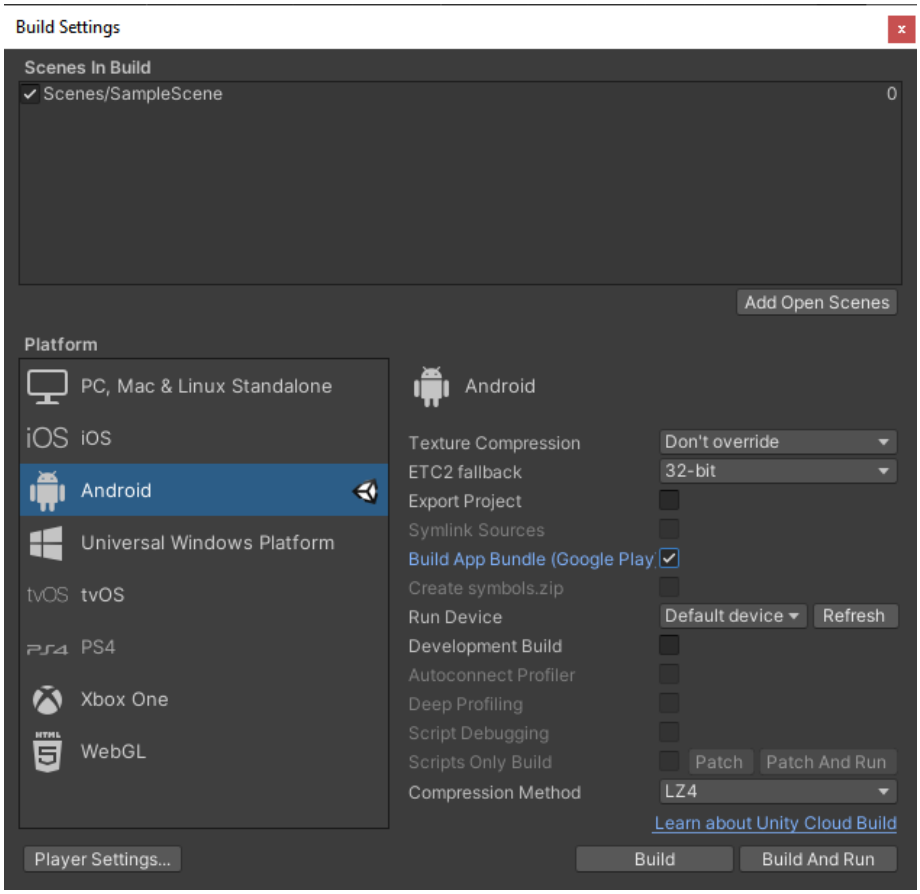

Рисунок 4.1 – Попереднє налаштування збірки проекту

Далі перейдемо до розробки самих механік. У якості лазеру був використаний елемент Line, який включає в себе компонент LineRenderer, який в свою чергу відповідає за відображення лазеру у вікні гри. Додаємо елемент у структуру додатку, та налаштовуємо його (рис. 4.2).

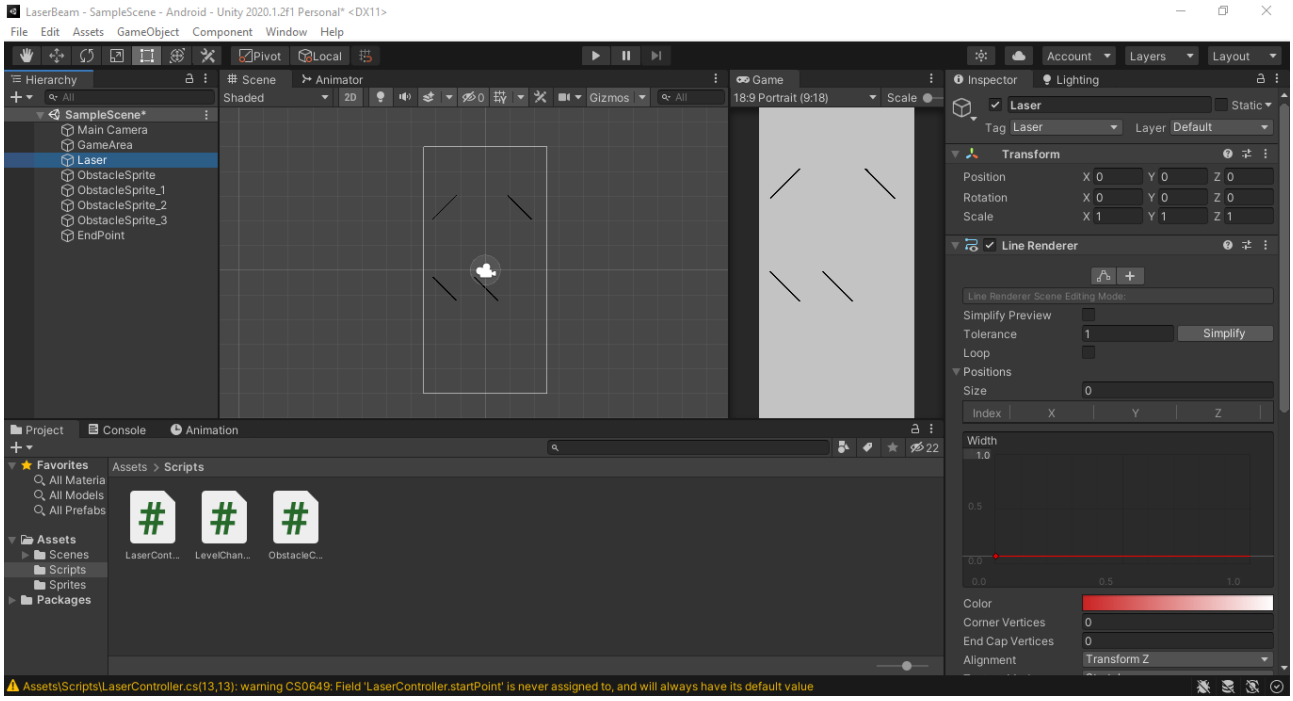

Рисунок 4.2 – Налаштування елементу Line

Наступним кроком додаємо новий компонент у вигляді скрипта до створеного раніше елемента, та почнемо написання коду для відбивання лазеру від примітивів. Для цього після додавання компоненту відкриємо створений скрипт у середовищі розробки та розпочнемо написання коду (рис. 4.3). Для імітування відбивання лазеру був використаний `Physics2D.Raycast`. Він створює промінь проти колайдерів у сцені.

Промінь концептуально схожий на лазерний промінь, який випускається з точки вздовж певного напрямку. Контакт з будь-яким колайдером буде виявлений і про нього буде отримане повідомлення.

Із повним кодом скрипту лазеру можна ознайомитись у додатку Б.

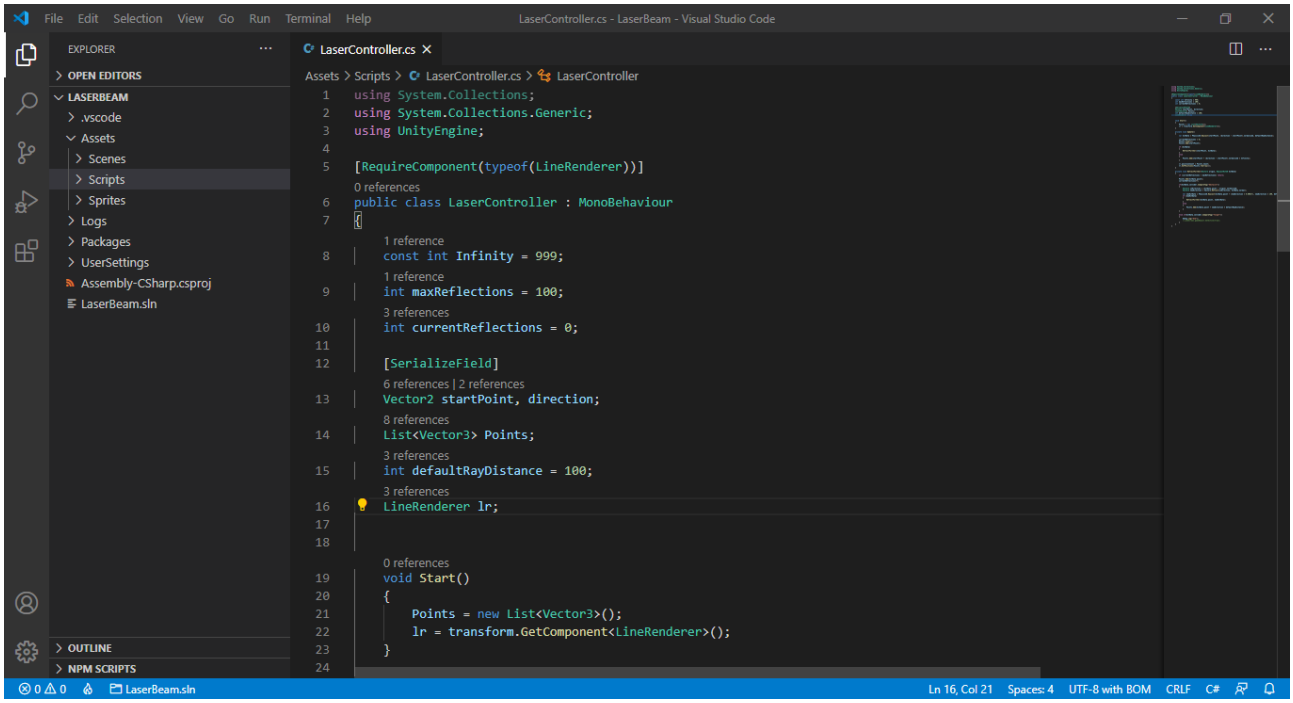

Рисунок 4.3 – Процес створення лазеру

Для тестування лазеру необхідно додати примітив до ігрової області. Для цього була зроблена просто лінія у Adobe Illustrator, у майбутньому вона буде замінена на повноцінний спрайт. Створену лінію необхідно додати у ігрове середовище, та додати компонент EdgeCollider2D (рис. 4.4) для того, щоб лазер міг відбиватися від її поверхні.

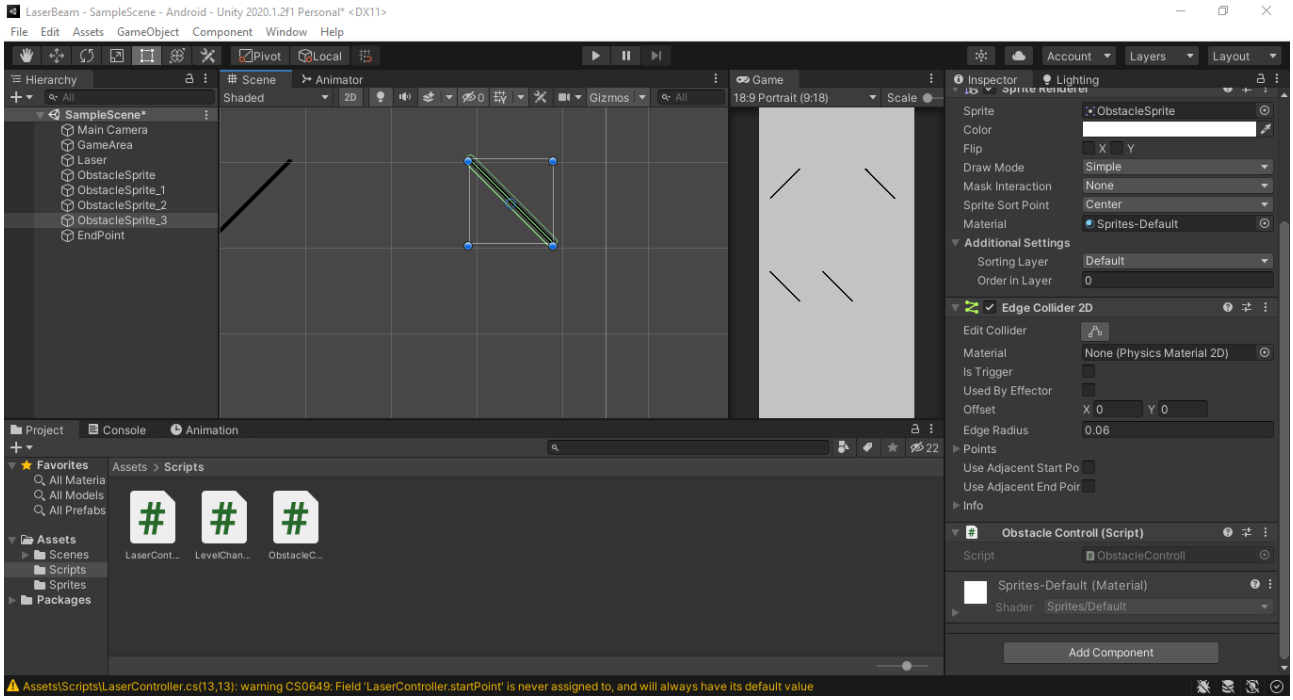

Рисунок 4.4 – Додавання компоненту до ігрового об’єкта

Коли усі примітиви з налаштованими колайдерами були додані на ігрову область, можна перейти до тестування механіки лазеру. Для цього необхідно встановити початкову позицію лазеру та запустити попередній перегляд додатку як показано на рис. 4.5.

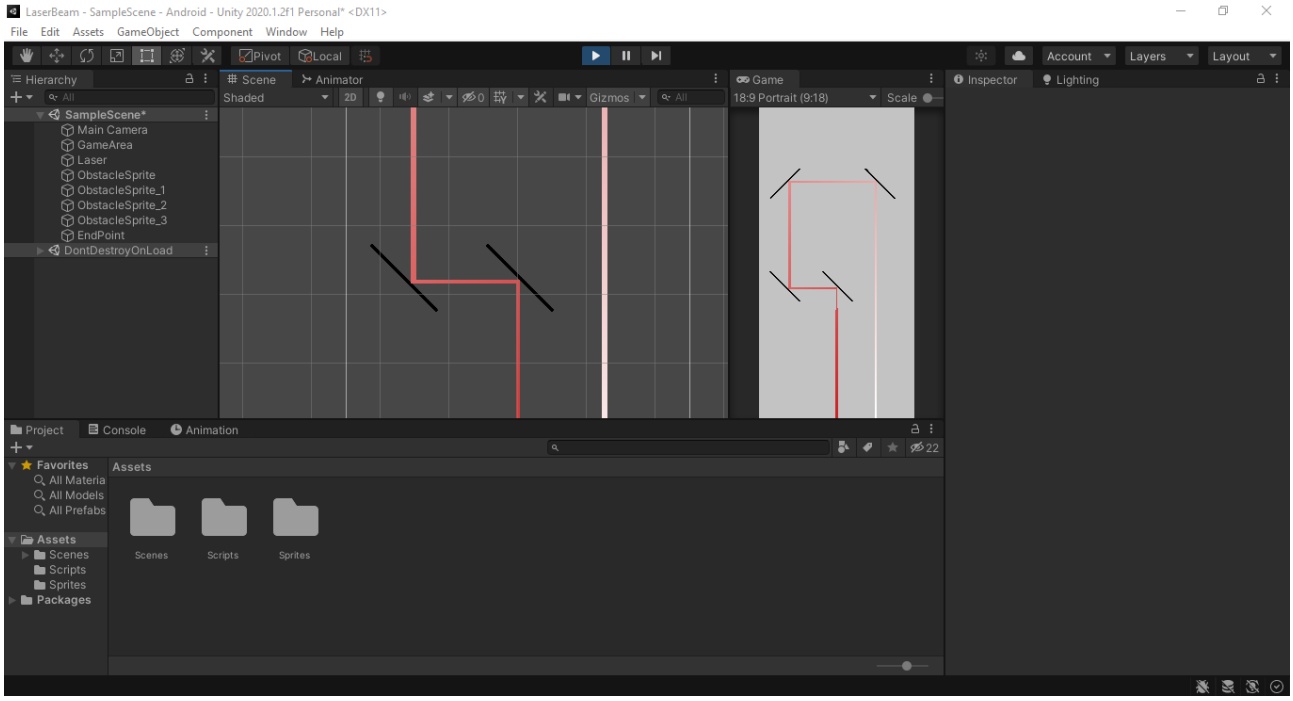

Рисунок 4.5 – Процес тестування механіки відбивання лазеру

Наступним кроком була розробка механіки обертання примітивів для керуванням напрямку лазера. Для цього елемента примітиву був доданий скрипт, та написаний скрипт зміни положення примітиву (рис. 4.6).

Код обертання виглядає доволі просто, але він буде модифікуватися у майбутній розробці. Лістинг скрипта обертання можна знайти у додатку Б.

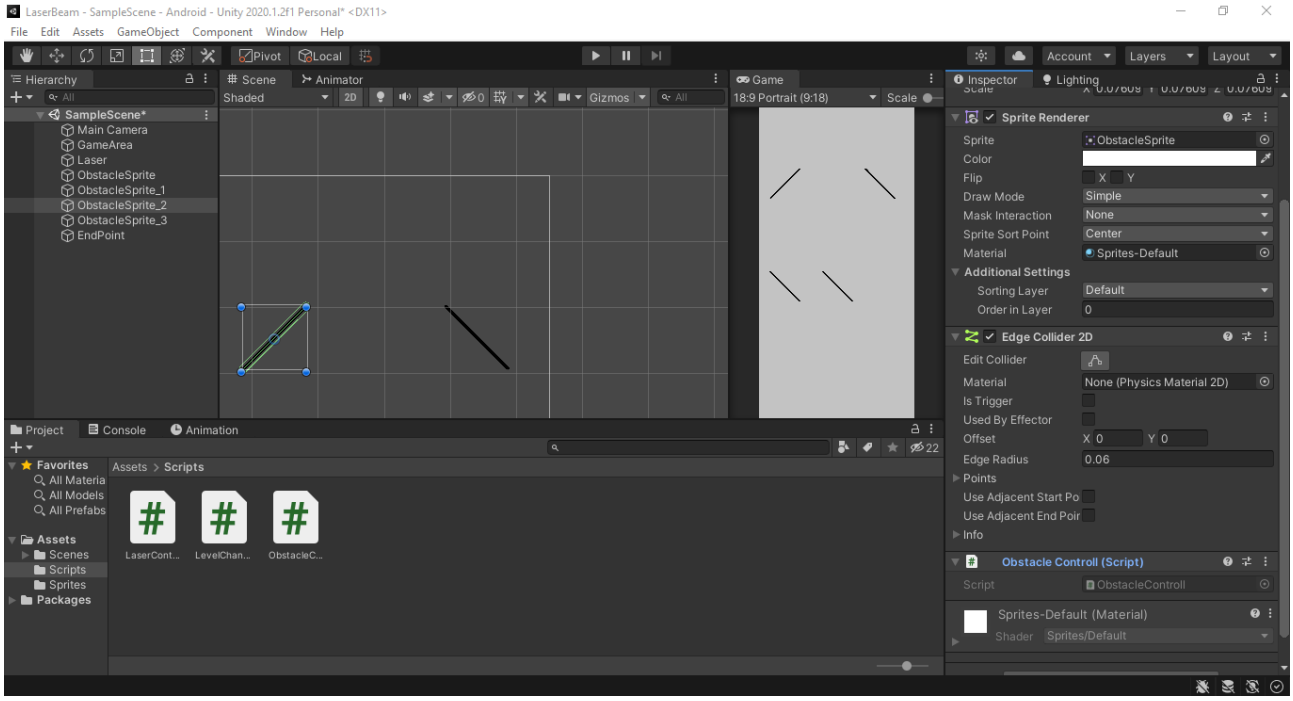

Рисунок 4.6 – Додавання скрипту до елементу

Протестуємо отриманий прототип запустивши попередній перегляд додатку (рис. 4.7).

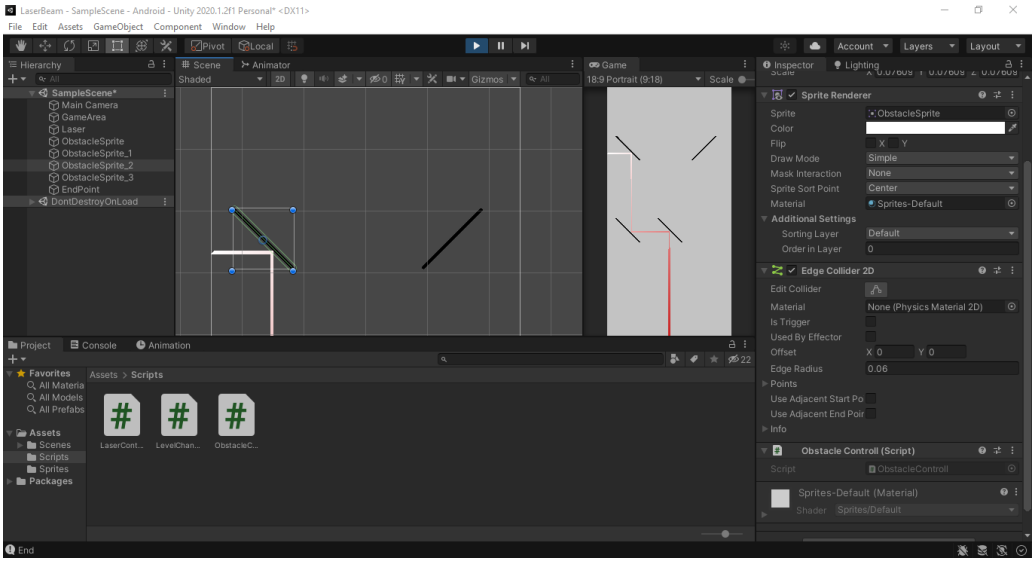

Рисунок 4.7 – Тестування прототипу

Після успішного тестування та розробки основних ігрових механік можна переходити до налаштування менеджеру рівнів.

Налаштування менеджеру рівнів також є важливою частиною розробки прототипу. Для реалізації менеджеру рівнів необхідно модифікувати скрипт лазеру, та додати на ігрову область невидимий колайдер. При перетині колайдеру лазером буде виконаний перехід до наступного рівня. Створення колайдеру зображено на рис. 4.8.

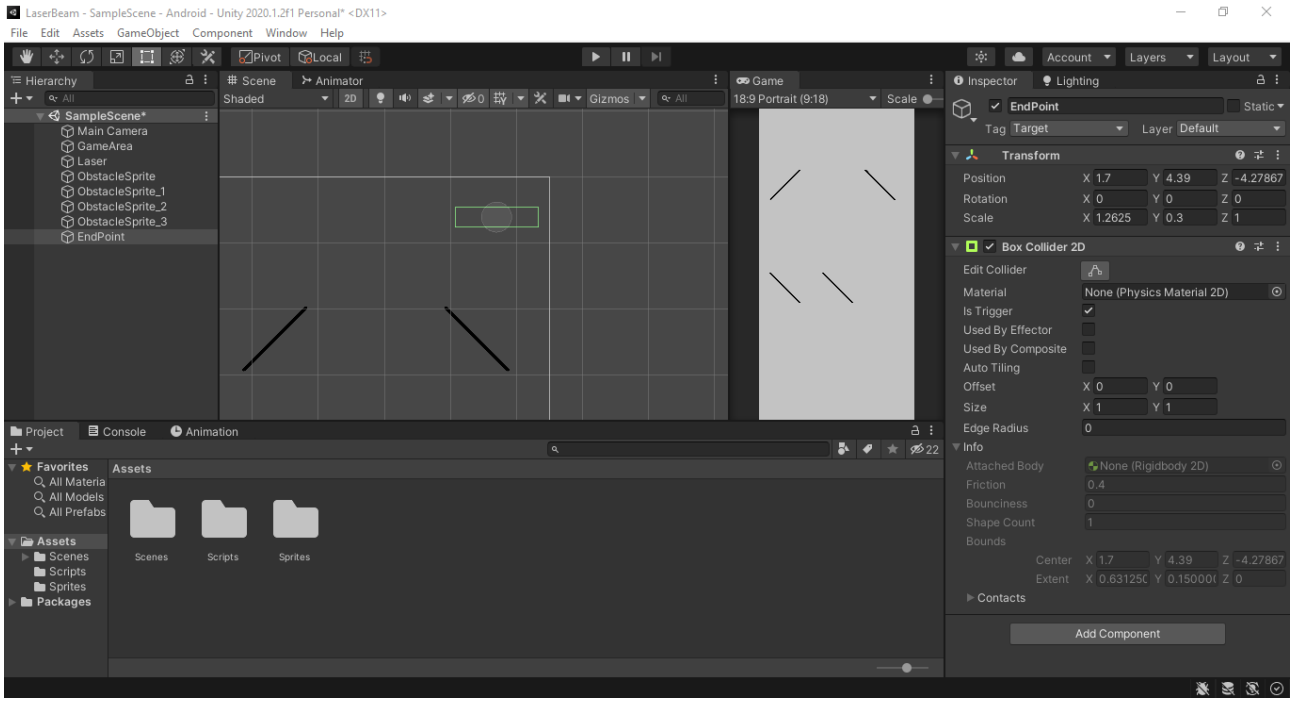

Рисунок 4.8 – Додавання колайдеру на ігрову область

Наступним кроком необхідно додати новий рівень у налаштуваннях збірки (рис. 4.9).

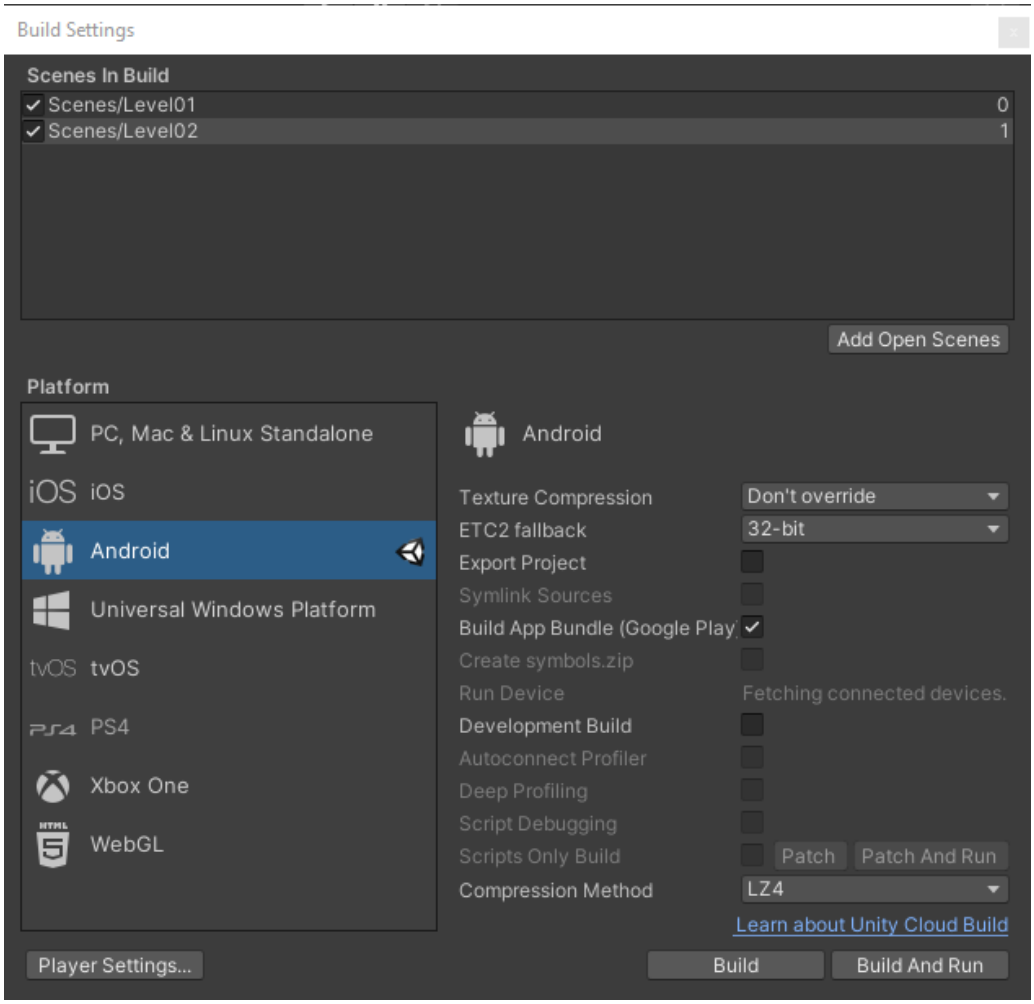

Рисунок 4.9 – Зміна налаштувань збірки

Для того щоб відбувалася подія завантаження наступного рівня необхідно модифікувати скрипт лазеру так, щоб після перетину колайдеру з тегом «Target» лазером рівень змінювався (рис. 4.10)

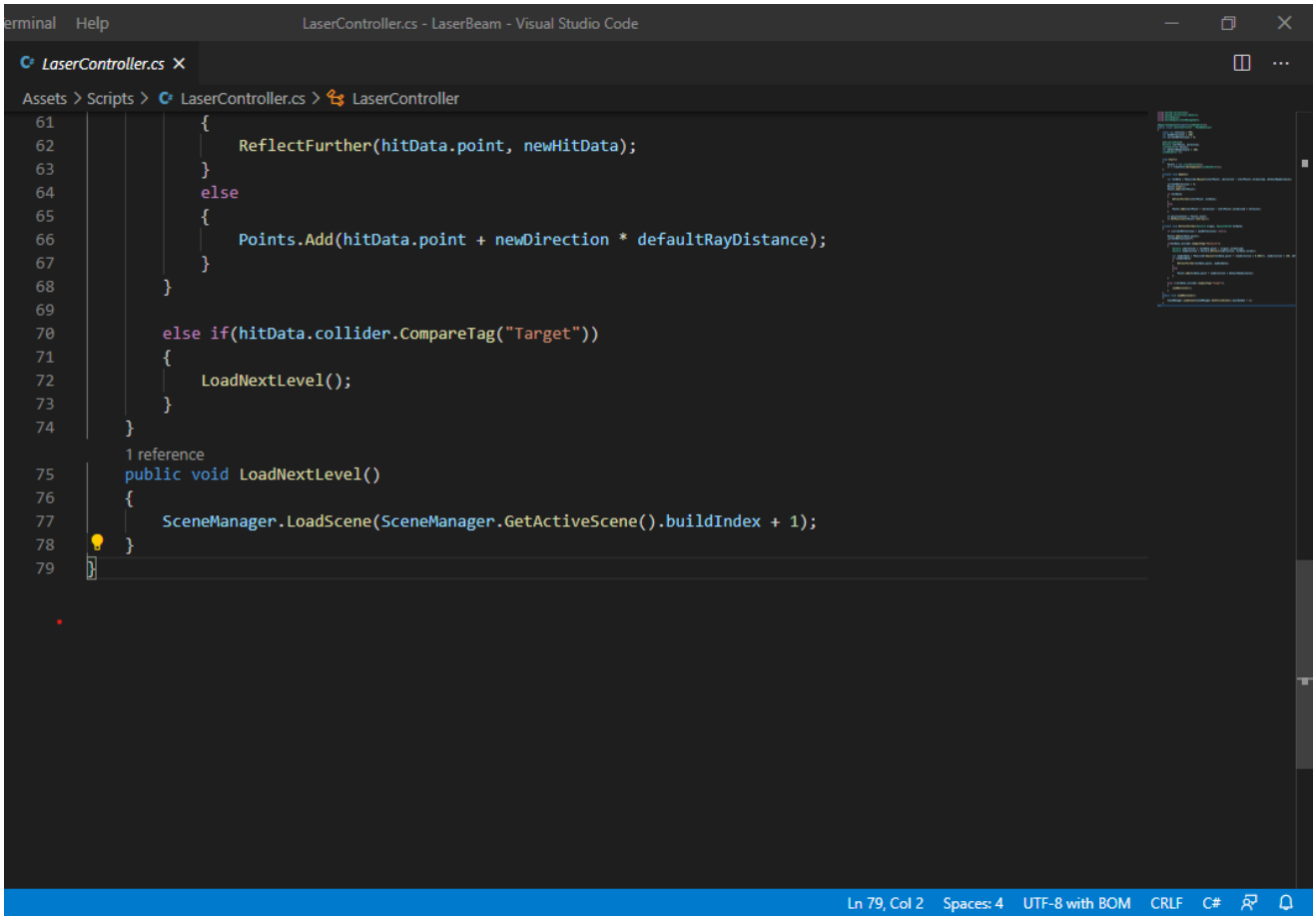

Рисунок 4.10 – Модифікація коду скрипта лазера

4.2 Розробка інтерфейсу додатку

Розробку інтерфейсу слід починати з створення нової сцени, що буде призначена спеціально для головного меню, меню вибору рівня, та меню з результатами проходження ігрових рівнів.

Після створення сцени необхідно додати Canvas на сцену, розмістивши у ньому елемент Panel на якому будуть розміщені кнопки та текст (рис. 4.11).

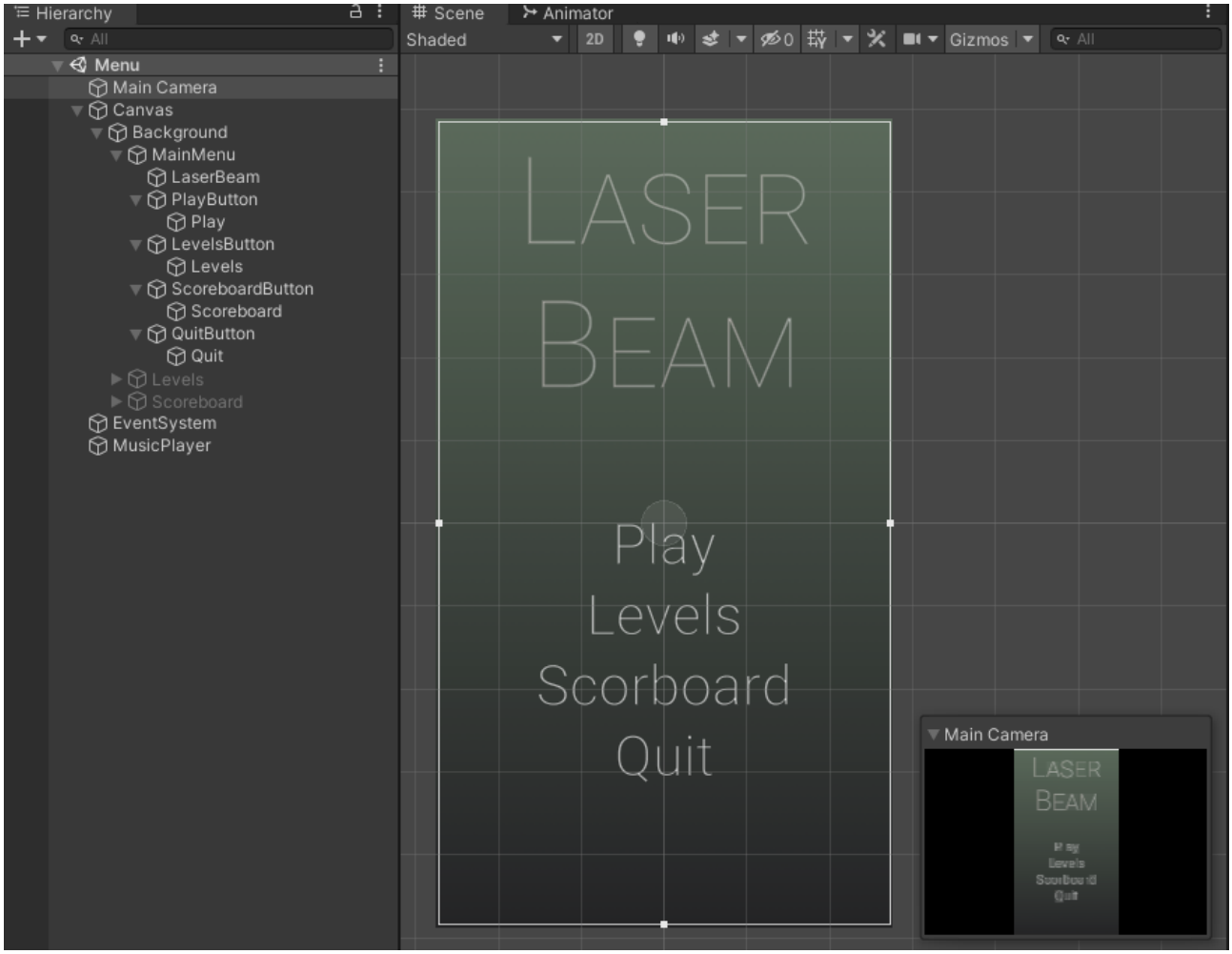

Рисунок 4.11 – Структура головного меню

Для відображення тексту буде використаний елемент TextMeshPro через те що він має велику кількість налаштувань в порівнянні із стандартним елементом Text.

Для створення кнопки також будем використовувати елемент із бібліотеки TextMeshPro (рис. 4.12).

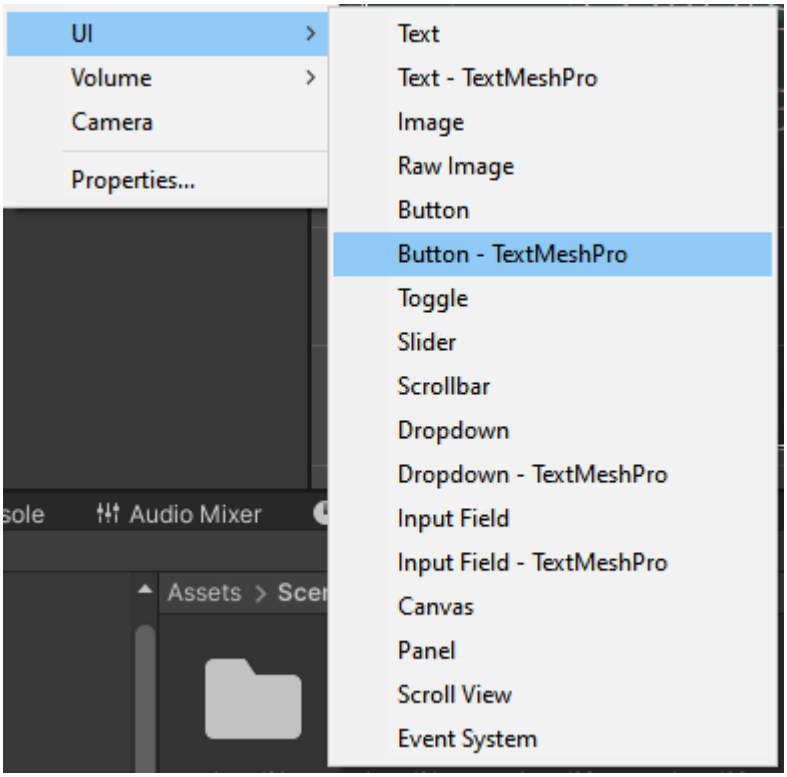

Рисунок 4.12 – Структура головного меню

Щоб використовувати нестандартні шрифти у тексті що створений за допомогою елемента TextMeshPro необхідно створити asset. Для цього необхідно додати необхідний шрифт у проект та використати вбудований Font Asset Creator (рис. 4.13).

У Font Asset Creator необхідно додати потрібний шрифт, виконати настройки та натиснути кнопку Generate Font Atlas. Після цього шрифт можна буде використовувати.

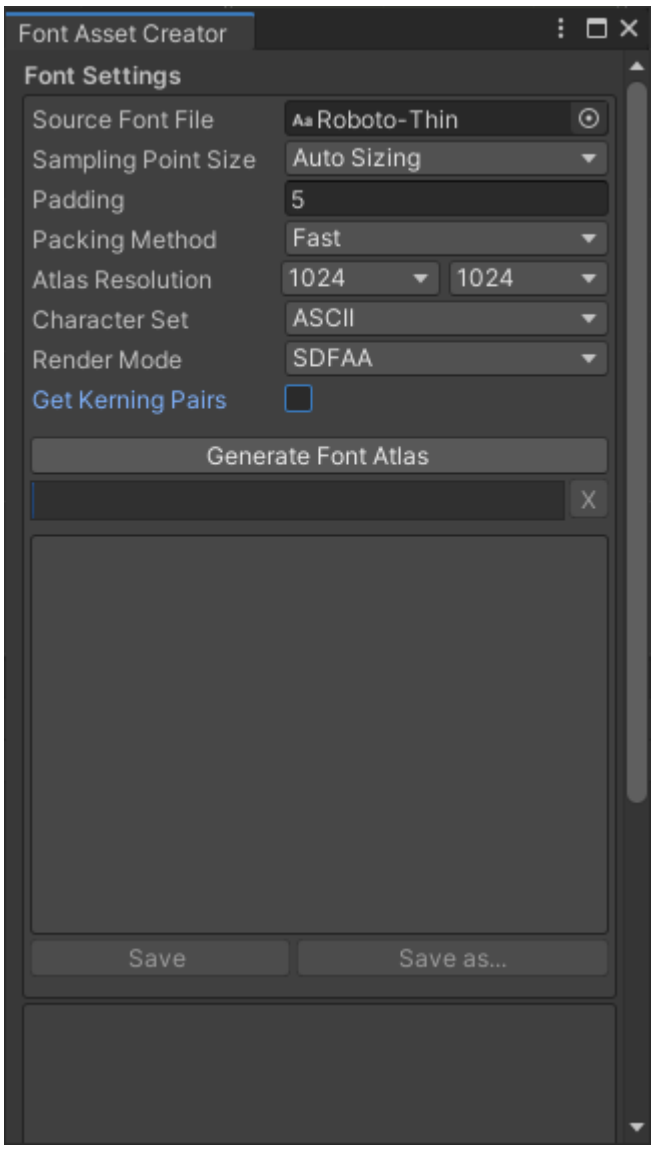

Рисунок 4.13 – Вікно Font Asset Creator

Для розробки головного меню необхідно розмістити елементи на канвасі та розроботи задній фон для додатку. Для створення заднього фону був використаний сайт mdigi.tools (рис. 4.14), після чого елементи меню були розставлені на свої місця та виконана їх прив’язка (рис 4.15).

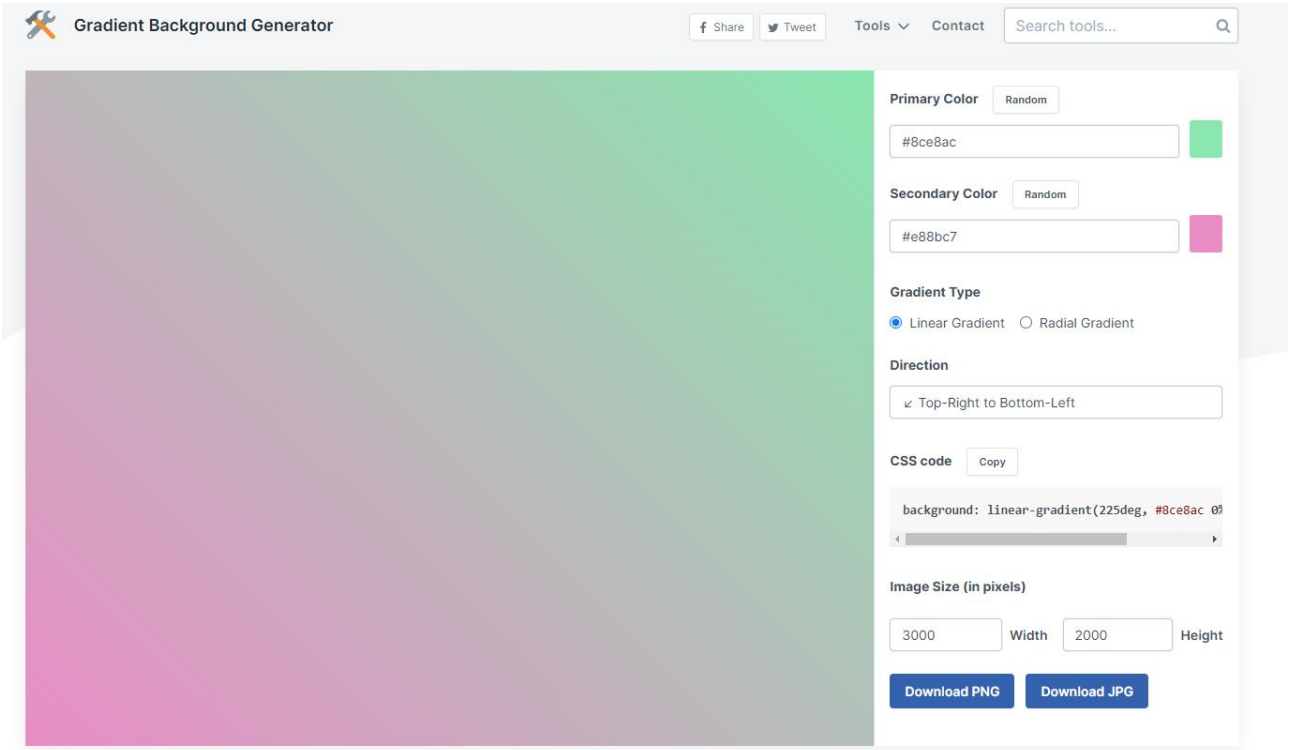

Рисунок 4.14 – Створення градієнту для заднього фону додатку

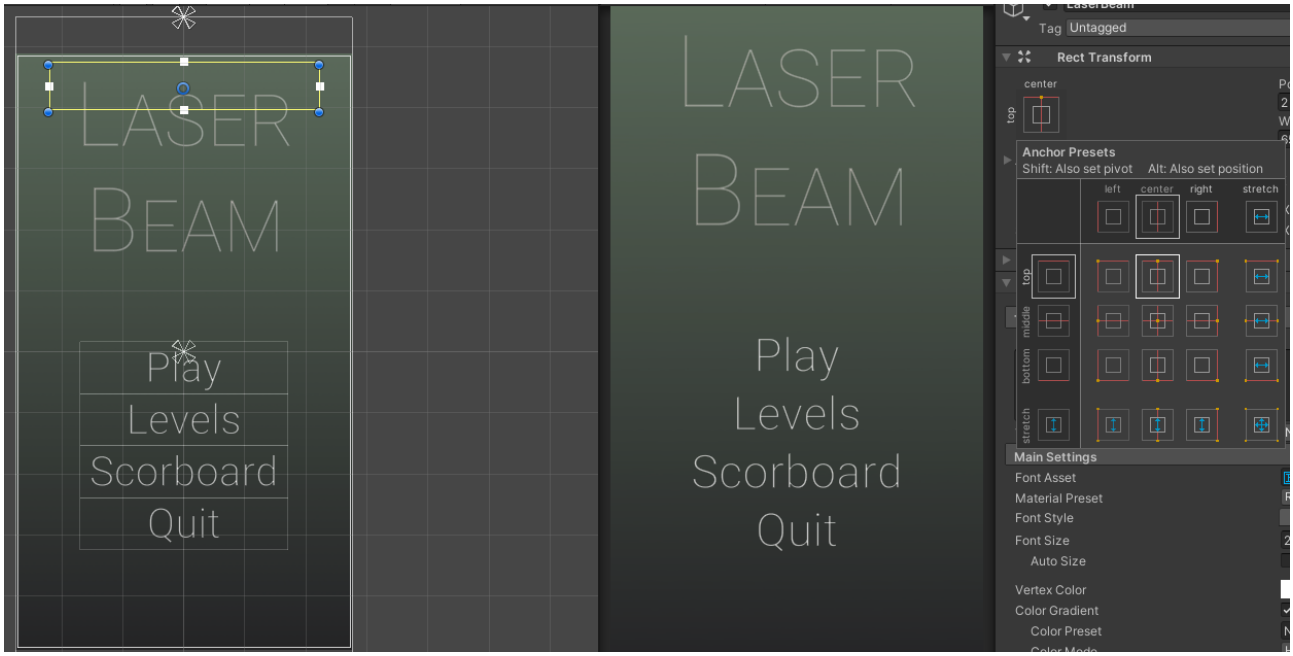

Рисунок 4.15 – Розміщення елементів на канвасі та виконання прив'язок

Після розміщення елементів необхідно налаштувати функціювання кнопок, для цього треба додати скрипт до елемента панель та додати код бажаної функції (рис. 4.16). Повний код скрипту головного меню можна знайти у додатку Б.

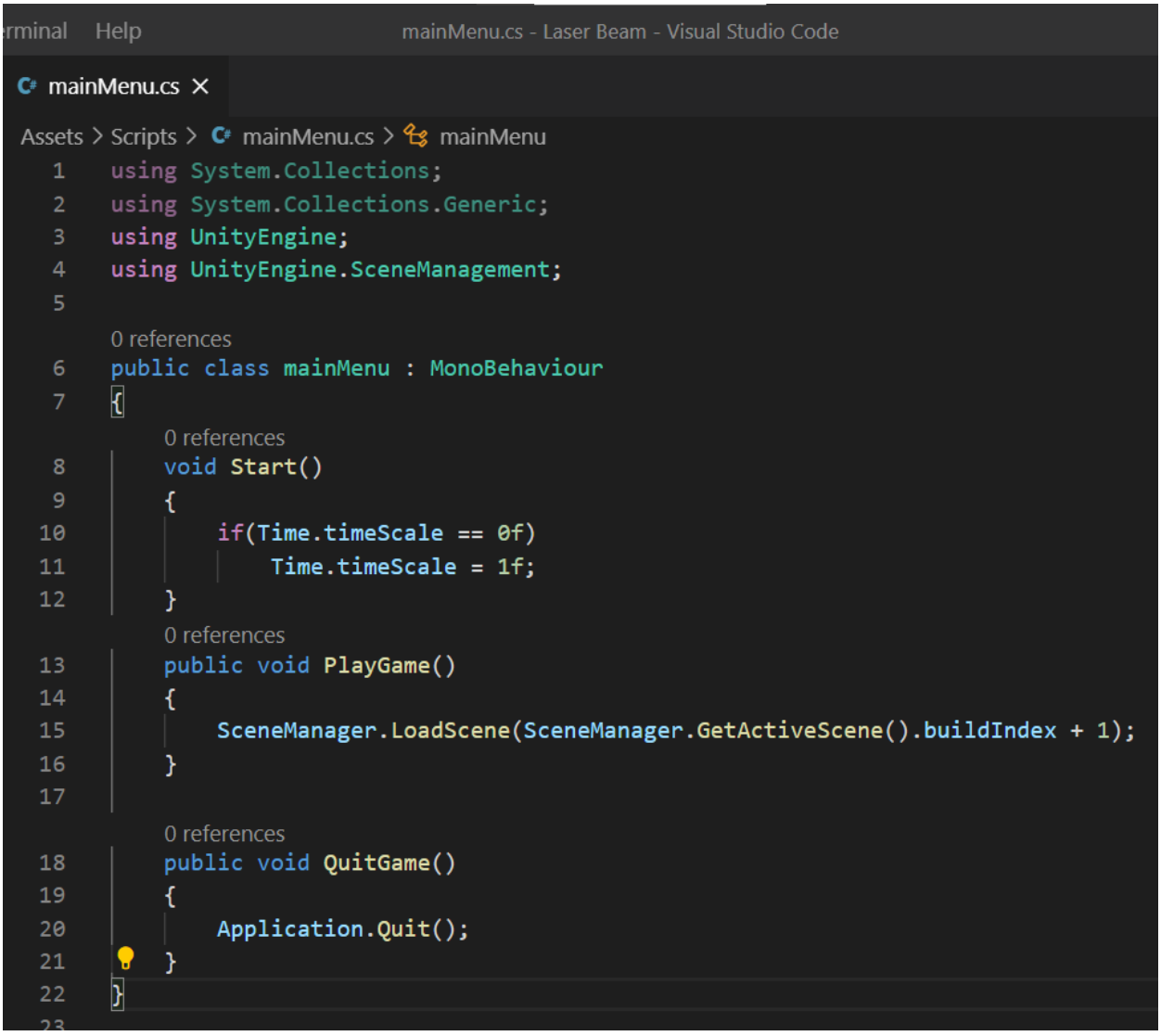

Рисунок 4.16 – Код функцій кнопок у середовищі розробки

Далі необхідно прив’язати написаний код до кнопок. Для цього треба у параметрах елементу додати скрипт, та обрати необхідну функцію (рис. 4.17).

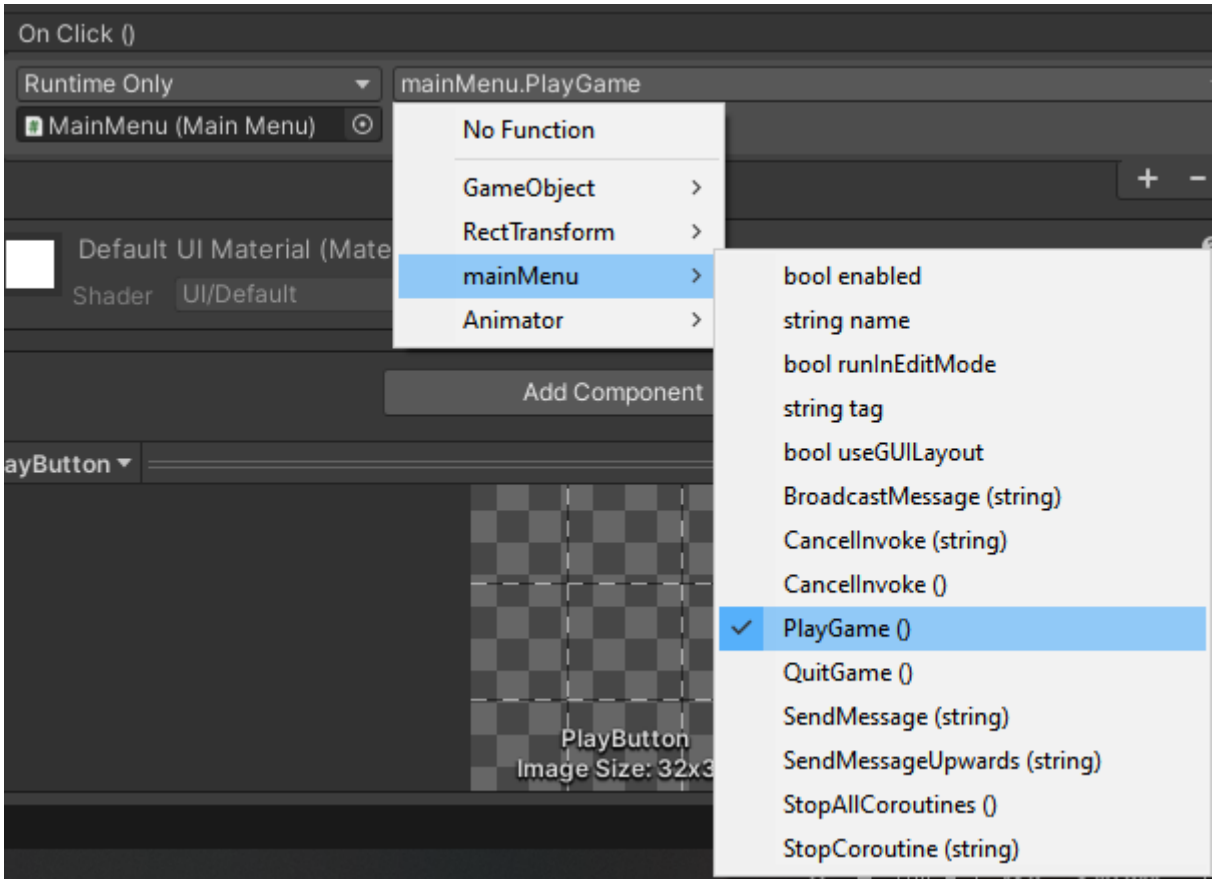

Рисунок 4.17 – Прив'язка функції до кнопки

Після того як головне меню створено, можна переходити до розробки інтерфейсу вибору ігрового рівня. Для цього по аналогії необхідно створити елемент панель у уже існуючому канвасі та додати до нього кнопки які будуть відповідати за вибір рівню та кнопку для повернення у головне меню (рис. 4.18). Для відображення тексту, по аналогії з попереднім меню, був використаний шрифт створений раніше.

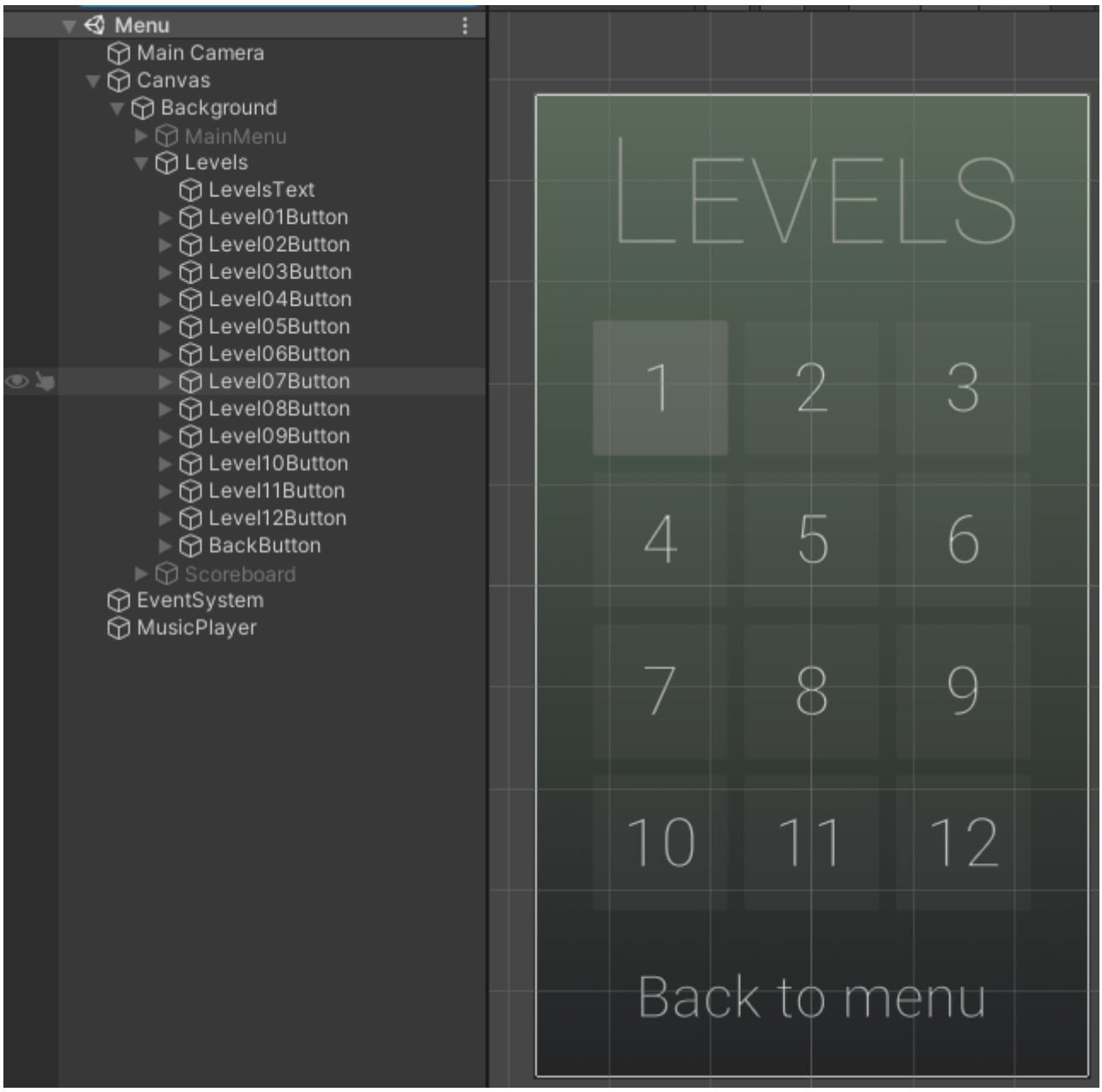

Рисунок 4.18 – Створення інтерфейсу меню вибору рівня

Далі до кнопок треба прив’язати функції, які будуть відкривати обраний рівень. Для цього необхідно створити скрипт менеджеру рівнів (рис. 4.19), та прив’язати функції до кнопок (рис. 4.20). Повний код скрипту меню вибору рівня можна знайти у додатку Б.

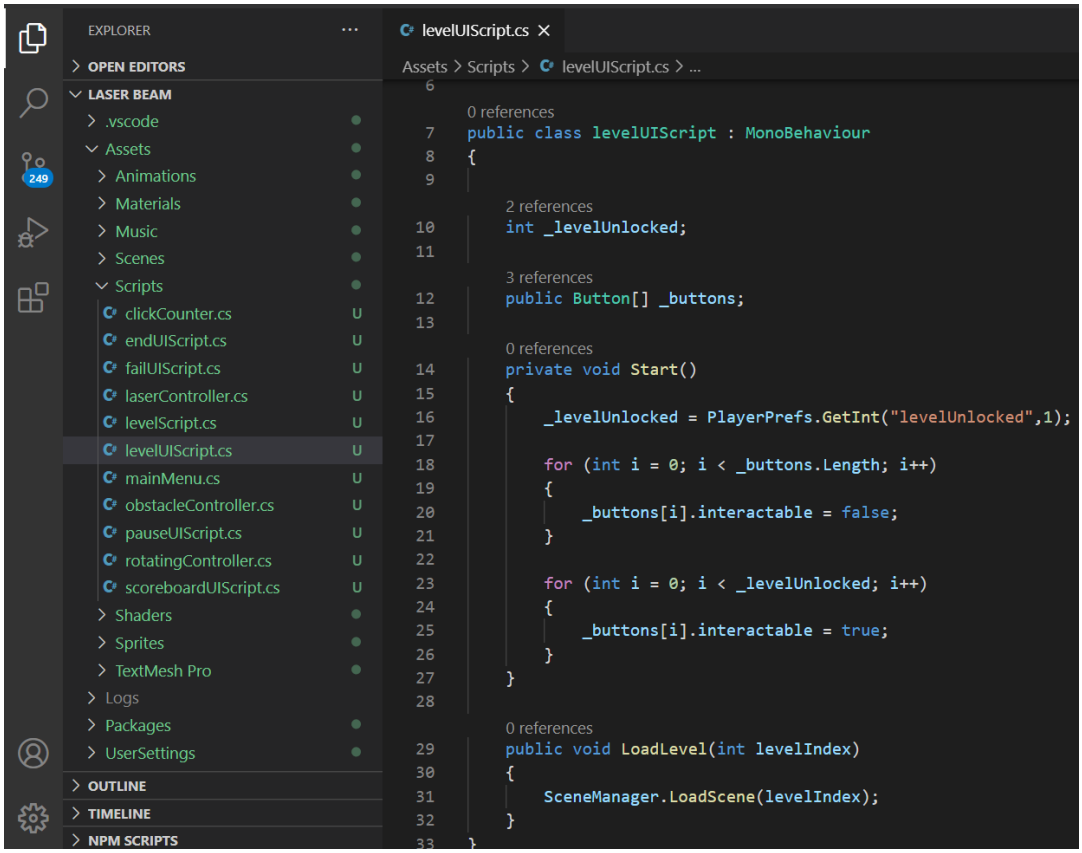

Рисунок 4.19 – Код скрипту меню вибору рівнів у середовищі розробки

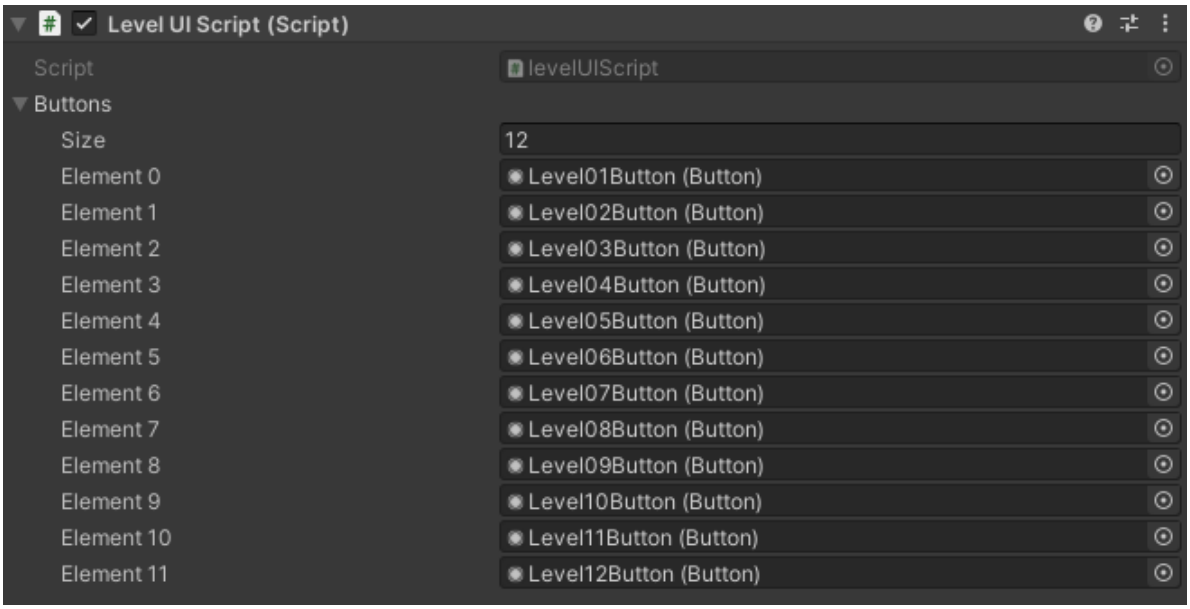

Рисунок 4.20 – Прив'язка кнопок до скрипта

Тепер можна перейти до створення інтерфейсу меню результатів. Для цього був просто доданий текст з назвою рівня та кількість очок за нього і одна кнопка для повернення у головне меню (рис. 4.21). Для стилізації тексту був використаний шрифт створений раніше.

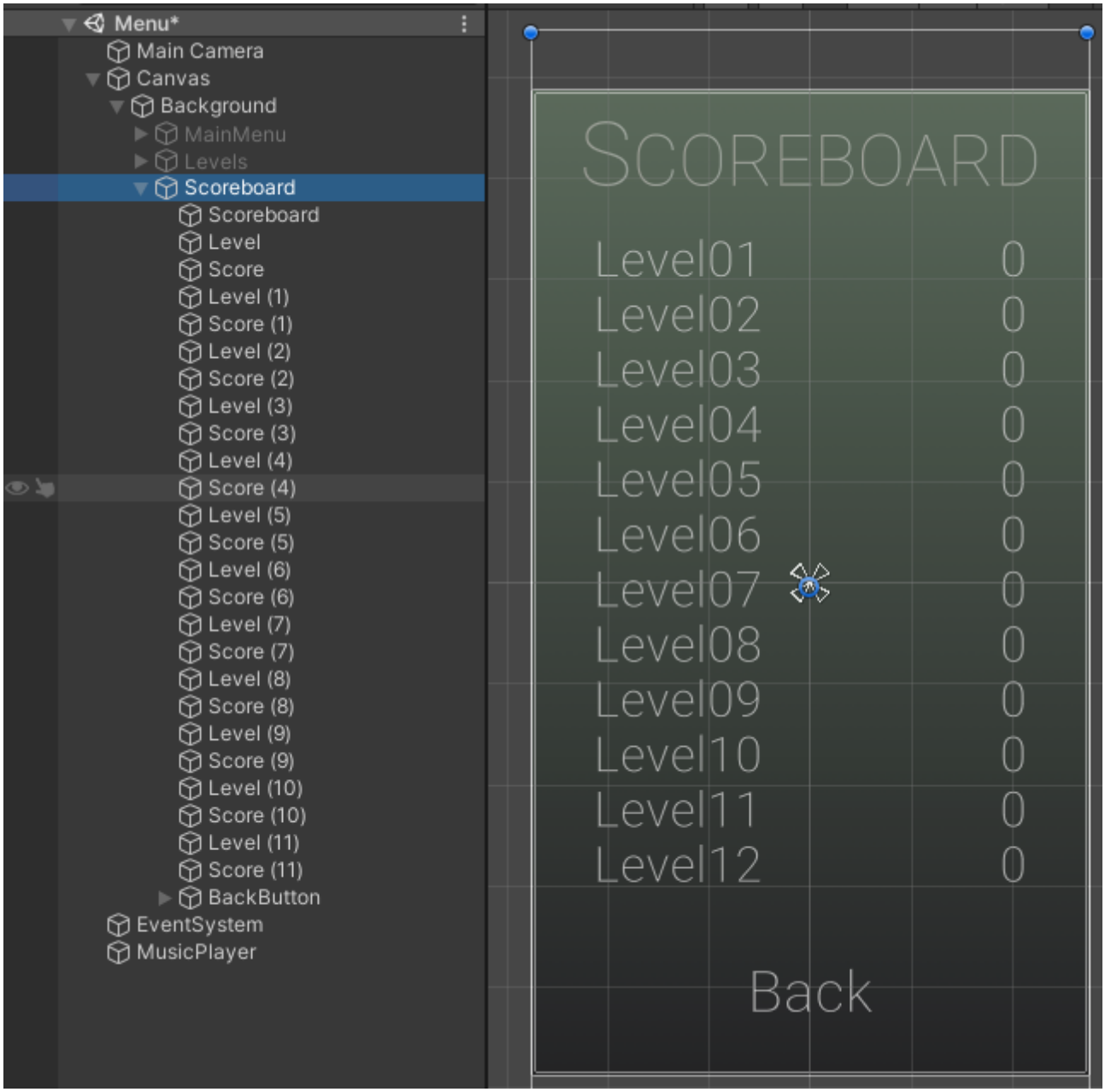

Рисунок 4.21 – Структура меню результатів проходження рівнів

Також, для відображення результатів на екрані необхідно створити скрипт (рис. 4.22). Повний код скрипту меню перегляду результатів проходження можна знайти у додатку Б.

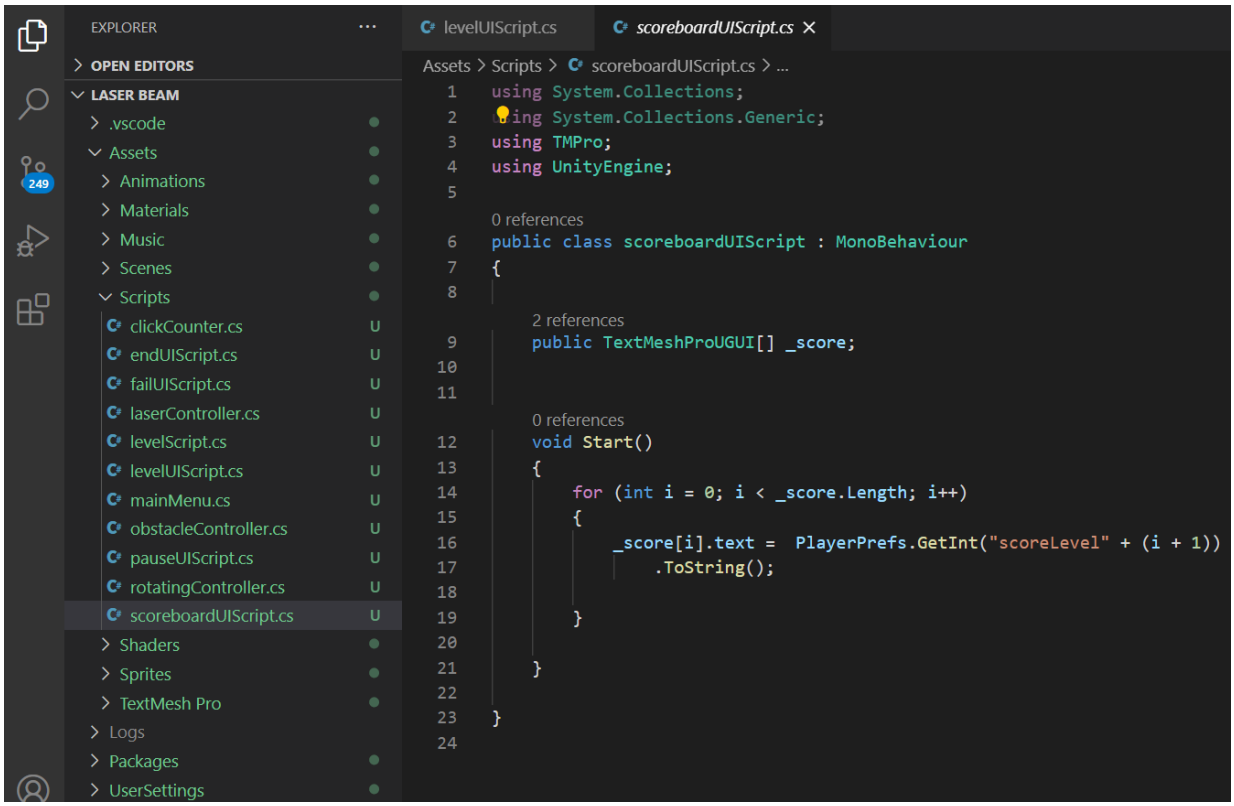

Рисунок 4.22 – Код відображення результатів проходження

Після створення інтерфейсів головного меню можна переходити до розробки ігрових інтерфейсів, а саме: інтерфейсу гри, інтерфейсу вдалого та невдалого проходження рівнів та інтерфейсу паузи.

Для усіх цих інтерфейсів був використаний раніше створений шрифт, та елементи TextMeshPro.

Ігровий інтерфейс включає в себе лише кнопку паузи та текст з кількістю доступних натискань на дзеркала, його структуру показано на рис. 4.23.

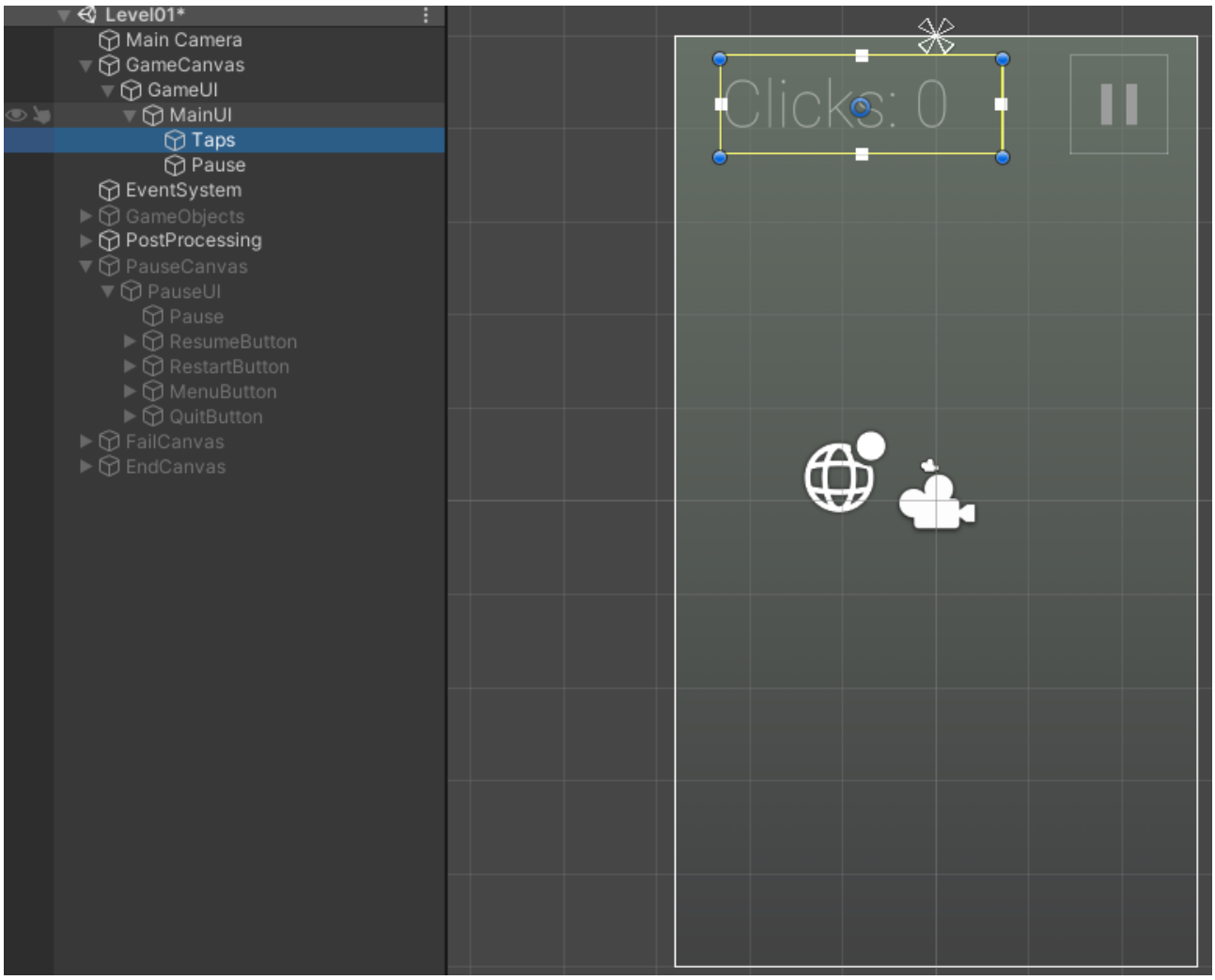

Рисунок 4.23 – Структура ігрового інтерфейсу

Для відображення кількості натискань та функціювання кнопки паузи був створений скрипт (рис. 4.24). Після цього даний скрипт був доданий до кнопки.

Також для налаштування кількості доступних натискань даний параметр був винесений у параметри ігрового об'єкта меню (рис. 4.25).

Створений скрипт також відповідає за виклик інших інтерфейсів, у випадках коли гарець пройшов рівень невдало. Повний код скрипту ігрового інтерфейсу можна знайти у додатку Б.

```
obstacleController.cs   levelScript.cs ×
Assets > Scripts > levelScript.cs > levelScript > Pause()

        0 references
17      private void Start()
18      {
19          if( Time.timeScale == 0f) Time.timeScale = 1f;
20          globalClickController.sharedInstance._clickCounter = _clickCounter;
21          _clickText.text = "Clicks: " + _clickCounter.ToString();
22      }
23
        0 references
24      private void Update()
25      {
26          if (Input.GetMouseButtonDown(0))
27          {
28              _clickText.text = "Clicks: " + globalClickController.sharedInstance._clickCounter.ToString();
29          }
30
31          if(globalClickController.sharedInstance._clickCounter == 0)
32          {
33              _failCanvas.SetActive(true);
34
35              Animator animator = _failUI.GetComponent<Animator>();
36
37              if(animator != null)
38              {
39                  animator.SetBool("failTransition", true);
40              }
41
42              Invoke("TimeStop", 2f);
43          }
44      }
45
        0 references
46      public void Pause()
47      {
48          Animator animator = _pauseUI.GetComponent<Animator>();
49
50          if(animator != null)
51          {
52              animator.SetBool("pauseTransition", true);
```

Рисунок 4.24 – Код скрипту ігрового інтерфейсу

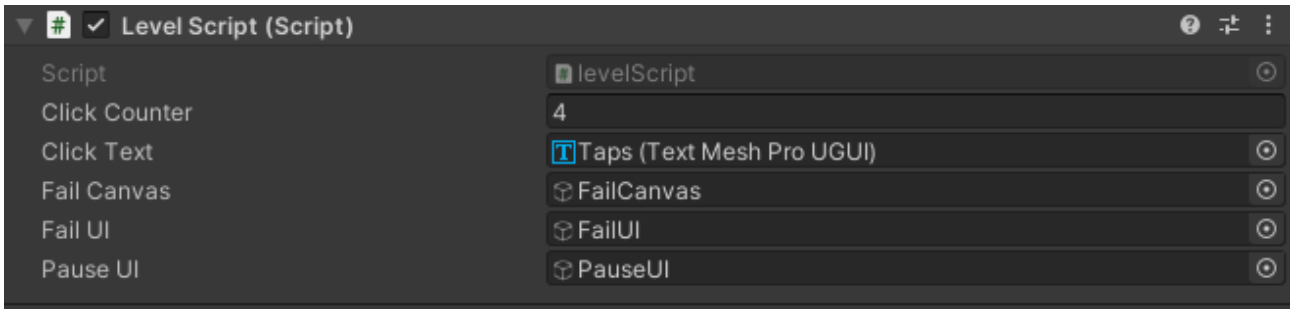

Рисунок 4.25 – Параметри ігрового об’єкту інтерфейсу

Наступний інтерфейс – інтерфейс паузи, який складається лише з його назви та кнопок з функціями (рис. 4.26). Для функціювання кнопок був розроблений скрипт (рис. 4.27), після чого кожна функція була прив’язана до своєї кнопки (рис. 4.28).

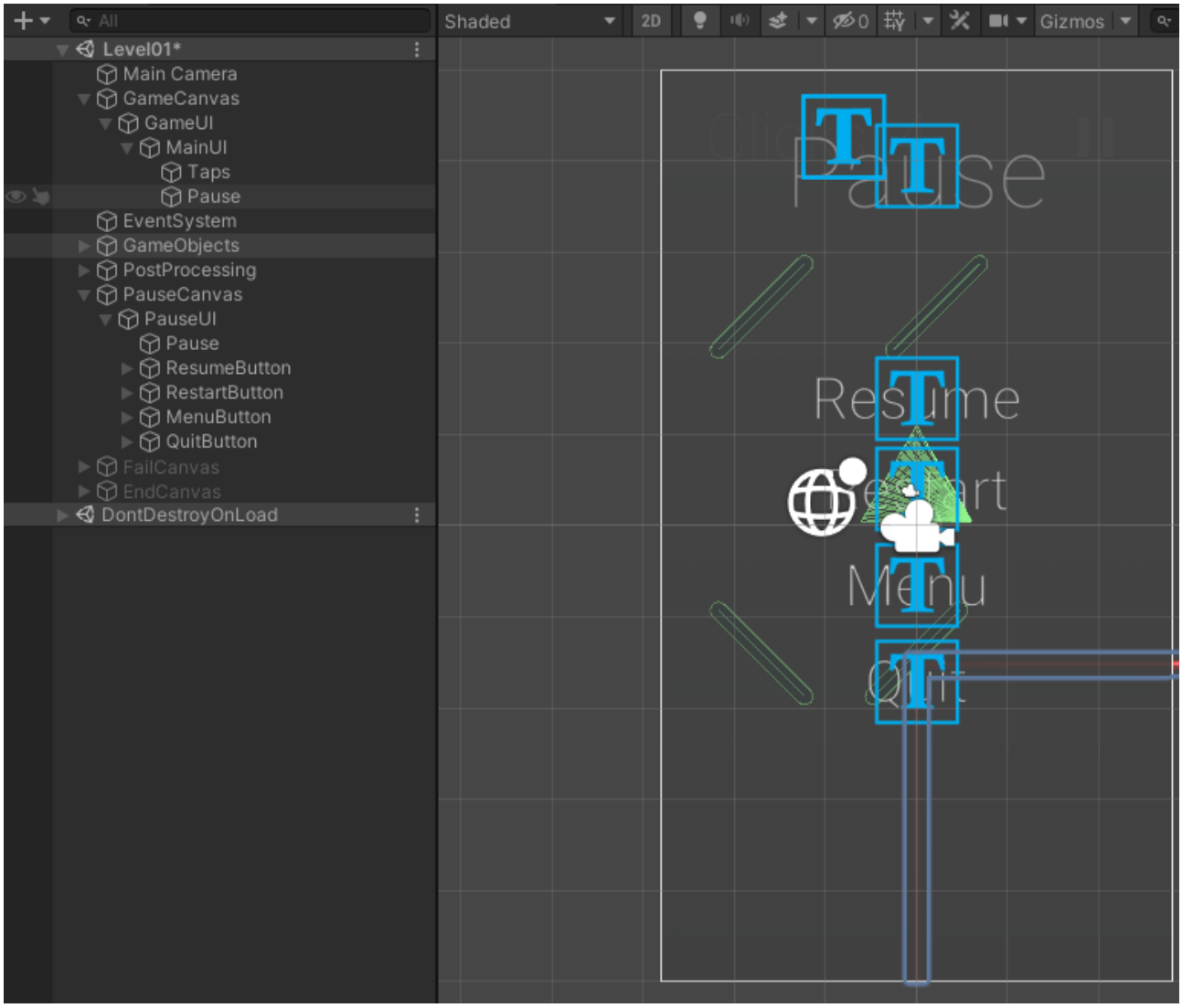

Рисунок 4.26 – Структура інтерфейсу паузи

Рисунок 4.27 – Код скрипту інтерфейсу паузи

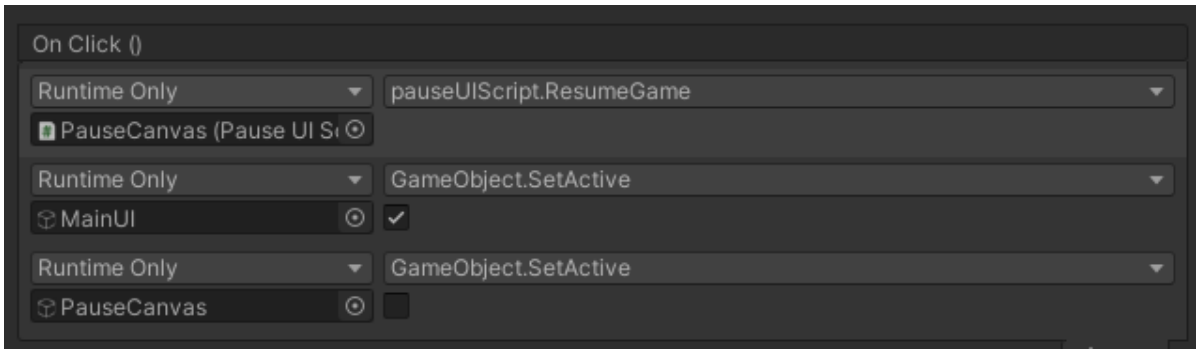

Рисунок 4.28 – Параметри кнопки «Resume»

Далі був розроблений інтерфейс невдалого проходження рівню. По аналогії з інтерфейсом паузи він включає в себе лише назву, та функціональні кнопки. Структуру можна побачити на рис. 4.29.

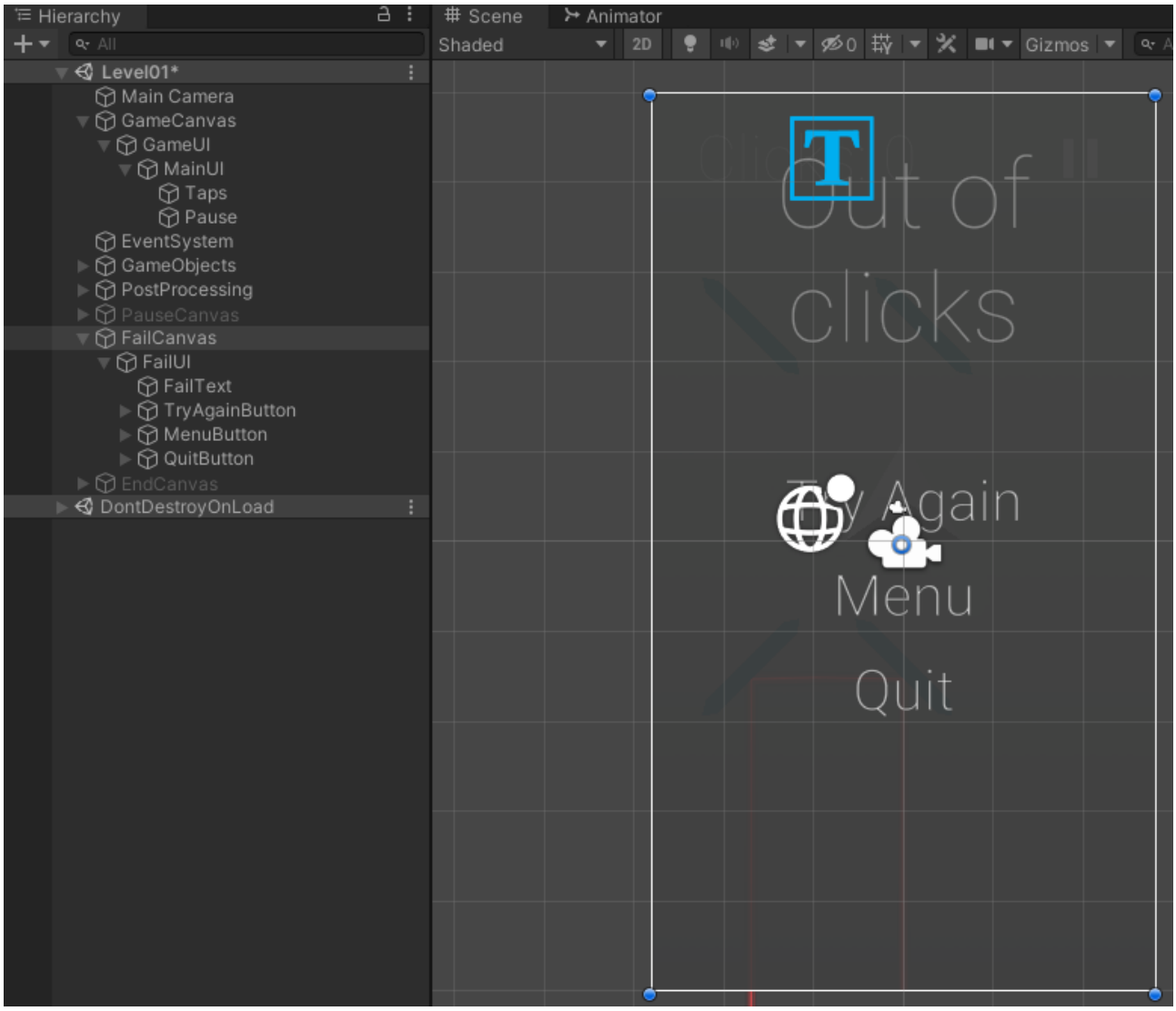

Рисунок 4.29 – Структура інтерфейсу невдалого проходження

Для функціювання кнопок був розроблений скрипт зображений на рис. 4.30.

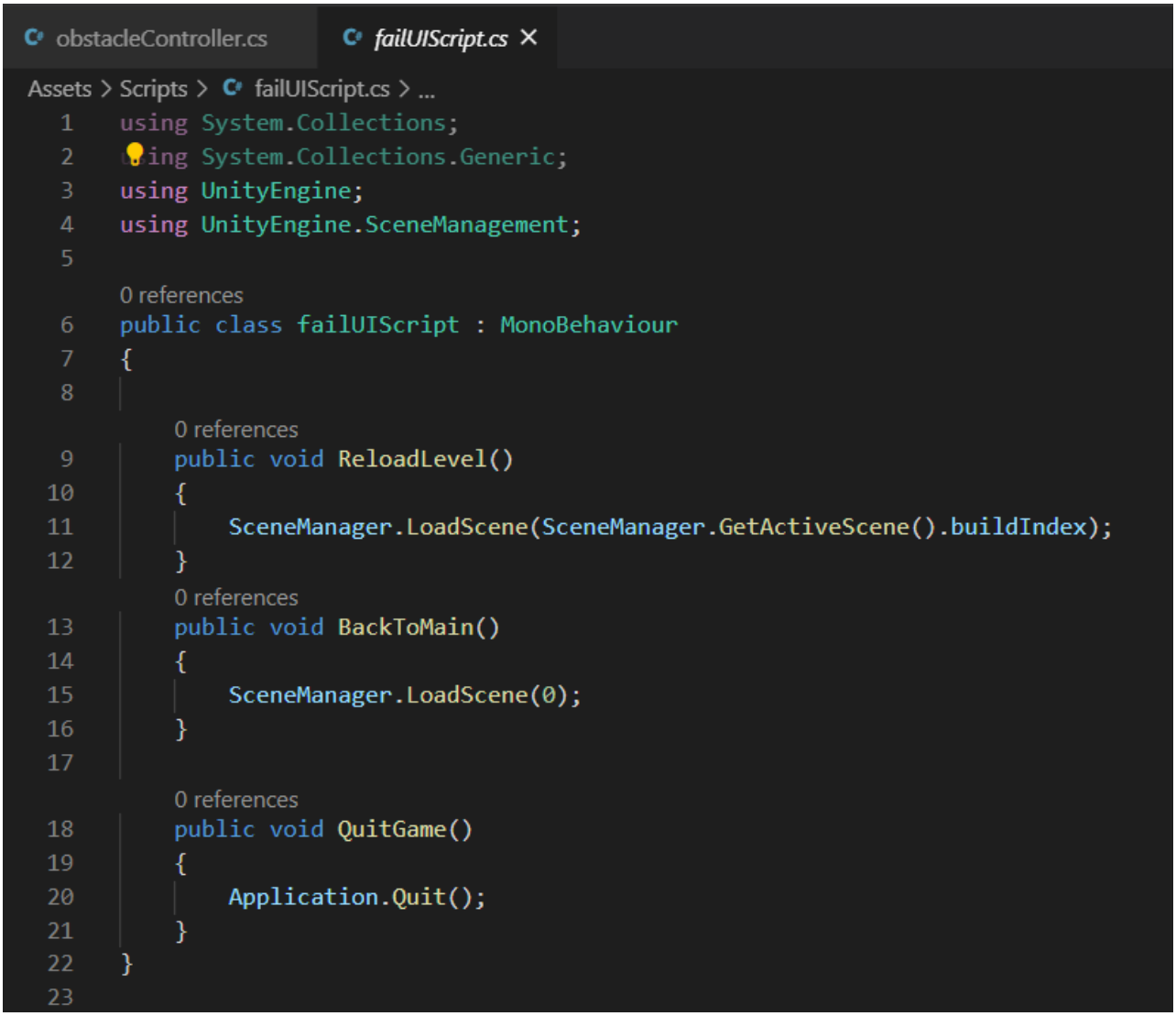

Рисунок 4.30 – Код скрипту інтерфейсу невдалого проходження

Останній інтерфейс який був розроблений – інтерфейс вдалого проходження рівня. Його структура включає в себе функціональні кнопки, та текст який відображає результат проходження рівня (рис. 4.31).

Для функціювання кнопок, та відображення результату проходження був розроблений скрипт (рис. 4.32). Даний скрипт відповідає за отримання кількості використаних натискань гравцем і виходячи із цього формує поточний

результат. Для відображення результату у скрипт необхідно було передати ігровий об’єкт тексту (рис. 4.33).

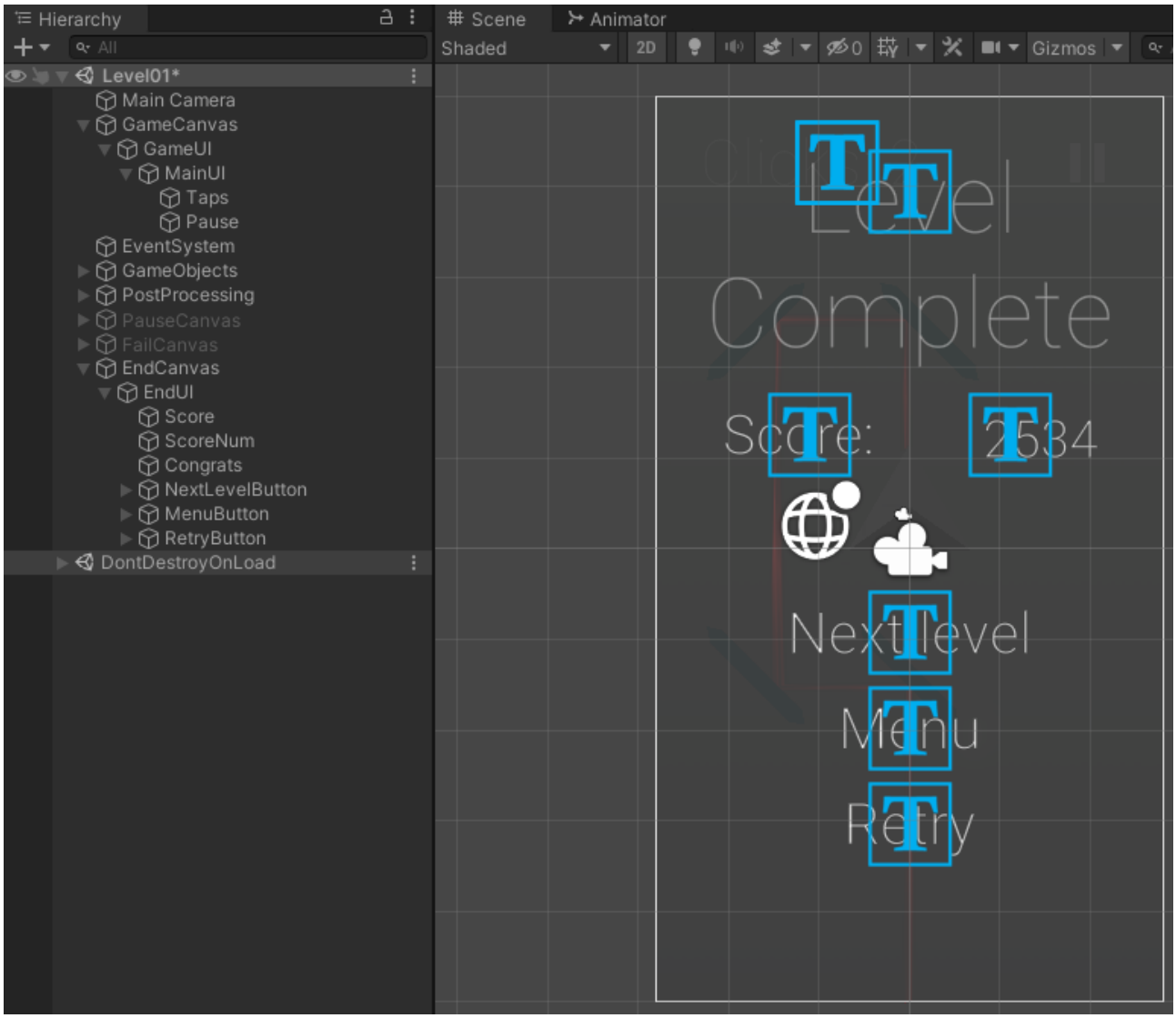

Рисунок 4.31 – Структура інтерфейсу вдалого проходження

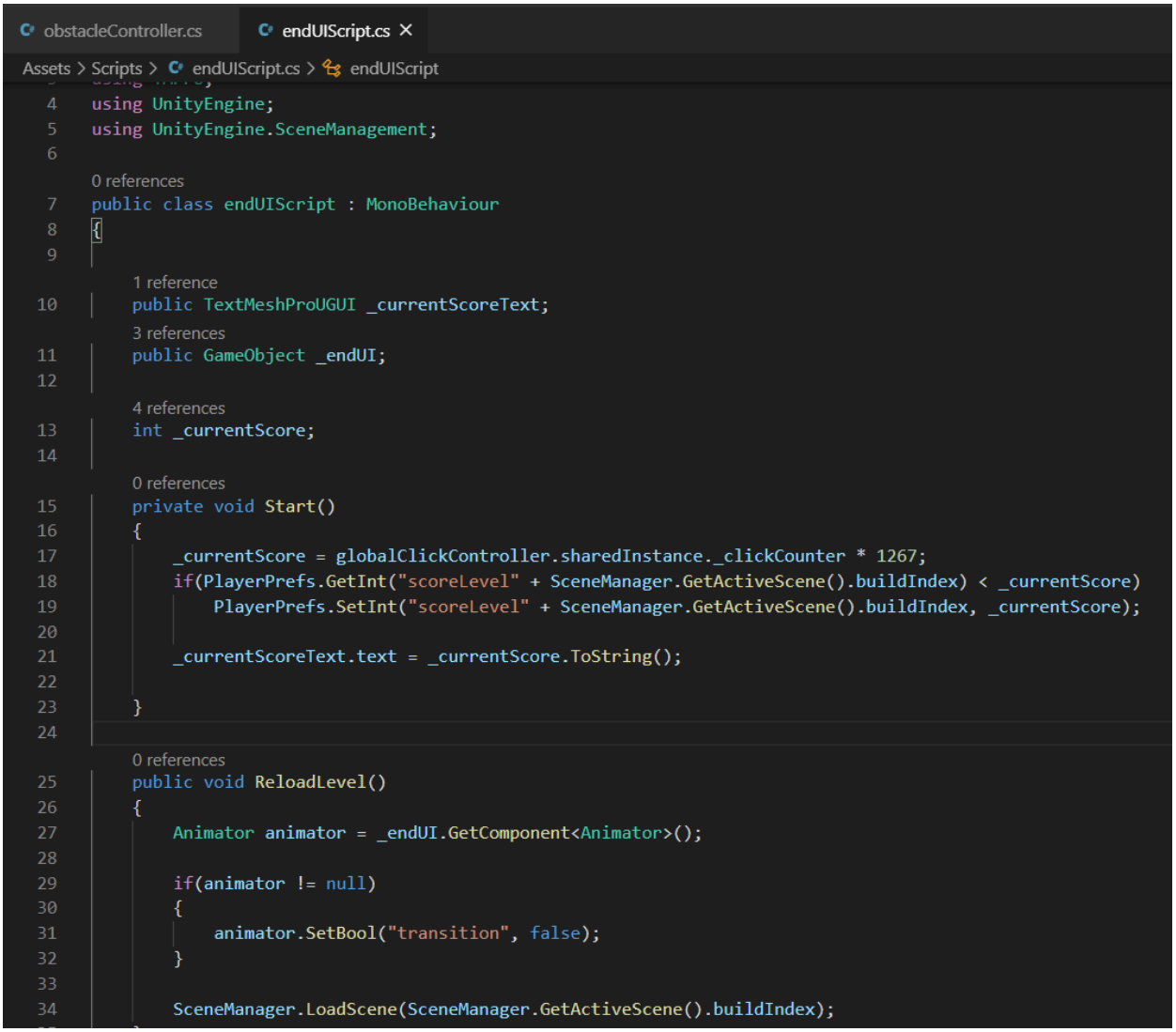

Рисунок 4.32 – Код скрипту інтерфейсу вдалого проходження

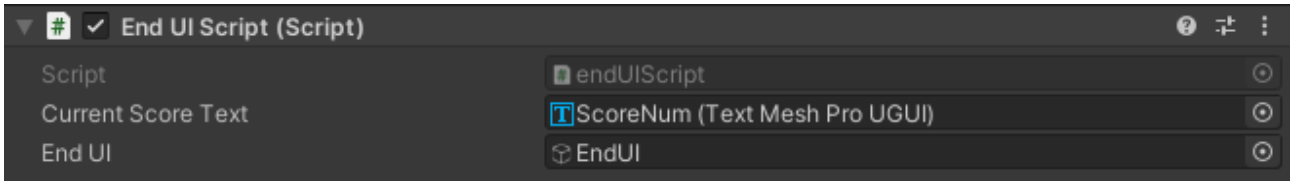

Рисунок 4.33 – Передача ігрового об’єкту у скрипт

4.3 Створення шейдерів та налаштування візуальних ефектів

Наступним етапом розробки є розробка спрайтів (елементів гри). Для цього була використана програма Adobe Illustrator. Першим чином необхідно створити ціль у вигляді піраміди (рис. 4.34). Для цього були створені три фігури та об'єднанні у групу. Структуру спрайта можна побачити на рис. 4.35.

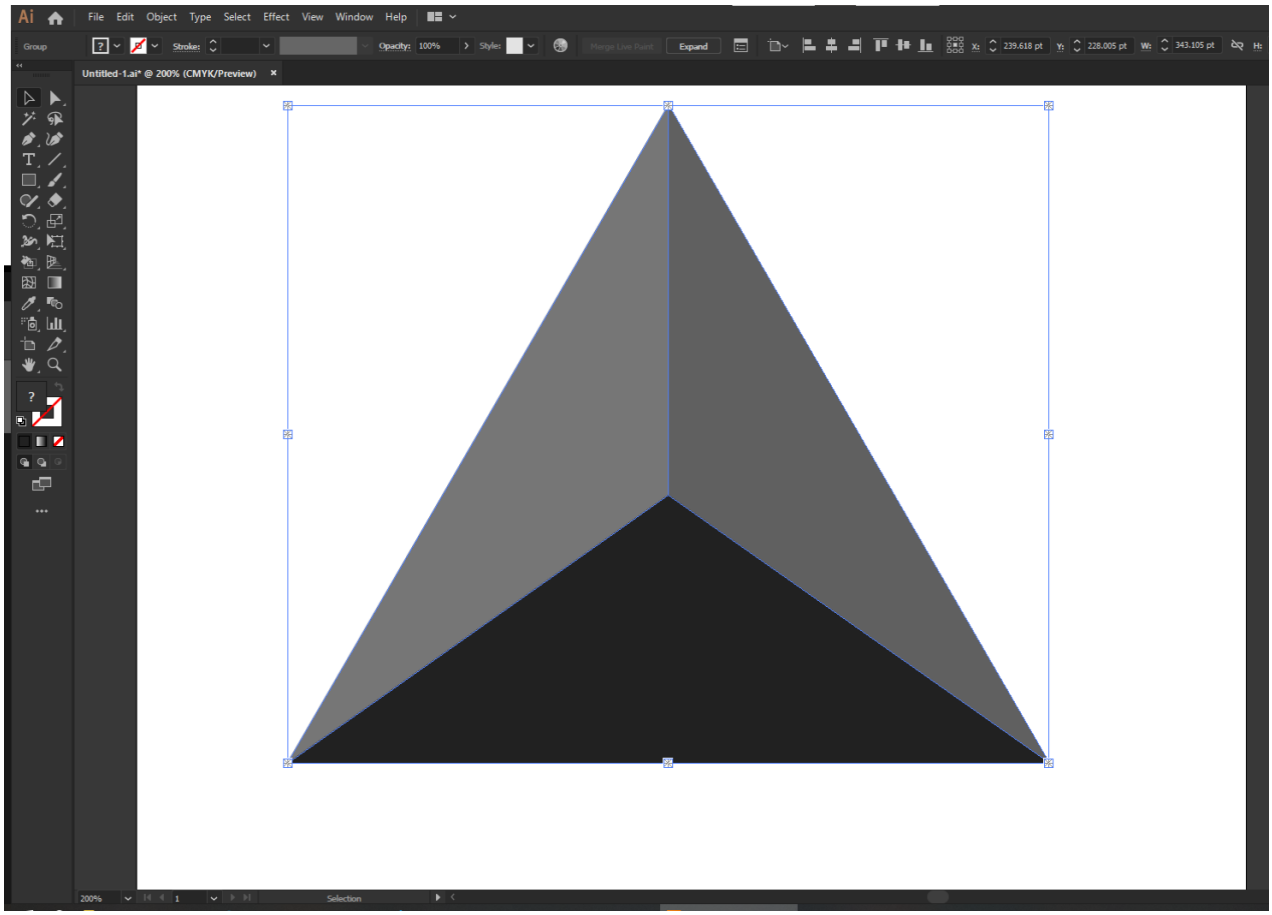

Рисунок 4.34 – Створення спрайту головної цілі

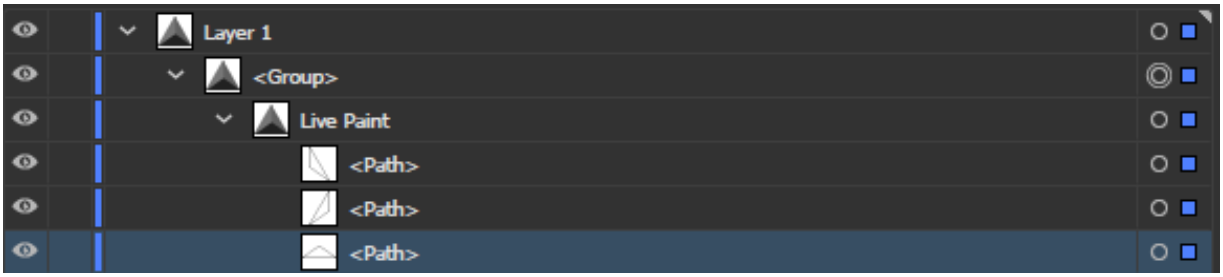

Рисунок 4.35 – Структура спрайта головної цілі

Після створення спрайту необхідно налаштувати його на сцені. Необхідно додати елементу колайдер (рис. 4.36), та визначити для нього тег (рис. 4.37), щоб у майбутньому використати його для ідентифікації.

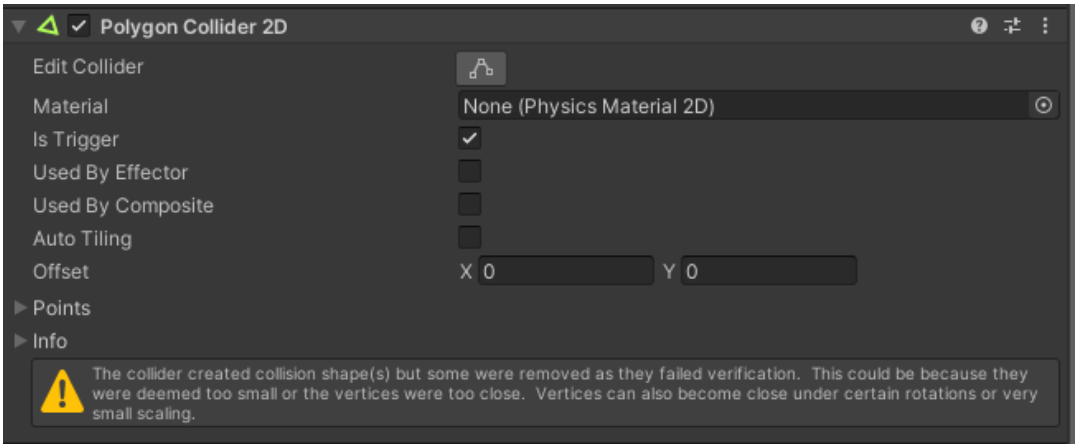

Рисунок 4.36 – Налаштування колайдеру головної цілі

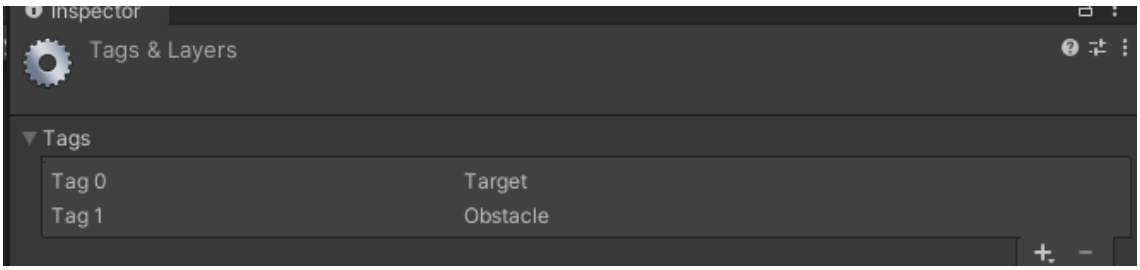

Рисунок 4.37 – Налаштування тегу головної цілі

Далі був розроблений спрайт дзеркала (рис. 4.38), після чого для нього буде розроблений шейдер.

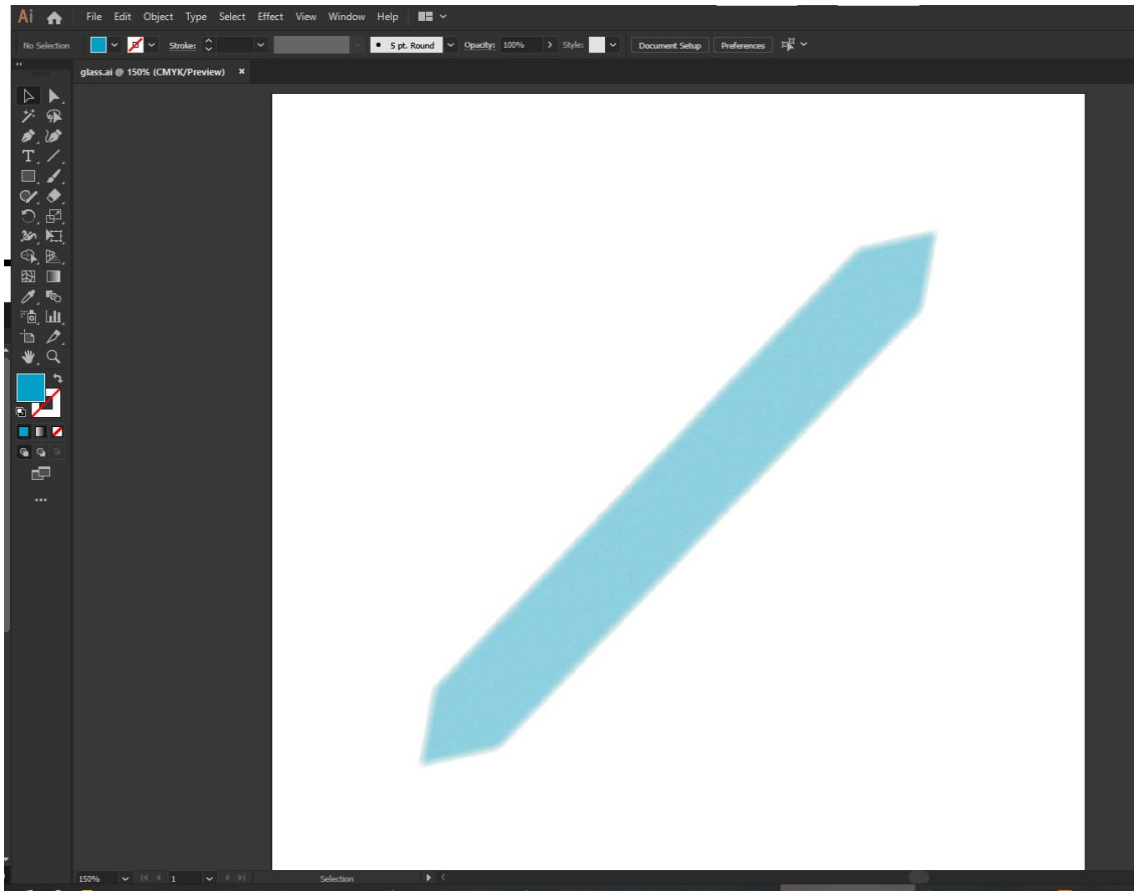

Рисунок 4.38 – Створення спрайту дзеркала

По аналогії з головною цілю необхідно додати елементу колайдер (рис. 4.39), та визначити для нього тег (рис. 4.40), щоб у майбутньому використати його для ідентифікації.

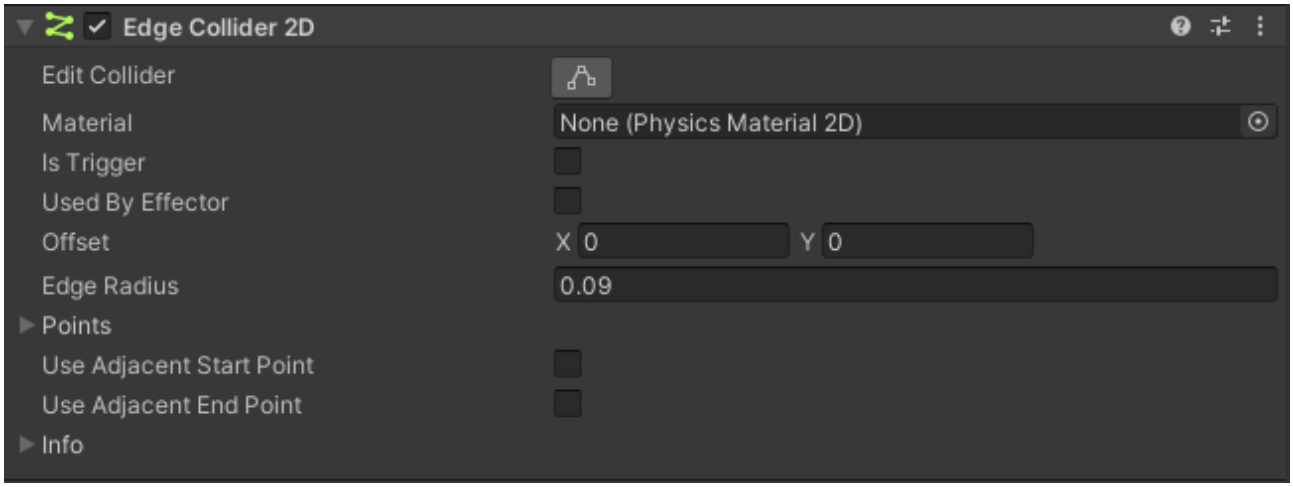

Рисунок 4.39 – Налаштування колайдеру спрайту дзеркала

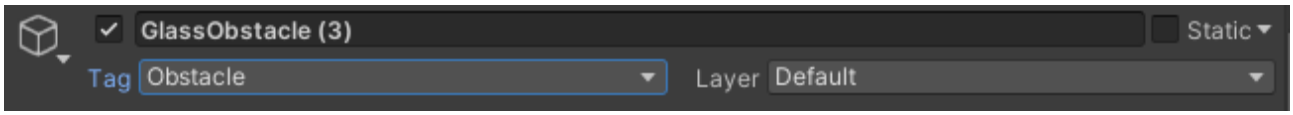

Рисунок 4.40 – Налаштування тегу спрайту дзеркала

Для того щоб перейти до наступного етапу розробки необхідно додати додатковий пакет до проекту, а саме пакет Universal RP (рис. 4.41). Даний пакет відповідає за відображення пост-обробки, та шейдерів у додатку.

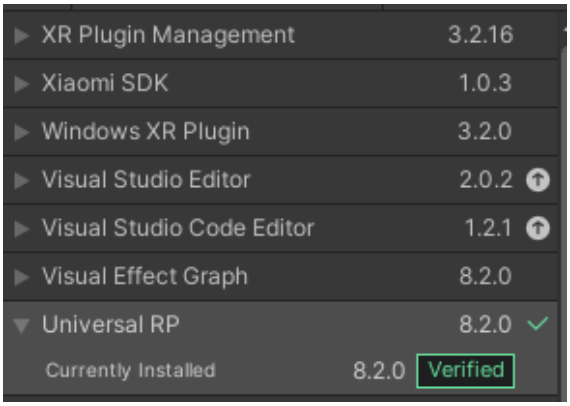

Рисунок 4.41 – Менеджер пакетів рушія Unity

Щоб гра не виглядала темною був доданий та налаштований елемент освітлення Global Light 2D (рис. 4.42).

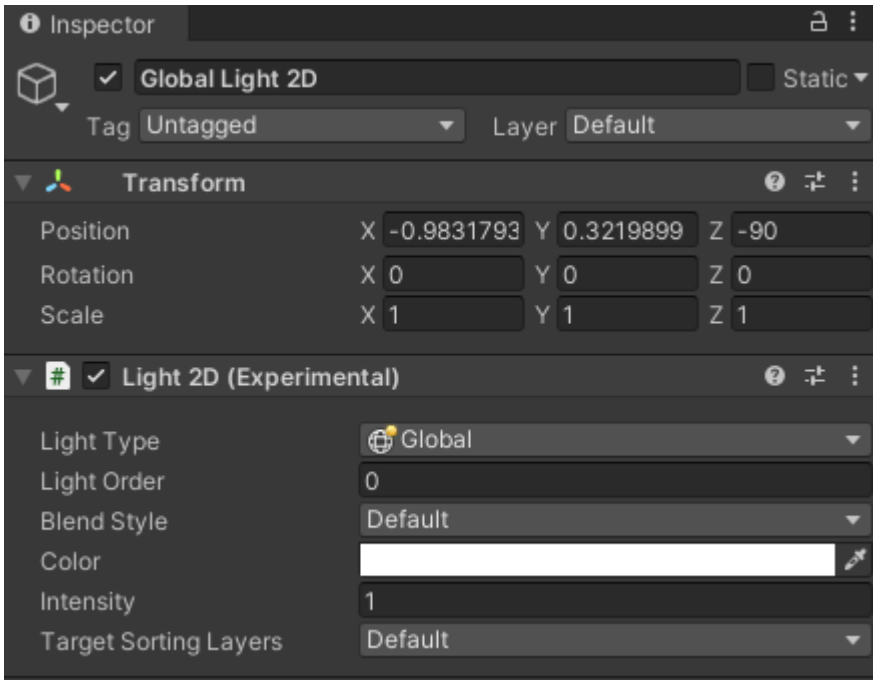

Рисунок 4.42 – Параметри елементу освітлення сцени

Для того щоб додати грі красок була додана пост-обробка, та створений глобальний профіль. До створеного профілю був доданий компонент світіння. Його налаштування можна побачити на рис. 4.43.

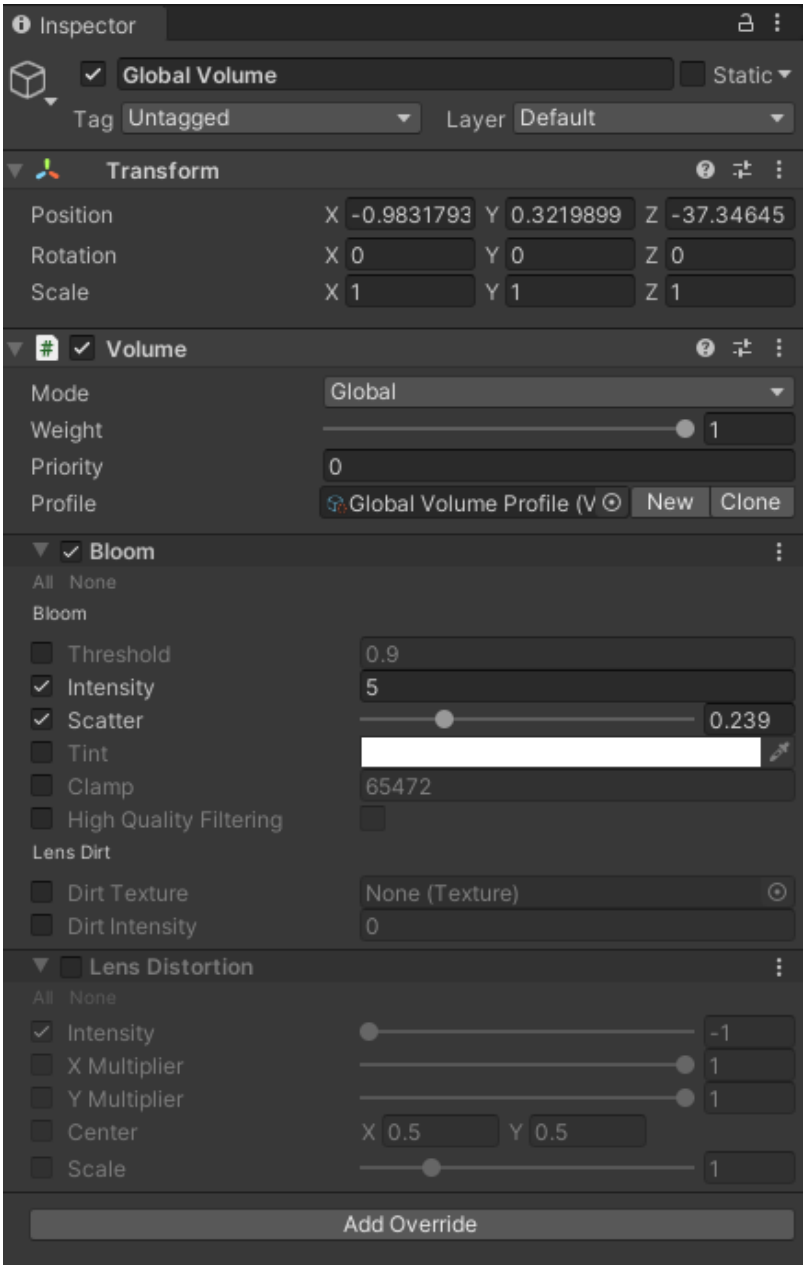

Рисунок 4.43 – Параметри елементу пост-обробки

Наступним етапом є створення ігрових анімацій. Розробка анімацій інтерфейсів майже ідентична, тому нижче буде показано процес створення на прикладі меню невдалого проходження. Для створення анімації необхідно додати аніматор до елемента. Для цього необхідно натиснути на кнопку

«Create» обравши необхідний елемент, та попередньо перейти на вкладку анімації (рис. 4.44).

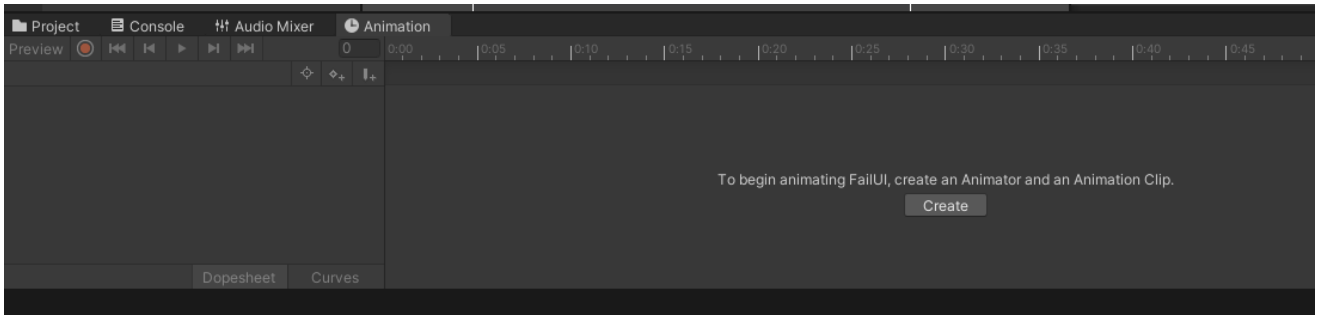

Рисунок 4.44 – Початок створення анімації

Після додавання аніматору були створені початкові ключі стану елементів на шкалі часу, завдяки цьому буде створена анімація меню (рис. 4.45).

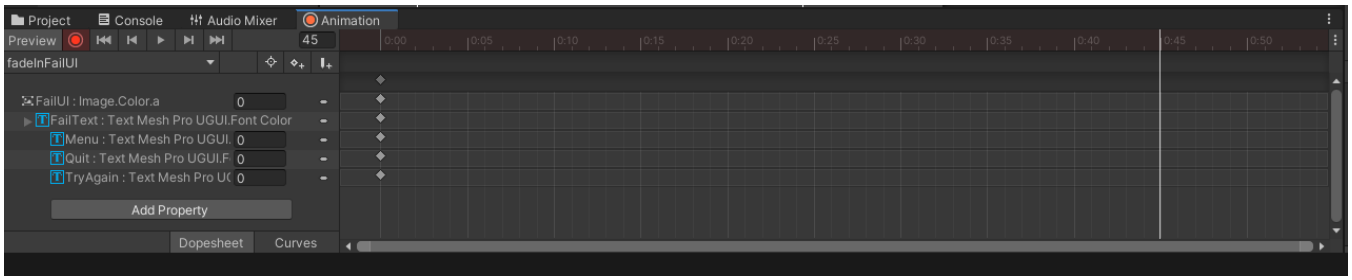

Рисунок 4.45 – Створення ключів на шкалі часу

Для того щоб анімація працювала необхідно додати ключі зміни стану і для інших позицій (рис. 4.46), для плавної анімації краще виділити більше часу.

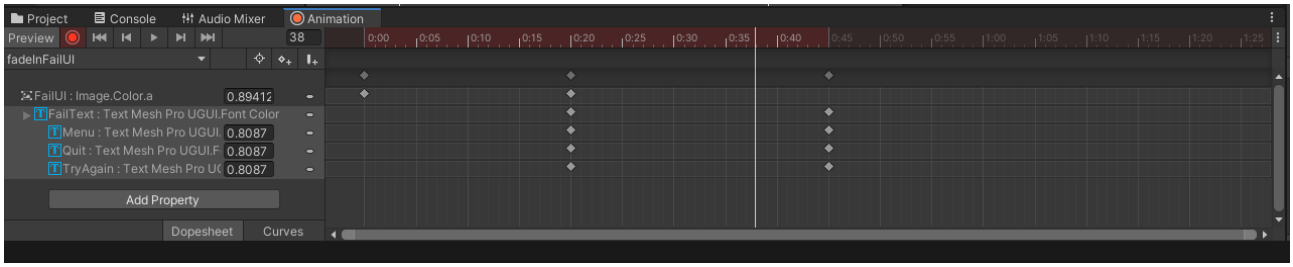

Рисунок 4.46 – Створення ключів на шкалі часу

Після створення анімації автоматично створиться файл аніматору. Аніматор використовують для зміни станів, а саме для програвання певної анімації коли об’єкт перебуває в зазначеному стані. Стандартний аніматор показано на рис. 4.47.

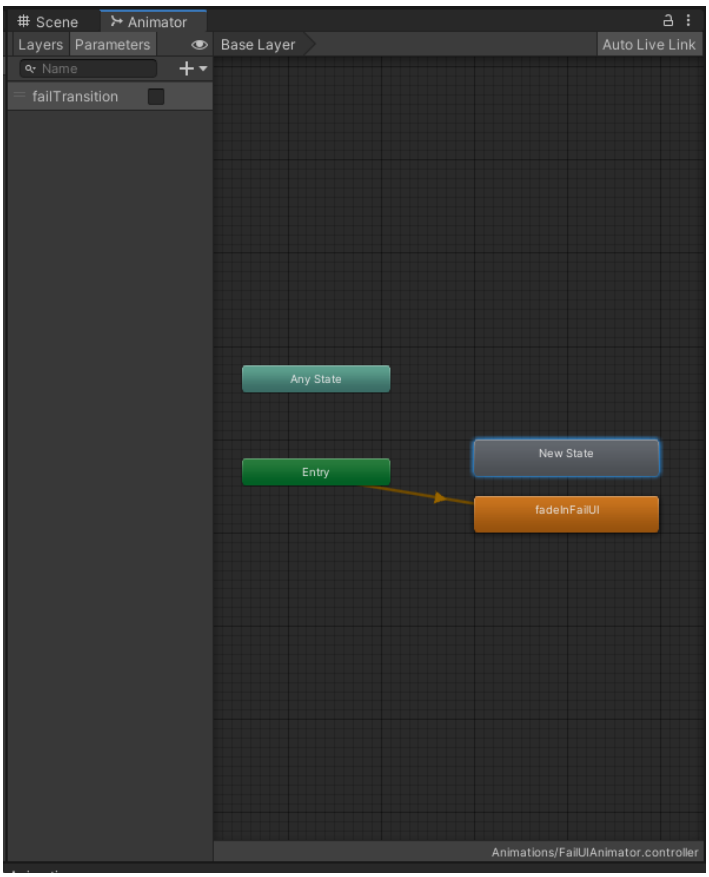

Рисунок 4.46 – Початковий вигляд аніматору

Для правильного функціювання станів був створений пустий стан, та змінна для контролю анімації (рис. 4.47). Коли змінна стає true програється анімація.

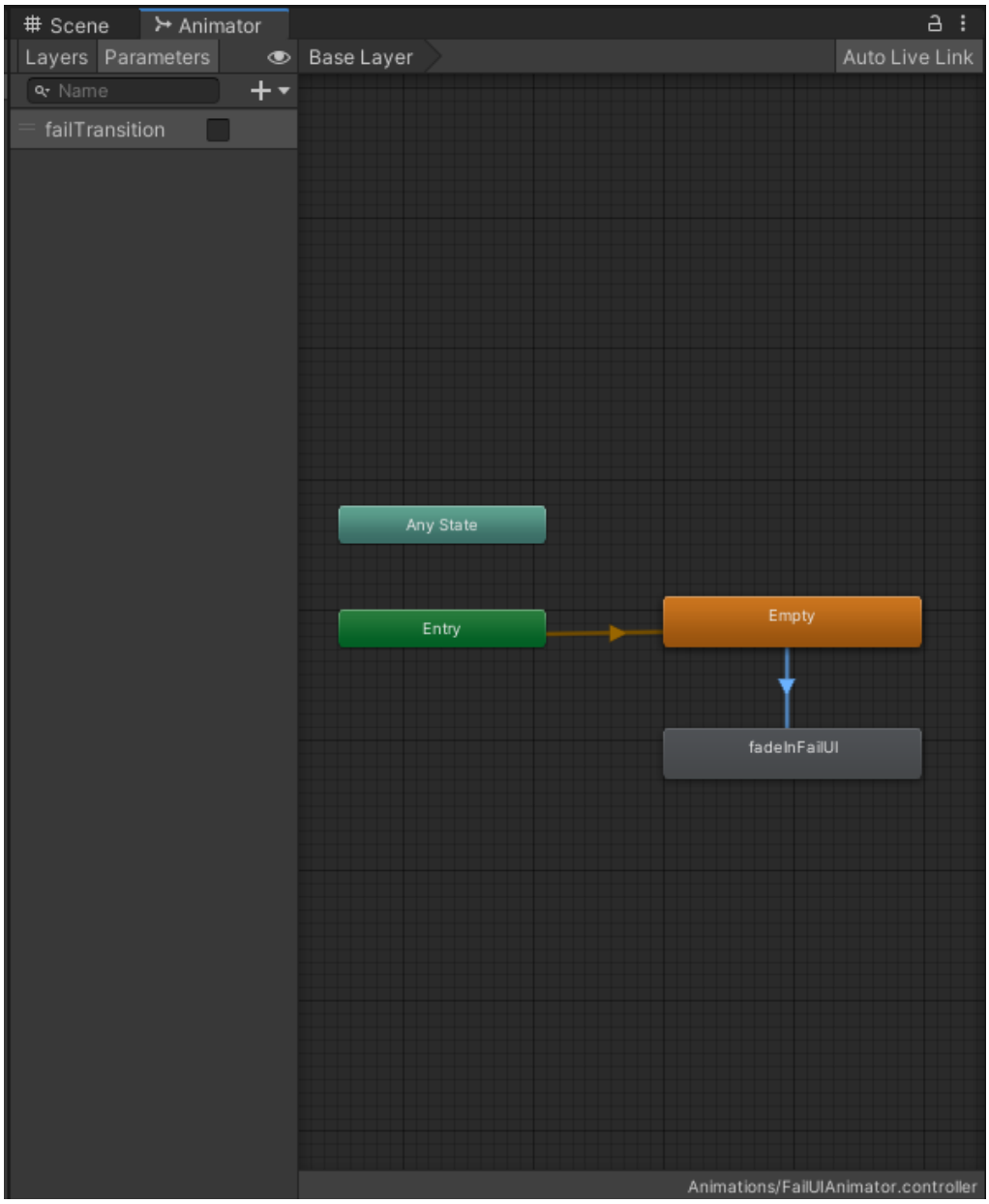

Рисунок 4.47 – Фінальний вигляд аніматору

Наступна анімація яка була розроблена – анімація обертання дзеркала. Для її розробки необхідно було створити три окремі анімації (рис. 4.48). Кожна анімація починається с певної позиції об’єкта. Виходячи з цього необхідно було налаштувати аніматор таким чином, щоб анімації програвались коректно.

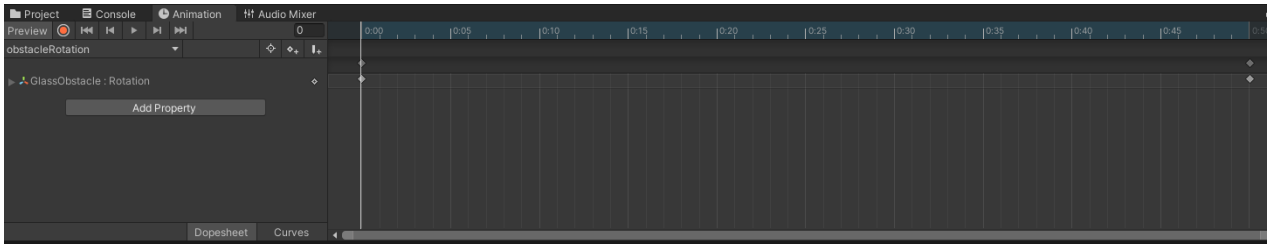

Рисунок 4.48 – Перша анімація дзеркала на шкалі часу

Як вже було сказано, створити анімацію не складає труднощів, основна проблема у правильній послідовності їх програвання. Для того щоб анімації програвались правильно, необхідно налаштувати аніматор (рис. 4.49).

Для правильного програвання анімації було додано три змінні які призначені для контролю переходів між анімаціями, та одну пусту анімацію, яка була встановлена початковою, щоб програвання починалося лише коли гравець натискає на кнопку.

Щоб перемикання створених змінних відбувалося коректно був розроблений скрипт керування ігровим об’єктом (рис. 4.50). З повним кодом скрипта можна ознайомитися у додатку Б.

Створений скрипт відповідає за отримання інформації про поточне положення об’єкту та вирішує яку змінну треба активувати для програвання необхідної анімації.

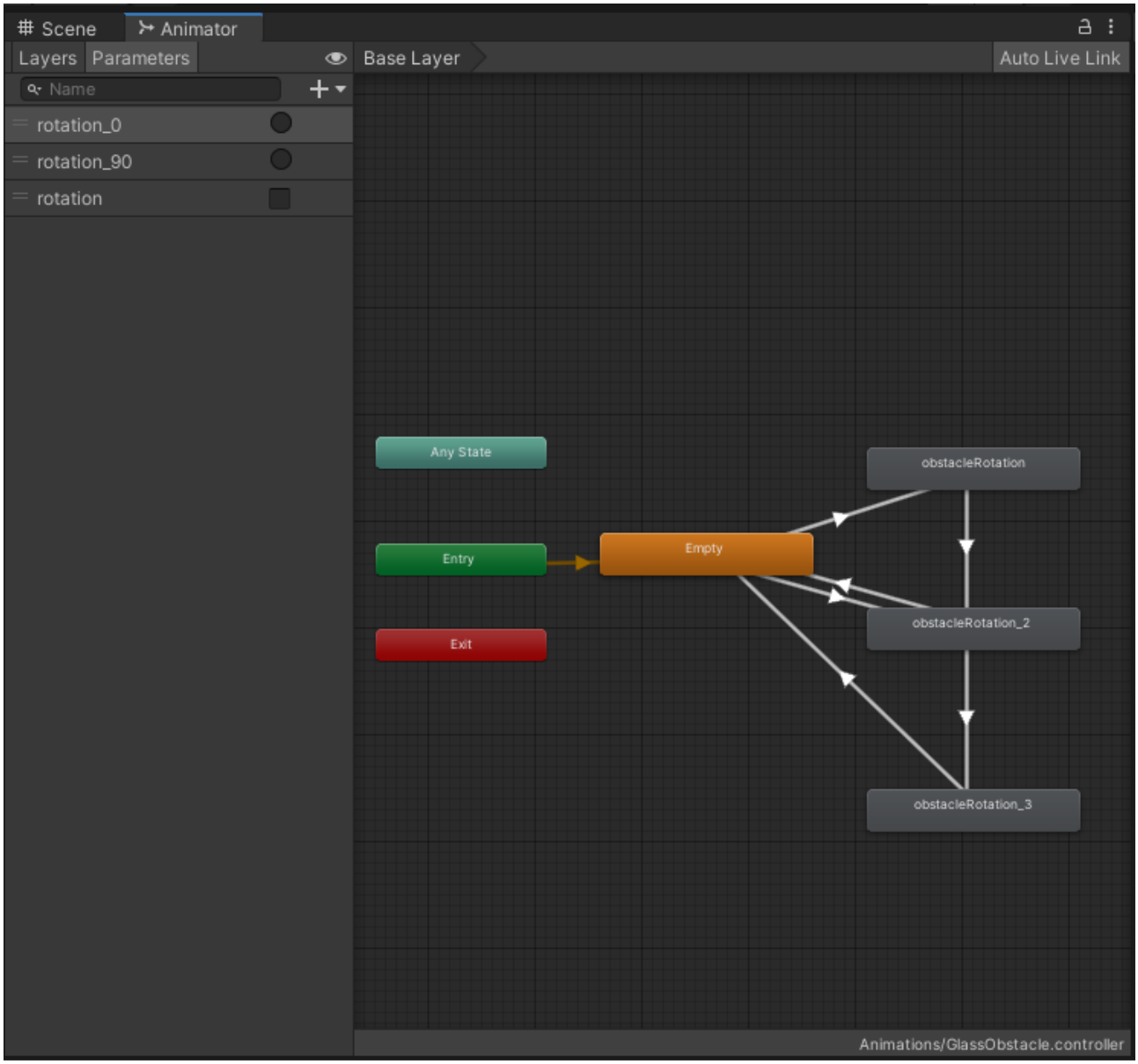

Рисунок 4.49 – Фінальний вигляд аніматору обертання ігрового об'єкта

obstacleController.cs

Assets > Scripts > obstacleController.cs

```
using UnityEngine.UI;

public class obstacleController : MonoBehaviour
{

    public GameObject _obstacle;
    private Animator _animator;

    void OnMouseDown()
    {
        if(Time.timeScale == 1f){
            _animator = _obstacle.GetComponent<Animator>();

            if(_animator != null)
            {
                if(_obstacle.transform.rotation.eulerAngles.z == 0 || _obstacle.transform.rotation.eulerAngles.z == 180)
                {
                    _animator.SetTrigger("rotation_0");
                    globalRotatingController.sharedInstance._isRotating = true;
                }

                else if(_obstacle.transform.rotation.eulerAngles.z == 90)
                {
                    _animator.SetTrigger("rotation_90");
                    globalRotatingController.sharedInstance._isRotating = true;
                }

                else
                {
                    _animator.SetTrigger("rotation_90");
                    _animator.SetBool("rotation", true);
                    globalRotatingController.sharedInstance._isRotating = true;
                }
            }
            Invoke("SetRotatingFalse", 1f);
            globalClickController.sharedInstance._clickCounter--;
        }

    }

```

Рисунок 4.50 – Код скрипта обертання ігрового об’єкта дзеркала

Останній етап пост-обробки додатку це створення шейдерів, а саме шейдеру лазера, та шейдеру дзеркала. Після цього на їх основі будуть створені матеріали для використання на ігровій сцені.

Для коректного відображення лазеру, у середовище ігрового рушія Unity, був створений файл з розширенням shadergraph (рис. 4.51), та був розроблений шейдер (рис. 4.52).

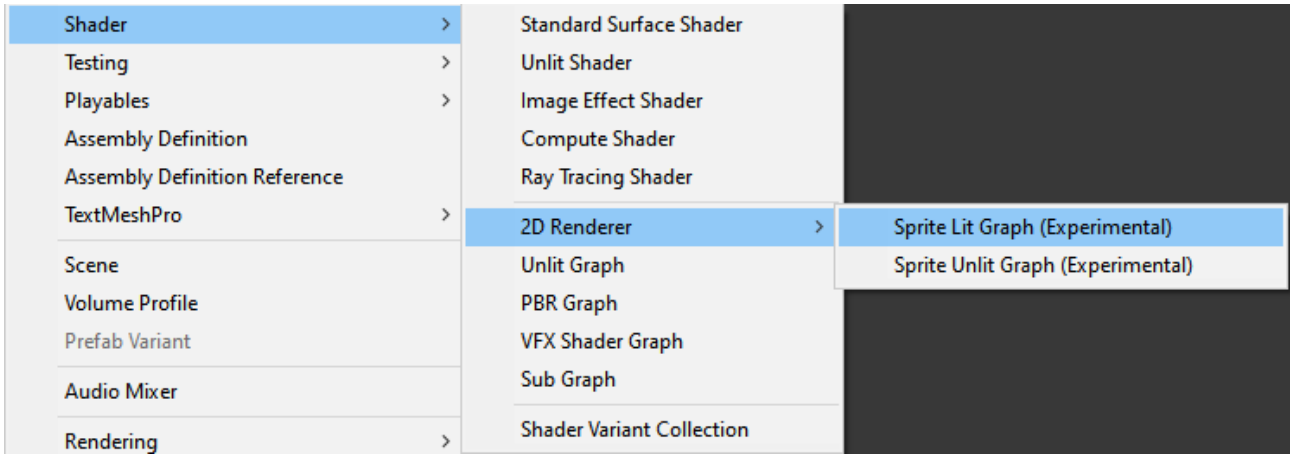

Рисунок 4.51 – Створення шейдеру лазера

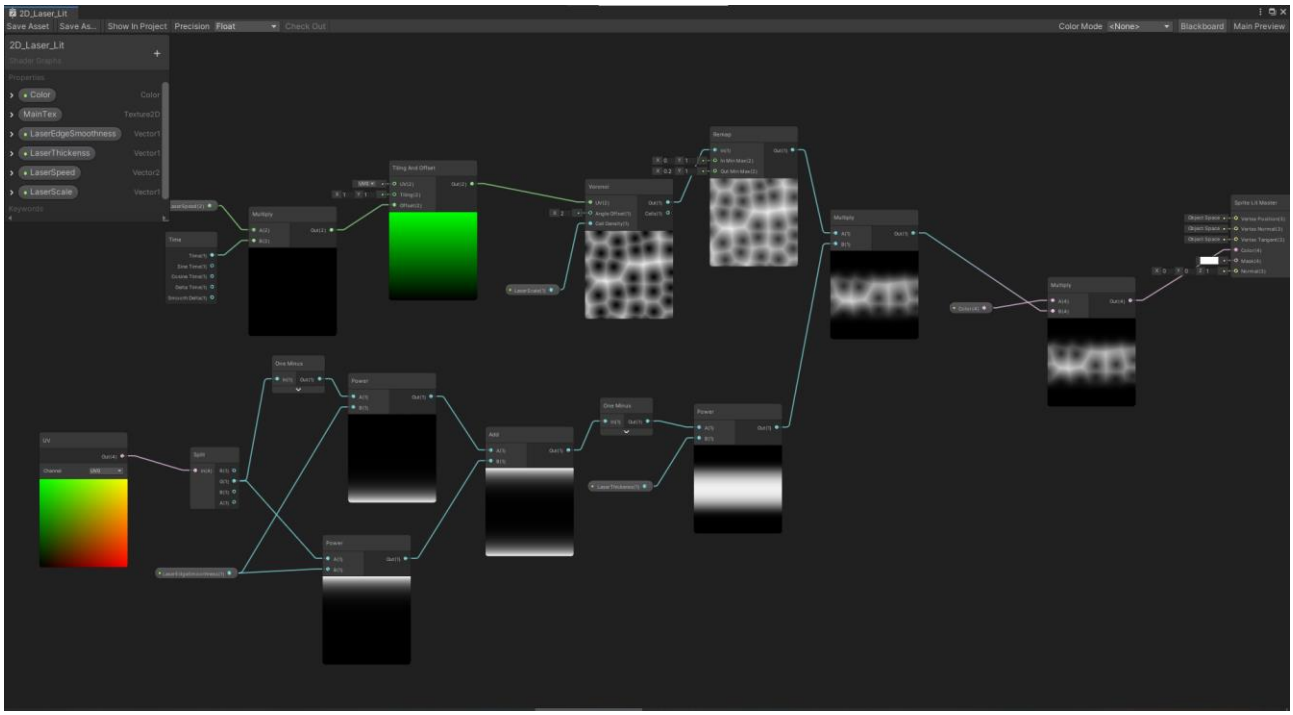

Рисунок 4.52 – Фінальний вигляд шейдеру

Даний шейдер включає в себе п’ять змінних (рис. 4.53), кожна із яких відповідає за певний параметр лазеру.

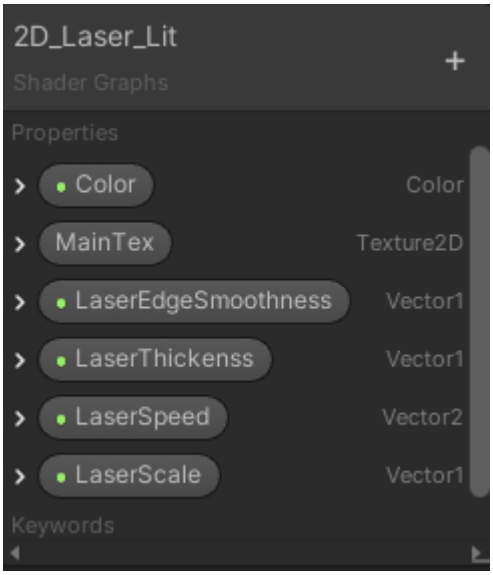

Рисунок 4.53 – Змінні шейдеру

Змінна «Color» відповідає за колір лазеру, «LaserEdgeSmoothness» відповідає за гладкість країв лазера, «LaserThickenss» в свою чергу відповідає за товщину лазера, «LaserSpeed» відповідає за швидкість анімації лазеру, та «LaserScale» за щільність частинок що імітують світлові частки.

Умовно шейдер можна поділити на дві частини, перша частина відповідає за формування лазеру, його товщину та чіткість (рис. 4.54), друга частина відповідає за анімацію лазеру, а саме за імітацію руху потоку світлових часток (рис. 4.55).

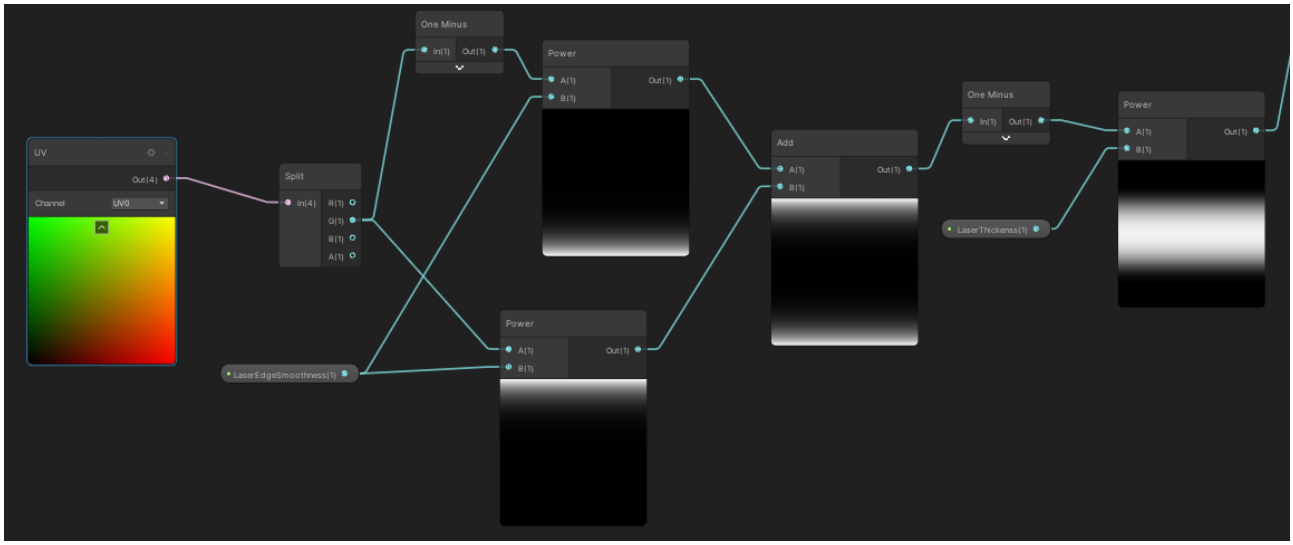

Рисунок 4.54 – Перша умовна частина шейдеру

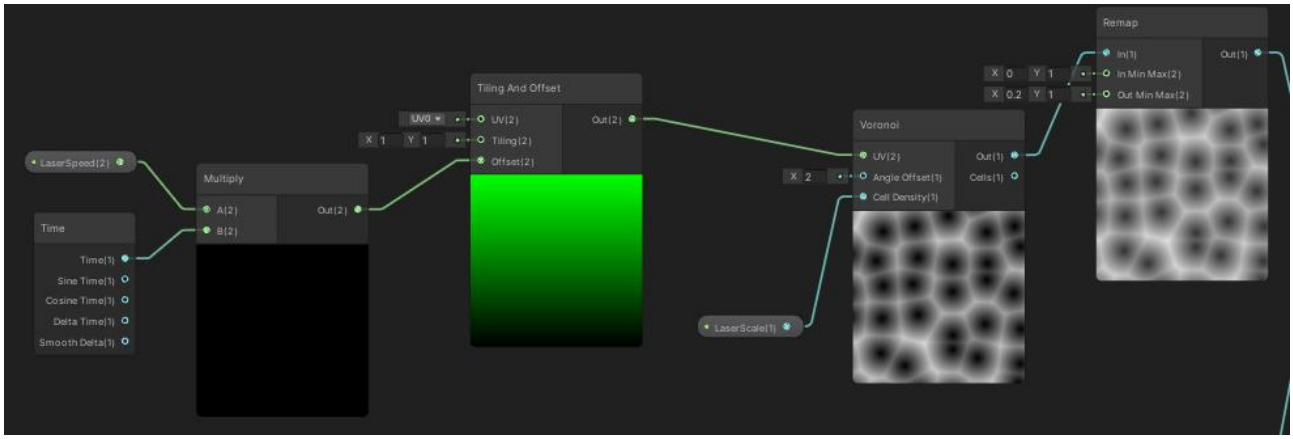

Рисунок 4.55 – Друга умовна частина шейдеру

Після того як шейдер було розроблено, необхідно було створити на його основі матеріал (рис. 4.56).

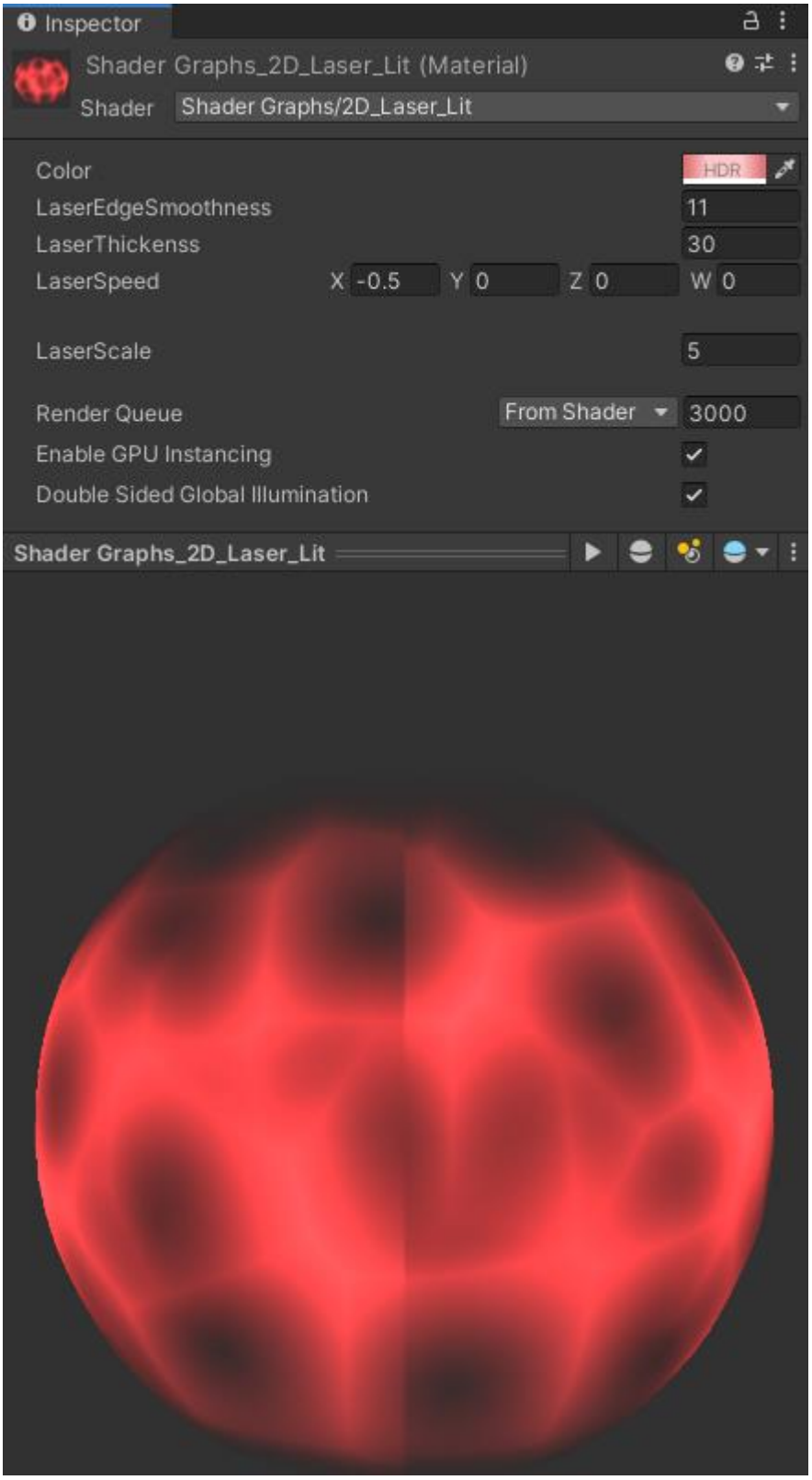

Рисунок 4.56 – Матеріал на основі шейдеру

Останній етап це додавання матеріалу до лазеру (рис. 4.57), та перегляд фінального результату (рис. 4.58).

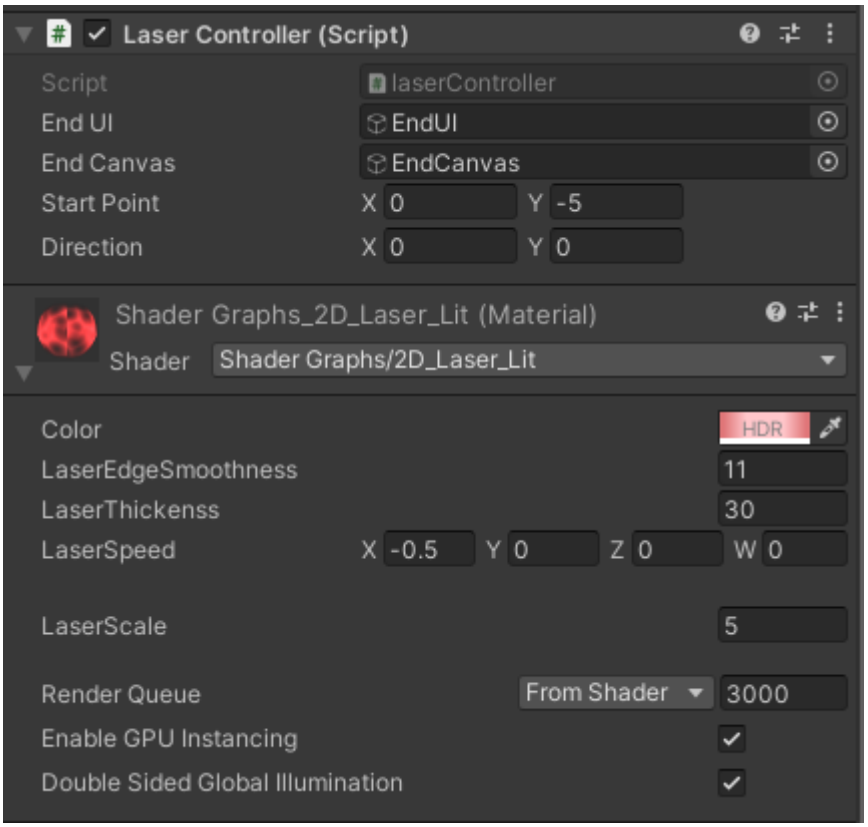

Рисунок 4.57 – Параметри лазеру із доданим матеріалом

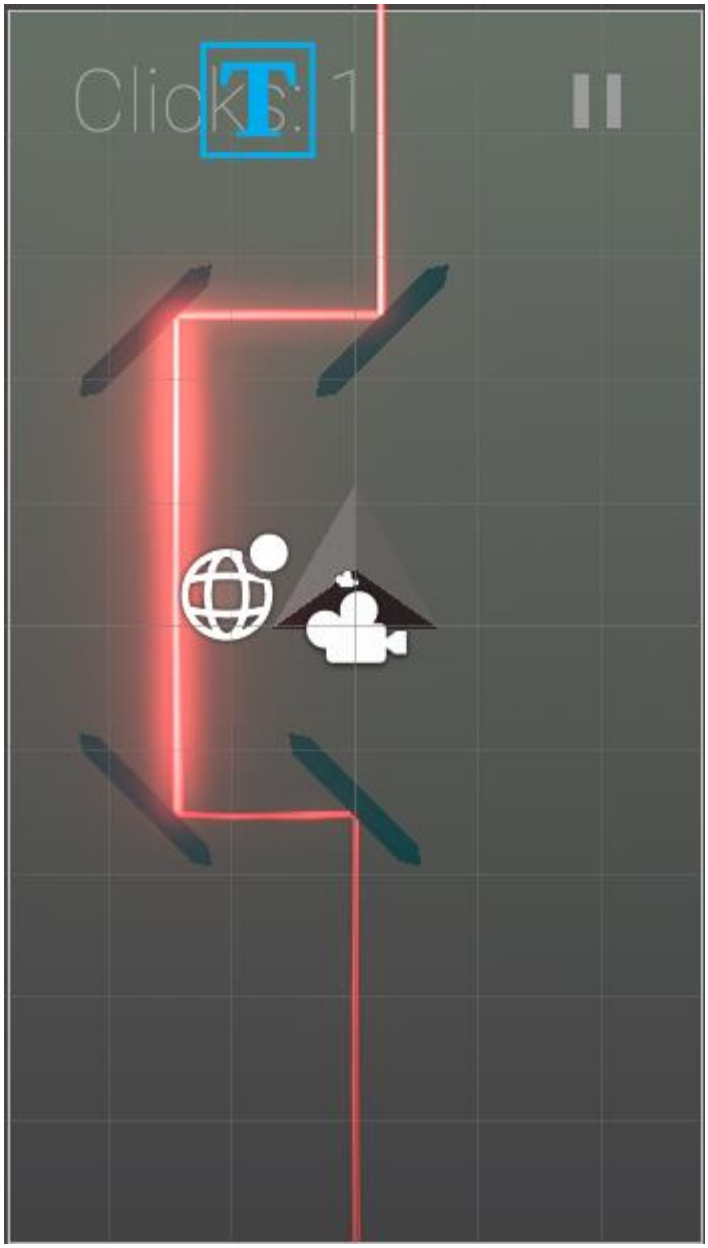

Рисунок 4.58 – Лазер з шейдером на ігровій сцені

4.4 Налаштування проекту та створення фінальної збірки

До налаштувань проекту можна віднести фінальні штрихи, такі як створення ігрових рівнів, додавання їх до збірки гри, та тестування додатку.

Також сюди відноситься процес налаштування проекту (налаштування орієнтації, додавання іконки, та створення файлу проекту).

Першим чином була додана іконка ігрового додатку, для цього у налаштуваннях проекту була встановлена іконка для старих девайсів (Legacy) (рис. 4.59), та для нових девайсів (API level 26+), які працюють на базі операційної системи Android версії 8.0 та вище (рис. 4.60).

Рисунок 4.59 – Встановлення іконки для старих пристроїв

Рисунок 4.60 – Встановлення іконки для нових пристроїв

Для коректного відображення гри на екрані мобільного пристрою було налаштована орієнтація гри при запуску. У настройках проекту у якості стандартної орієнтації була встановлена портретна, без можливості зміни (рис. 4.61).

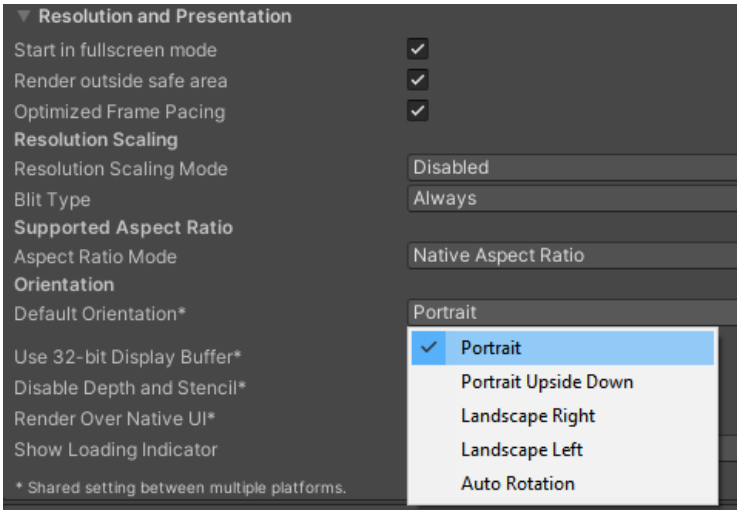

Рисунок 4.61 – Налаштування орієнтації ігрового додатку

Через те що перший рівень гри був розроблений уніфікованим, розробка наступних рівнів не викликає великих проблем. Потрібно лише скопіювати рівень, налаштувати його розташувавши ціль, дзеркала та лазер (рис. 4.62), і вказати доступну гравцю кількість натискань (рис. 4.63), після чого додати рівень у менеджер рівнів (рис. 4.64).

Створення ігрового рівня показане на прикладі другого рівня.

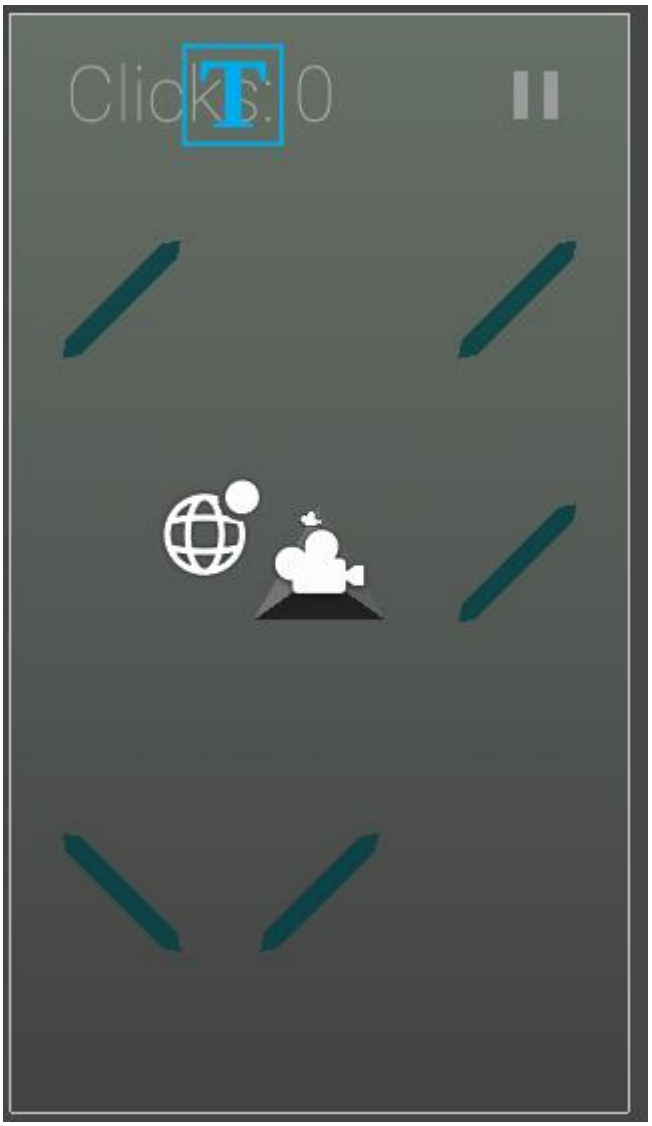

Рисунок 4.62 – Розташування ігрових об’єктів на сцені

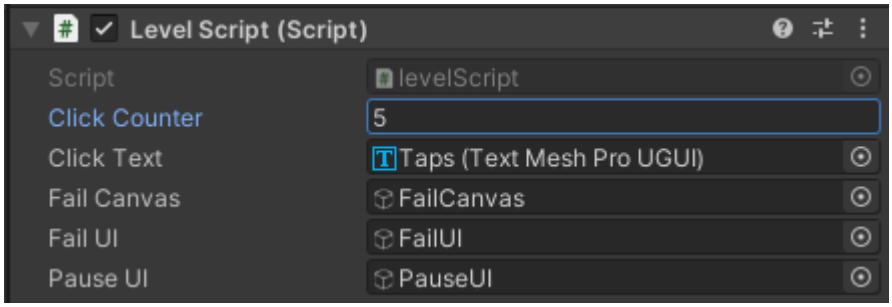

Рисунок 4.63 – Налаштування кількості доступних гравцю натискань

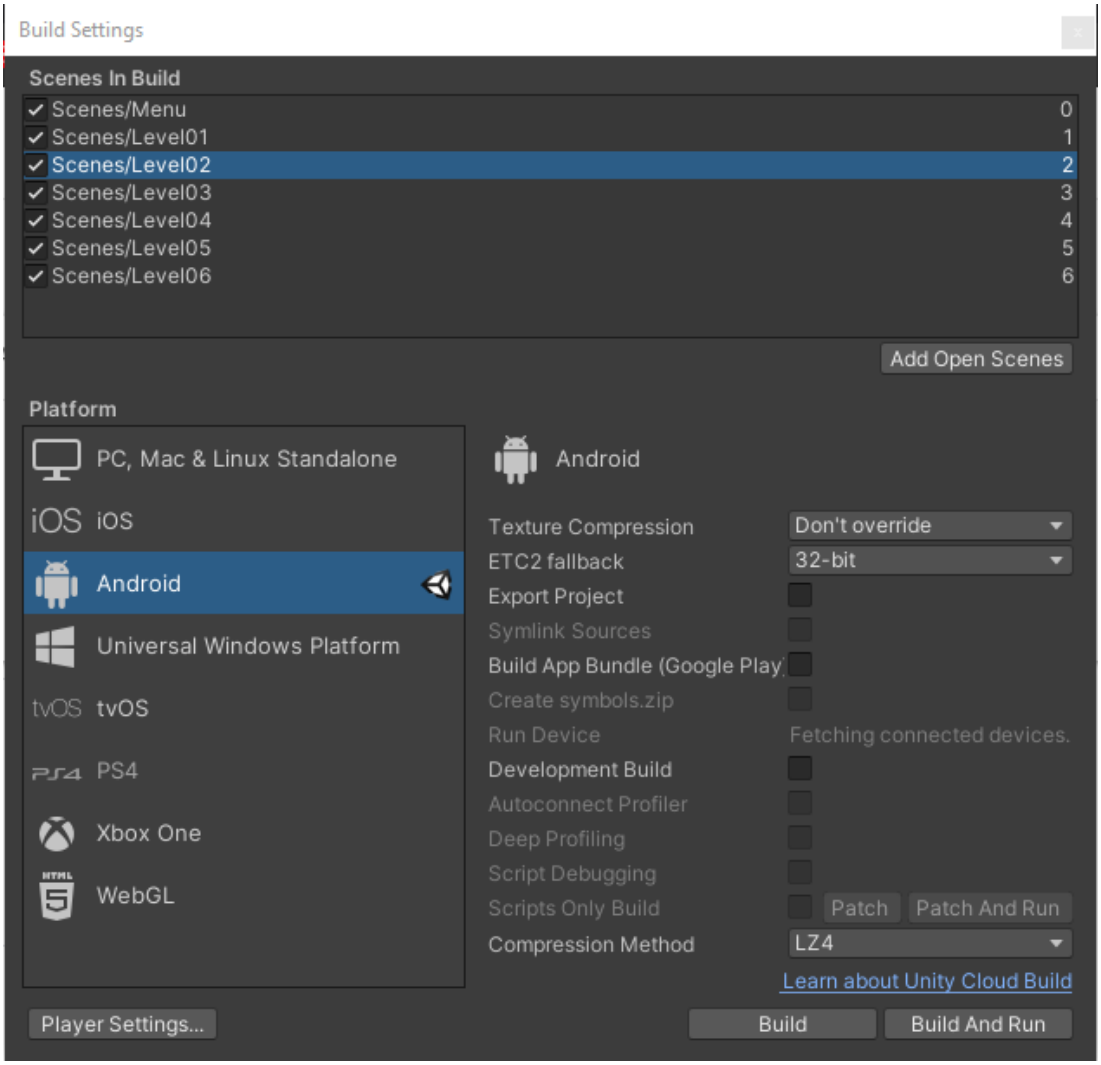

Рисунок 4.64 – Додавання рівню у менеджер рівнів

Після створення усіх ігрових рівнів необхідно лише натиснути на кнопку «Build» у налаштуваннях збірки, обрати місце зберігання файлу (рис. 4.65), після чого ігровий рушій налаштує та скомпілює готовий проект (рис. 4.66).

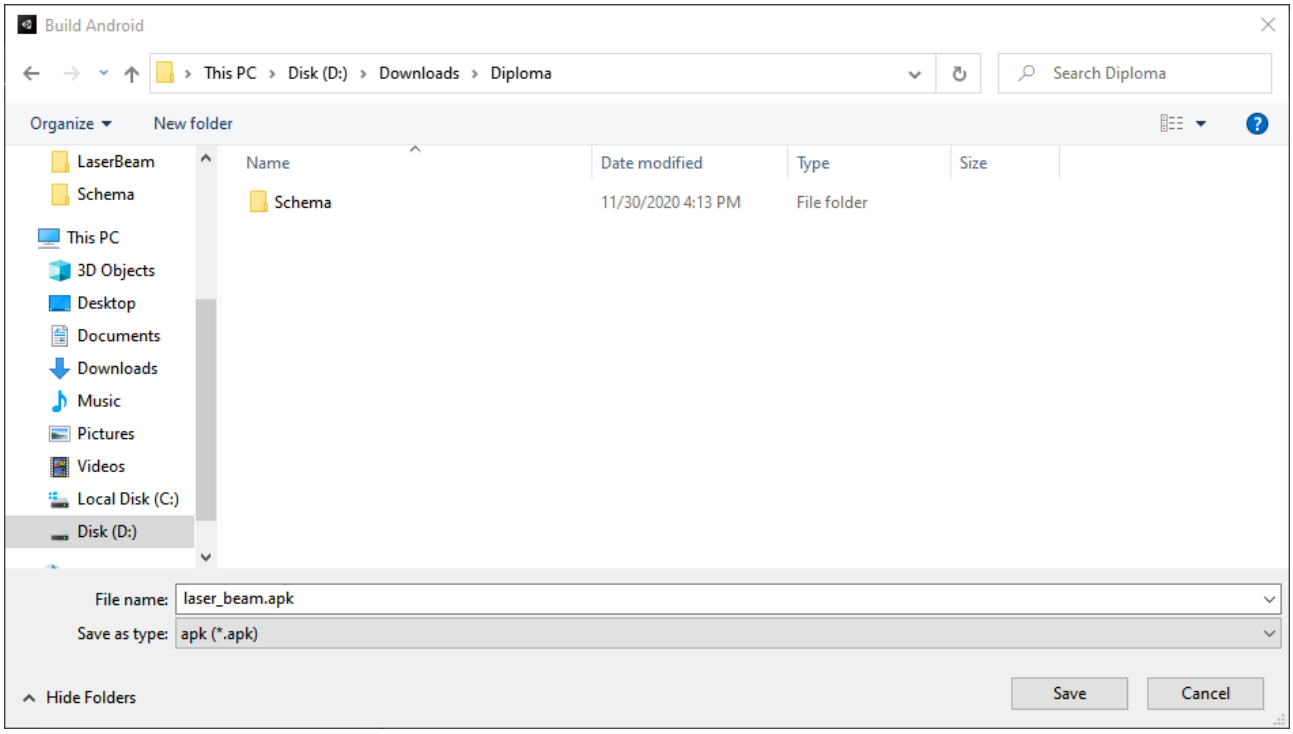

Рисунок 4.65 – Вибір місця розташування

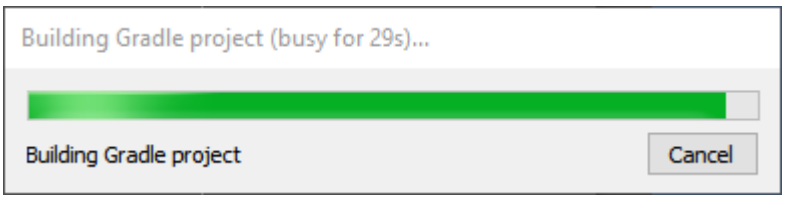

Рисунок 4.66 – Процес компіляції проекту

4.5 Тестування мобільного ігрового додатку

Після успішного проходження усіх етапів розробки, мобільний додаток був протестований.

Першим чином було протестоване головне меню, його відображення, анімація, функціонал кнопок (рис. 4.67).

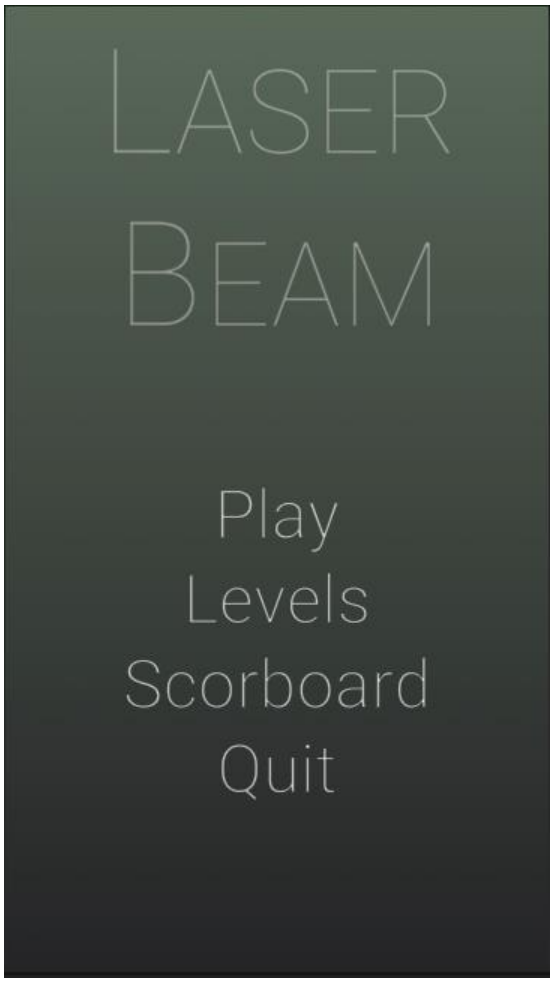

Рисунок 4.67 – Головне меню

Далі було протестоване меню вибору ігрового рівня, правильності відображення доступних рівнів, та правильного функціонування кнопок (рис. 4.68).

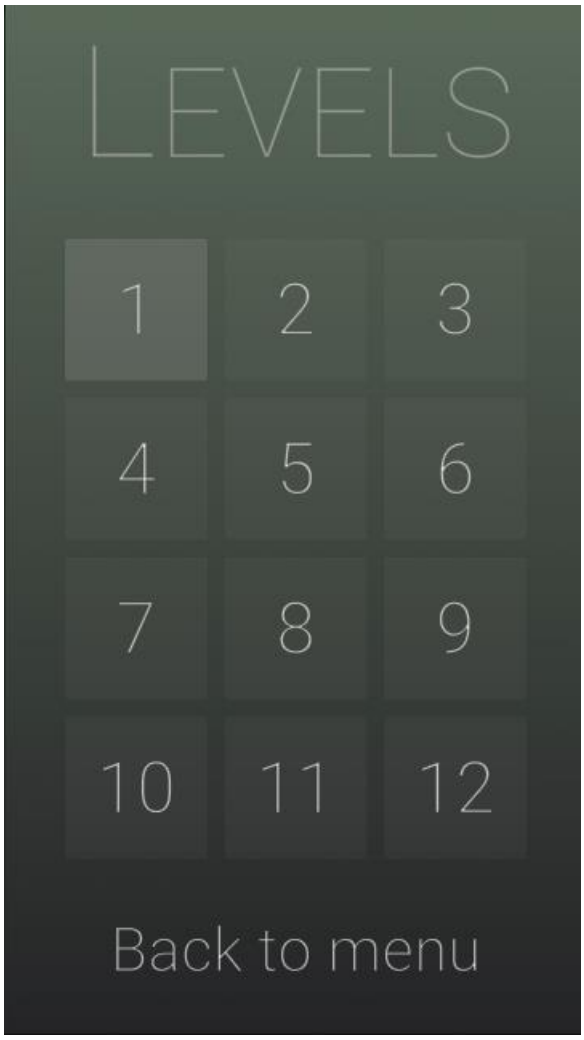

Рисунок 4.68 – Меню вибору ігрового рівня

Наступний пункт меню – меню результатів проходження. Було протестоване правильно відображення результатів, та коректне функціювання кнопок (рис. 4.69).

Рисунок 4.69 – Меню перегляду результатів проходження

Після повного тесту інтерфейсів головного меню були протестовані інтерфейси ігрового меню, та ігрові рівні взагалі.

Першим був протестований інтерфейс ігрового рівню, який включає в себе кнопку паузи та текст що відображає кількість доступних обертів (рис. 4.70).

Разом із інтерфейсом було протестоване правильне відображення спрайтів на екрані, та коректне відображення ефектів пост-обробки (світіння, освітлення, шейдери).

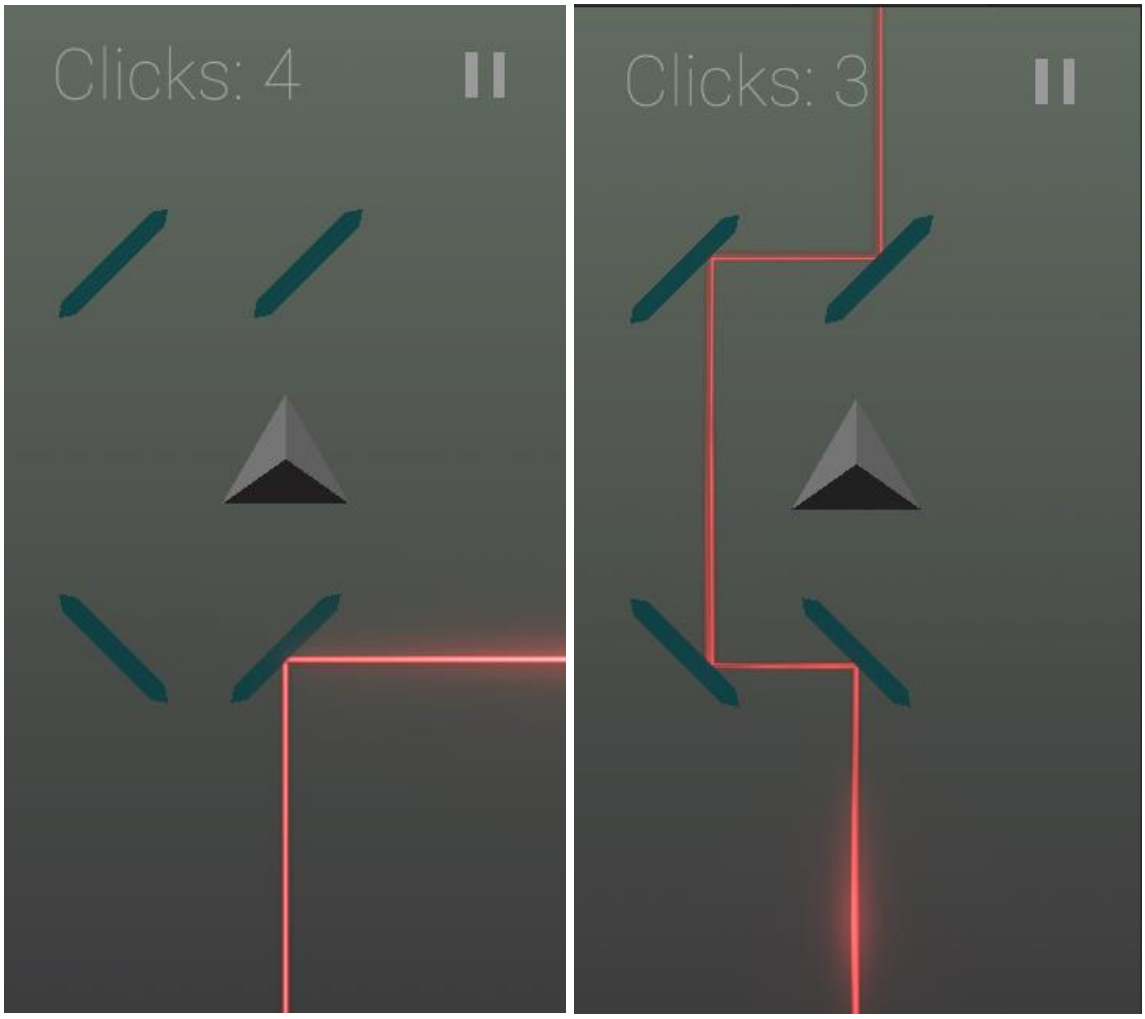

Рисунок 4.70 – Інтерфейс ігрового рівня

Далі був протестований інтерфейс меню паузи, його анімацію та коректне функціонування кнопок (рис. 4.71).

Також було протестована поведінка додатку у момент виклику меню паузи. Тест показав, що ігровий час зупиняється коли меню паузи на екрані, що повністю відповідає задумці.

Рисунок 4.71 – Інтерфейс ігрового рівня

Далі було протестоване меню невдалого проходження рівня, правильність його відображення, анімація, та коректне функціонування кнопок (рис. 4.72).

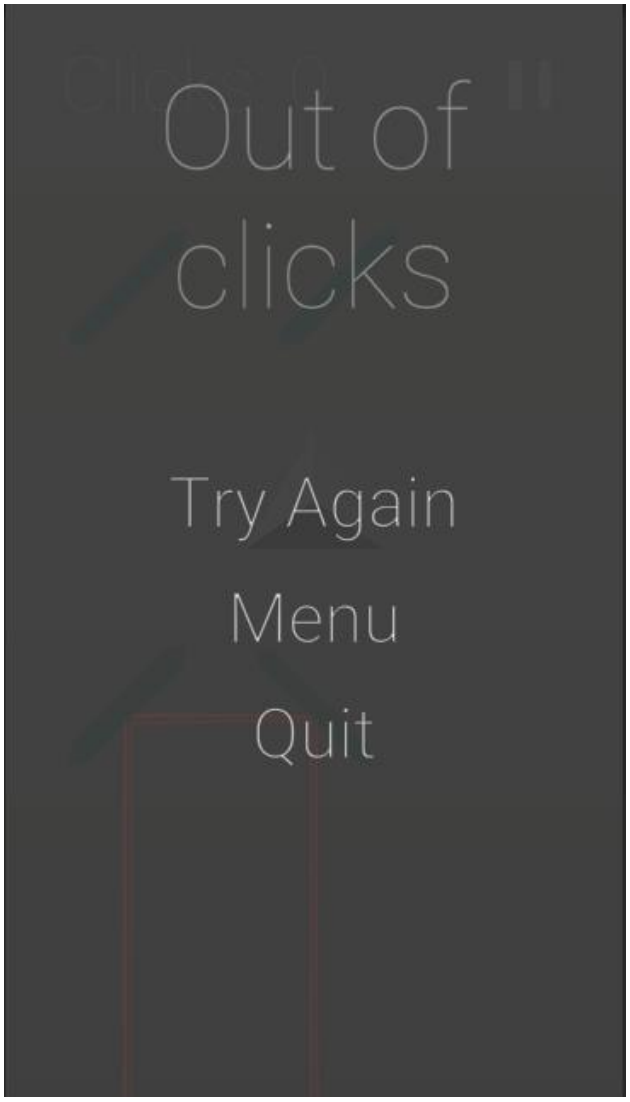

Рисунок 4.72 – Інтерфейс меню невдалого проходження рівня

Далі було протестоване меню вдалого проходження рівня, правильність його відображення, анімація, та коректне функціонування кнопок (рис. 4.73).

Разом з цим був протестований правильний виклик цього меню, та правильне відображення поточного рахунку за проходження рівня.

Рисунок 4.73 – Інтерфейс меню вдалого проходження рівня

Після того як усі інтерфейси були протестовані, можна перейти до тестування основної ігрової механіки, та тесту зберігання результатів проходження рівнів.

На рис. 4.74, рис. 4.75 та рис. 4.76 представлені результати тестування процесу проходження першого ігрового рівня.

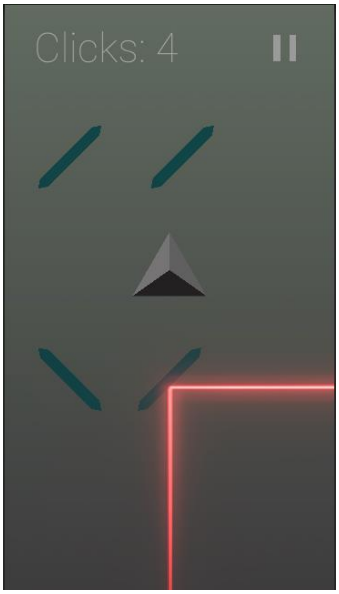

Рисунок 4.74 – Проходження першого рівня гри

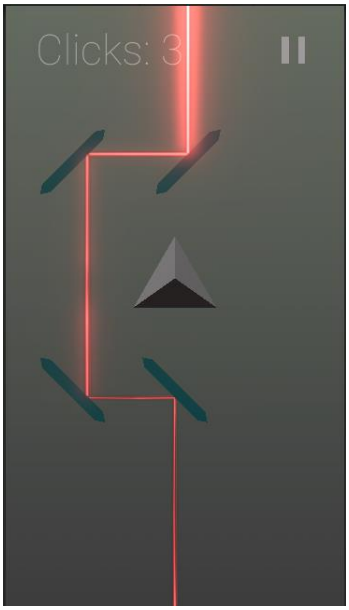

Рисунок 4.75 – Проходження першого рівня гри

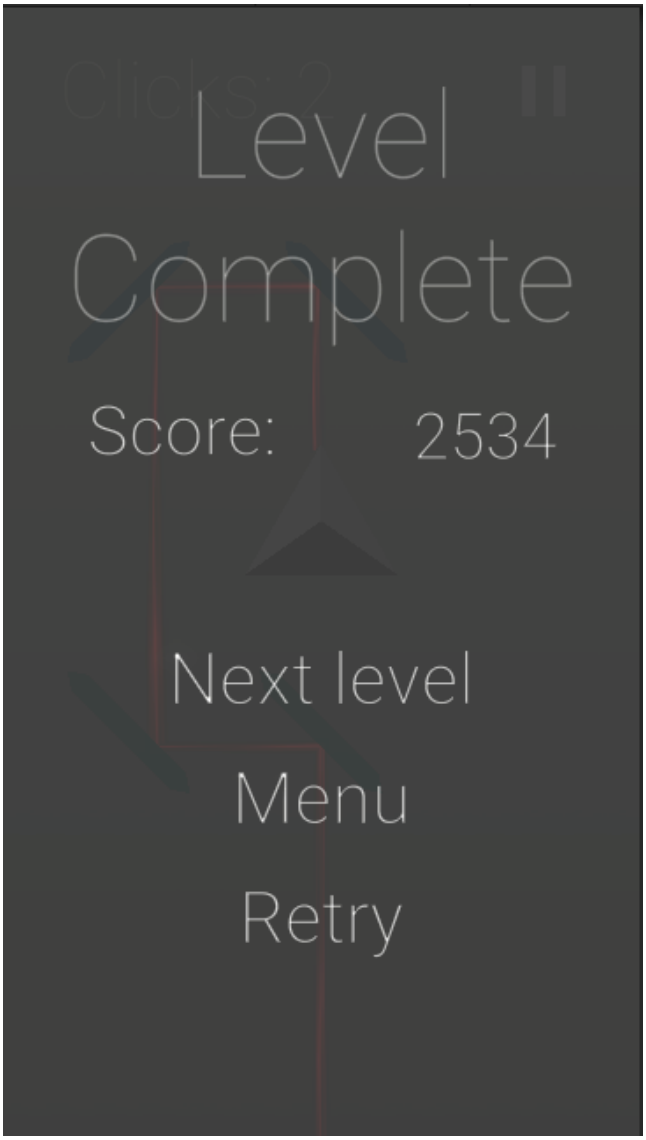

Рисунок 4.76 – Результат проходження першого рівня гри

Останнім етапом тестування є перевірка зберігання прогресу проходження гри. Перевірка була розбита на два умовні етапи: перевірка збереження результатів проходження у меню із результатами (рис. 4.77), та перевірка збереження прогресу проходження рівнів у меню із вибором ігрового рівня (рис. 4.78).

Рисунок 4.77 – Результат проходження першого рівня гри у меню результатів

Отриманий результат проходження ігрового рівня був збережений і відображається коректно у меню результатів.

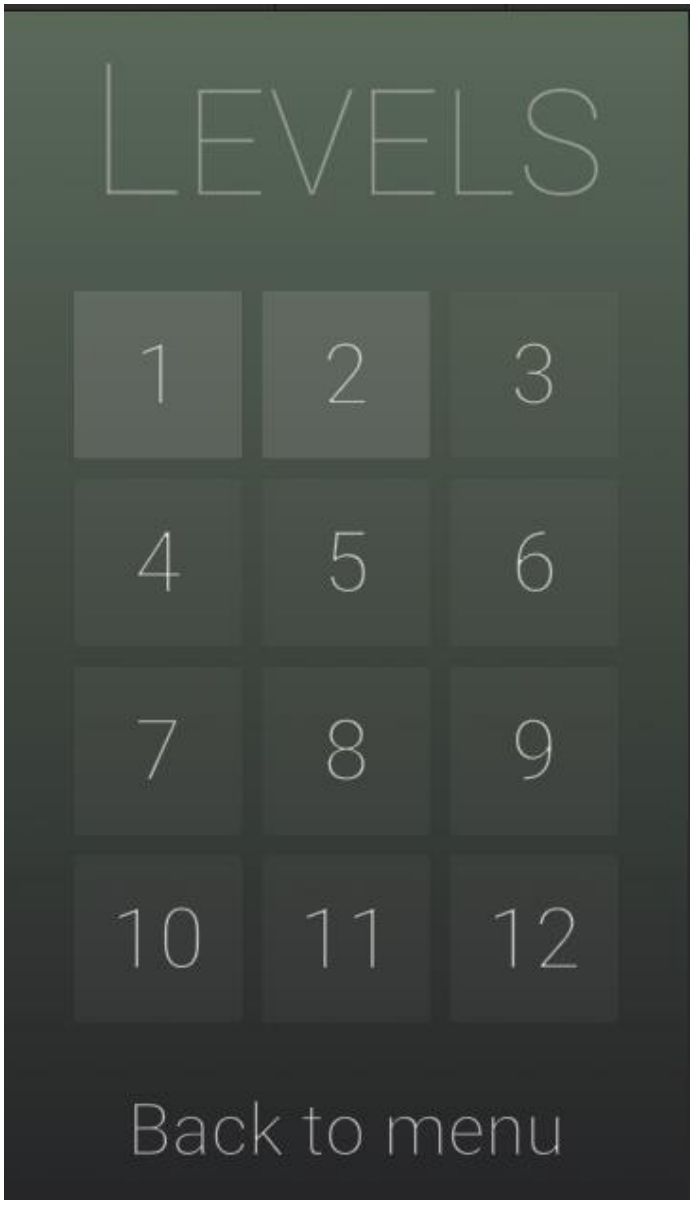

Рисунок 4.78 – Меню вибору ігрового рівня з розблокованим другим рівнем

Після успішного проходження першого рівня гравець отримав доступ до другого.

Під час тестування додатка усі анімації, функціональні кнопки, механіки спрацювали коректно та без збоїв.

ВИСНОВКИ

Гіпер-казуальний жанр захопив магазини цифрової дистриб’юції. Даний жанр включає в себе велику кількість механік, які були розглянуті під час дипломного проектування, але найбільший інтерес викликала саме механіка головоломка.

Було виявлено, що вона є найкориснішою, але найменш популярною ігровою механікою.

Під час виконання роботи були проаналізовані предметна область та основні ігрові гіпер-казуальні механіки та проекти. На основі їх аналізу були ви-значені мета та задачі дипломного проектування, що включали перелік функціональних вимог, яким має відповідати розроблюваний ігровий додаток. Було проведено дослідження інструментів та засобів реалізації, в результаті чого був обраний основний інструментарій, необхідний для реалізації проекту.

Під час планування ІТ-проекту були побудовані діаграми структуризації та організації робіт, календарного планування та ідентифіковані ризики.

У ході виконання етапу проектування гіпер-казуального додатку були розроблені діаграма прецедентів та структурно-функціональна діаграма у нотації IDEF0 типу TO-BE.

У результаті аналізу існуючих ігор був розроблений прототип, з урахуванням бажаної механіки, на базі якого був створений ігровий мобільний додаток «Laser Beam» із використанням механіки типу головоломка.

Задля досягнення мети у проекті були розроблені ігрові механіки, спрайти, шейдери, створені ігрові інтерфейси та ігрові анімації, також були розроблені ігрові рівні. Для зберігання прогресу проходження гри був використаний інструмент ігрового рушія PlayerPrefs. Протестований ігровий додаток був скомпілюваний в файл з розширенням apk.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Digi-capital projects worldwide game revenues to hit $200b by 2023: веб-сайт. URL:https://www.gamesindustry.biz/articles/2019-03-22-digi-capital-projects-worldwide-game-revenues-to-hit-usd200b-by-2023 (дата звернення 20.10.20).

2. Press release: Voodoo welcomes goldman sachs as new shareholder (pdf): звіт. Voodoo. 2017. 2 с. URL: https://uploads.strikinglycdn.com/files/a7ad09f5-4a58-4a3a-87fa-57ae2b601288/20180528%20Voodoo-GS%20MBD%20-%20ENGLISH.pdf (дата звернення 23.10.20)

3. Hyper-casual in 2019 "may be even more complicated to succeed without the product and marketing help from expert publishers": веб-сайт. URL: https://www.pocketgamer.biz/news/69881/voodoo-on-hyper-casual-it-may-be-even-more-complicated/ (дата звернення 18.09.20).

4. Game design: next level: підручник. Gingko Press Inc., 2019. 240 с.

5. Sharon Boller, Karl M. Kapp. Play to learn: everything you need to know about designing effective learning games: підручник. Association for talent development, 2017. 168 с.

6. L. Bertolini. Hands-on game development without coding: create 2d and 3d games with visual scripting in unity: підручник. Packt publishing, 2018. 430 с.

7. Top 10 game mechanics for hyper casual games: веб-сайт. URL: https://mobilefreetoplay.com/top-10-game-mechanics-for-hyper-casual-games/ (дата звернення 18.09.20).

8. C. Hodent. The gamer's brain: how neuroscience and ux can impact video game design: підручник. Crc Press; 1st Edition, 2017. 272 с.

9. C. W. Totten. Level design: processes and experiences: підручник. A K Peters/CRC Press; 1st Edition, 2016. 408 с.

10. Hyper-casual games: mobile gaming’s greatest genre: веб-сайт. URL: https://clevertap.com/blog/hyper-casual-games/ (дата звернення 23.12.18).

11. G. Kalmpourtzis. ducational game design fundamentals: a journey to creating intrinsically motivating learning experiences: підручник. A K Peters/CRC Press; 1st Edition, 2018. 360 с.

12. J. Schreier. Blood, sweat, and pixels: the triumphant, turbulent stories behind how video games are made: підручник. Harper Paperbacks, 2017. 304 с.

13. G. Barrett. Board game design advice: from the best in the world: підручник. CreateSpace independent publishing platform; 1st Edition, 2018. 258 с.

14. D. L. Daglow, R. Ismail. Indie games: from dream to delivery: підручник. Sausalito Media LLC; 1st Edition, 2018. 577 с.

15. N. Lovell. The pyramid of game design: designing, producing and launching service games: підручник. A K Peters/CRC Press; 1st Edition, 2018. 340 с.

16. G. Engelstein, I. Shalev. Building blocks of tabletop game design: an encyclopedia of mechanisms: підручник. CRC Press; 1st Edition, 2019. 516 с.

17. M. F. Atkins. Design a game (rookie get ready to code): підручник. Children's press; Illustrated edition, 2019. 32 с.

18. Defense acquisition university press. systems engineering fundamentals: підручник. Вірджинія, 2001. 222 с. URL: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf (дата звернення 11.04.19).

19. G. Booch, J. Rumbaugh, I. Jacobson. Unified modeling language user guide, The 2nd edition: підручник. Addison-Wesley, 2005. 496 с.

20. I. Sommerville. Software engineering, 8th edition: підручник. Pearson Education, 2007. 843 с.

21. Unreal engine 4 game development essentials: підручник. Бріменхем, 2016. 266 с.

22. B. Sewell. Blueprints visual scripting for unreal engine: підручник. Packt Publishing, 2015. 190 с.

23. H. Ferrone. Learning C# by developing games with unity 2019: code in C# and build 3D games with unity, 4th edition: підручник. Packt Publishing, 2019. 342 с.

24. Timothy A. Budd. C++ for java programmers: підручник. Oregon State University, Corvallis, Oregon, 1998. 29 с. URL: http://web.engr.oregonstate.edu/~budd/Books/cforj/info/preface.pdf (дата звернення 23.12.18).

25. C # /. NET history lesson. james kovac's weblog*:* веб-сайт. URL: http://jameskovacs.com/2007/09/07/cnet-history-lesson/ (дата звернення 04.02.19).

26. What are the best C# IDEs? Slant: веб-сайт. URL: https://www.slant.co/topics/4118/viewpoints/2/~c-ides~visual-studio (дата звернення 11.04.19).

27. Microsoft launches visual studio code, a free cross-platform code editor for OS X, Linux and Windows*:* веб-сайт. URL: https://techcrunch.com/2015/04/29/microsoft-shocks-the-world-with-visual-studio-code-a-free-code-editor-for-os-x-linux-and-windows/ (дата звернення 04.02.19).

28. Creativity for all*:* веб-сайт. URL: https://www.adobe.com/creativecloud.html (дата звернення 04.02.19).

29. IDEF0 – Part 1 (understanding it). Syque: веб-сайт. URL: http://www.syque.com/quality_tools/tools/Tools19.htm (дата звернення 19.04.19).

30. S. Vestnik. Functional and information modeling of production using IDEF methods: наукова стаття. Journal of mechanical engineering 55, 2009. URL: https://www.researchgate.net/publication/235636672_Functional_and_information_modeling_of_production_using_IDEF_methods (дата звернення 19.10.20).

31. Основи роботи з пакетом BPWin. Студопедія: веб-сайт. URL: https://studopedia.org/10-17823.html (дата звернення 14.05.19).

32. J. Freund, B. Rücker. Real-Life BPMN (4th edition): Includes an introduction to DMN: підручник. Independently published, 2019. 337 с.

33. Fundamental approaches to software engineering: наукова стаття. 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings

34. Г. Буч, Д. Рамбо, И. Якобсон. Краткая история UML // Язык UML. Руководство пользователя: підручник. ДМК Пресс, 2006. 496 с.

35. E. Walters. Using UML activities to model business processes: A handbook for practitioners: підручник. Independently published, 2019. 100 с.

36. K. Nyisztor, M. Nyisztor. UML and object-oriented design foundations: understanding object-oriented programming and the unified modeling language (professional skills): підручник. Independently published, 2018. 122 с.

37. K. Nyisztor. Software development from a to z: learn about oop, uml, agile, kanban, scrum and so much more! get insights into the software development industry: підручник. Independently published, 2018. 115 с.

38. E. Mach. Object oriented analysis & design cookbook: introduction to practical system modeling: підручник. Independently published, 2019. 207 с.

ДОДАТОК А. ПЛАНУВАННЯ РОБІТ

А.1 Деталізація мети проекту методом SMART

Для визначення мети проекту на основі певних показників та вимірів використаємо метод SMART, що з назви як з абревіатури визначає показники постановки: Specific (конкретна), Measurable (вимірювана), Achievable (досяжна), Relevant (реалістична), Time-framed (обмежена у часі). Результати деталізації методом SMART розміщені у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Розробити ігровий мобільний додаток «Laser Beam» для цікавого та корисного проводження вільного часу
Measurable (вимірювана)	Розробити ігровий мобільний додаток «Laser Beam» за сімдесят днів
Achievable (досяжна)	Розробити ігровий мобільний додаток «Laser Beam» на основі доступних функціональних можливостей
Relevant (реалістична)	Розробити ігровий мобільний додаток «Laser Beam» на основі існуючих ігрових гіпер-казуальних механік
Time-framed (обмежена у часі)	Розробити ігровий мобільний додаток «Laser Beam» на основі сформованого календарного плану

А.2 Планування змісту структури робіт ІТ-проекту (WBS)

Для подальшої побудови WBS таблиці необхідно визначити перелік робіт для декомпозиції проекту розробки мобільного ігрового додатку «Laser Beam». На основі визначених рівнів майбутнього проекту будуємо таблицю WBS (рис. А.1)

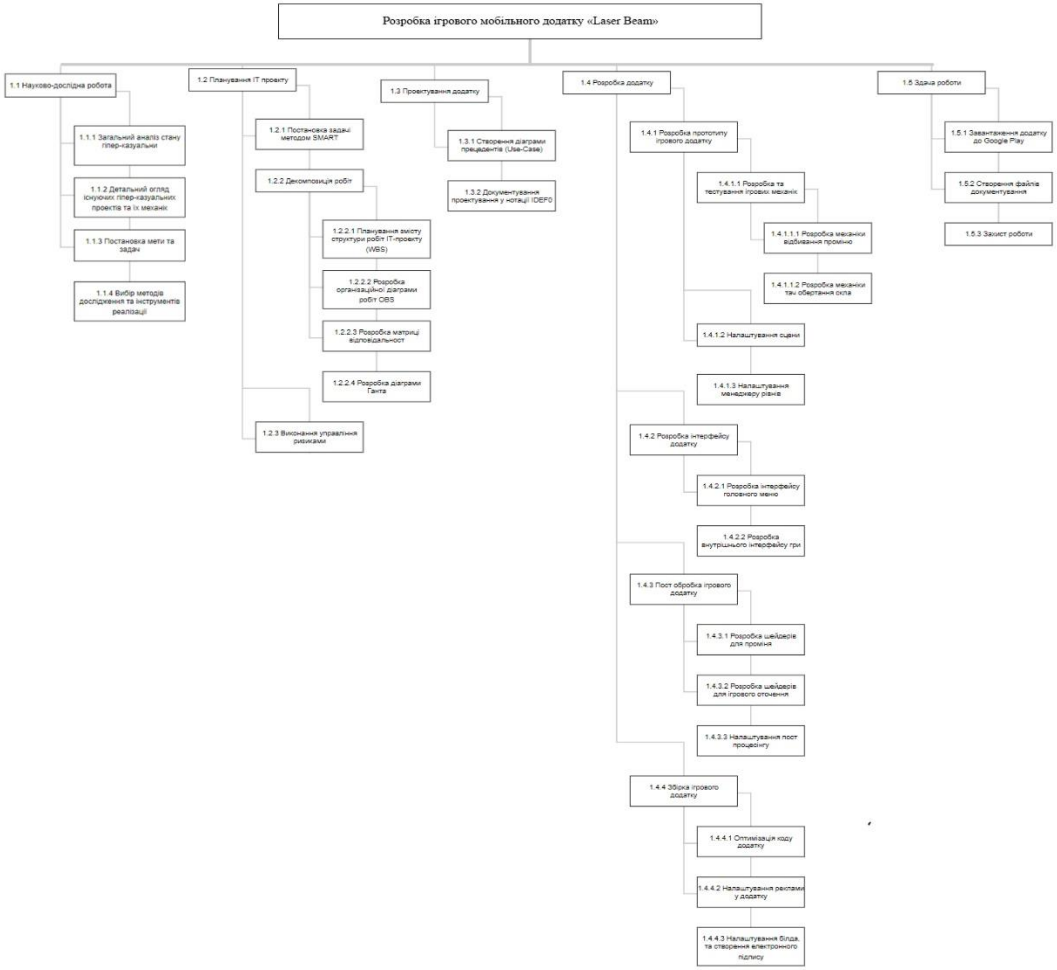

Рисунок А.1 - Діаграма декомпозиції робіт проекту (WBS)

А.3 Організаційна структура проекту (OBS)

Мета організаційної структури проекту – визначити виконавців, відповідальних за виконання робіт, тобто визначити ступінь участі різних працівників в реалізації рішення.

Визначимо виконавців робіт та інших зацікавлених сторін проекту:

1. Виконавець (Хвайра Т.С.Т.).

2. Науковий керівник (Федотова Н.А.).

3. Завідувач секції (Шендрик В.В.).

Після визначення зацікавлених сторін можна переходити то розробки структури. OBS-структура організації робіт проекту можна побачити на рис. А.2.

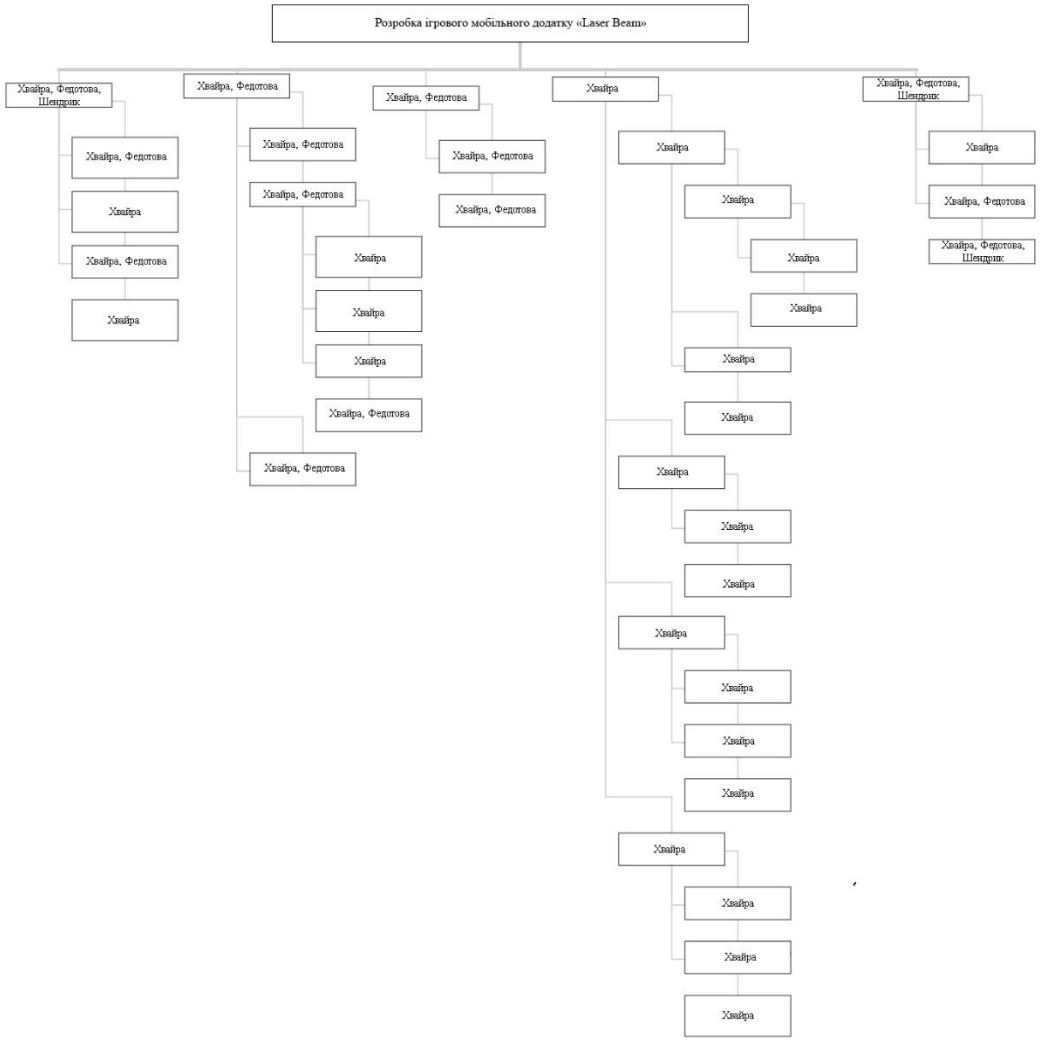

Рисунок А.2 – Діаграма організації робіт проекту (OBS)

На основі створених WBS та OBS структур робіт можна побудувати матрицю відповідальності. Вона закріплює за кожною елементарною роботою з ієрархії певного виконавця (зацікавлену сторону), забезпечуючи опис і узгодження структури відповідальності за виконання пакетів робіт. Матрицю відповідальності проекту наведено у таблиці А.2.

Таблиця А.2 – Матриця відповідальності

WBS\OBS	Хвайра Т.С.Т.	Федотова Н.А.	Шендрик В.В.
1 Розробка мобільного ігрового додатку "Laser Beam"			
1.1 Науково-дослідна робота			
1.1.1 Загальний аналіз стану гіпер-казуальних ігор у світі			
1.1.2 Детальний огляд існуючих гіпер-казуальних проектів та їх механік			
1.1.3 Постановка мети та задач			
1.1.4 Вибір методів дослідження та інструментів реалізації			
1.2 Планування IT проекту			
1.2.1 Постановка задачі методом SMART			
1.2.2 Декомпозиція робіт			
1.2.2.1 Планування змісту структури робіт IT-проекту (WBS)			
1.2.2.2 Розробка організаційної діаграми робіт OBS			

Продовження таблиці А.2 – Матриця відповідальності

WBS\OBS	Хвайра Т.С.Т.	Федотова Н.А.	Шендрик В.В.
1.2.2.3 Розробка матриці відповідальності			
1.2.2.4 Розробка діаграми Ганта			
1.2.3 Виконання управління ризиками			
1.3 Проектування додатку			
1.3.1 Створення діаграми прецедентів (Use-Case)			
1.3.2 Документування проектування у нотації IDEF0			
1.4 Розробка додатку			
1.4.1 Розробка прототипу ігрового додатку			
1.4.1.1 Розробка та тестування ігрових механік			
1.4.1.1.1 Розробка механіки відбивання променю			
1.4.1.1.2 Розробка механіки тач обертання скла			
1.4.1.2 Налаштування сцени			
1.4.1.3 Налаштування менеджеру рівнів			
1.4.2 Розробка інтерфейсу додатку			
1.4.2.1 Розробка інтерфейсу головного меню			
1.4.2.2 Розробка внутрішнього інтерфейсу гри			

Продовження таблиці А.2 – Матриця відповідальності

WBS\OBS	Хвайра Т.С.Т.	Федотова Н.А.	Шендрик В.В.
1.4.3 Пост обробка ігрового додатку			
1.4.3.1 Розробка шейдерів для проміня			
1.4.3.2 Розробка шейдерів для ігрового оточення			
1.4.3.3 Налаштування пост процесінгу			
1.4.4 Збірка ігрового додатку			
1.4.4.1 Оптимізація коду додатку			
1.4.4.2 Налаштування реклами у додатку			
1.4.4.3 Налаштування збірки, та створення електронного підпису			
1.5 Здача роботи			
1.5.1 Завантаження додатку до Google Play			
1.5.2 Створення файлів документування			
1.5.3 Захист роботи			

А.4 Побудова календарного графіка виконання ІТ-проекту

Для того, щоб мати реальне уявлення про тривалість виконання робіт з урахуванням обмеженості у використанні ресурсів, на підставі часткових

мережевих моделей, а також, проекту в цілому з урахуванням вихідних і святкових днів, будують календарний графік робіт.

Він є реальним розподілом робіт по пакету по календарними датами, тобто своєрідним розкладом виконання робіт. Діаграма Ганта є досить зручним для користування. Діаграма Ганта представляє собою відрізки, які розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, як складова плану, розміщуються по вертикалі. Початок, кінець і довжина відрізків на шкалі часу відповідають початку, кінця і тривалості завдання.

За допомогою програми Microsoft Project Professional побудовано діаграму Ганта (рис. А.3, рис. А.4).

	1 Розробка мобільного ігрового додатку "Laser Beam"	**70 days**	**Mon 9/7/20**	**Fri 12/11/20**	
	1.1 Науково-дослідна робота	**40 days**	**Mon 9/7/20**	**Fri 10/30/20**	
	1.1.1 Загальний аналіз стану гіпер-казуальних ігор у світі	10 days	Mon 9/7/20	Fri 9/18/20	
	1.1.2 Детальний огляд існуючих гіпер-казуальних проектів та їх механік	10 days	Fri 9/18/20	Thu 10/1/20	3
	1.1.3 Постановка мети та задач	10 days	Thu 10/1/20	Wed 10/14/20	4
	1.1.4 Вибір методів дослідження та інструментів реалізації	10 days	Wed 10/14/20	Tue 10/27/20	5
	1.2 Планування IT проекту	**12 days**	**Mon 11/2/20**	**Tue 11/17/20**	
	1.2.1 Постановка задачі методом SMART	1 day	Mon 11/2/20	Mon 11/2/20	6
	1.2.2 Декомпозиція робіт	**9 days**	**Tue 11/3/20**	**Fri 11/13/20**	
	1.2.2.1 Планування змісту структури робіт IT-проекту (WBS)	1 day	Tue 11/3/20	Tue 11/3/20	8
	1.2.2.2 Розробка організаційної діаграми робіт OBS	1 day	Wed 11/4/20	Wed 11/4/20	10
	1.2.2.3 Розробка матриці відповідальності	1 day	Thu 11/5/20	Thu 11/5/20	11
	1.2.2.4 Розробка діаграми Ганта	3 days	Fri 11/6/20	Tue 11/10/20	12
	1.2.3 Виконання управління ризиками	3 days	Thu 11/12/20	Mon 11/16/20	13
	1.3 Проектування додатку	**4 days**	**Tue 11/17/20**	**Fri 11/20/20**	
	1.3.1 Створення діаграми прецедентів (Use-Case)	2 days	Tue 11/17/20	Wed 11/18/20	14
	1.3.2 Документування проектування у нотації IDEF0	2 days	Thu 11/19/20	Fri 11/20/20	16
	1.4 Розробка додатку	**50 days**	**Mon 9/7/20**	**Fri 11/13/20**	
	1.4.1 Розробка прототипу ігрового додатку	**15 days**	**Mon 9/7/20**	**Fri 9/25/20**	
	1.4.1.1 Розробка та тестування ігрових механік	**5 days**	**Mon 9/7/20**	**Fri 9/11/20**	

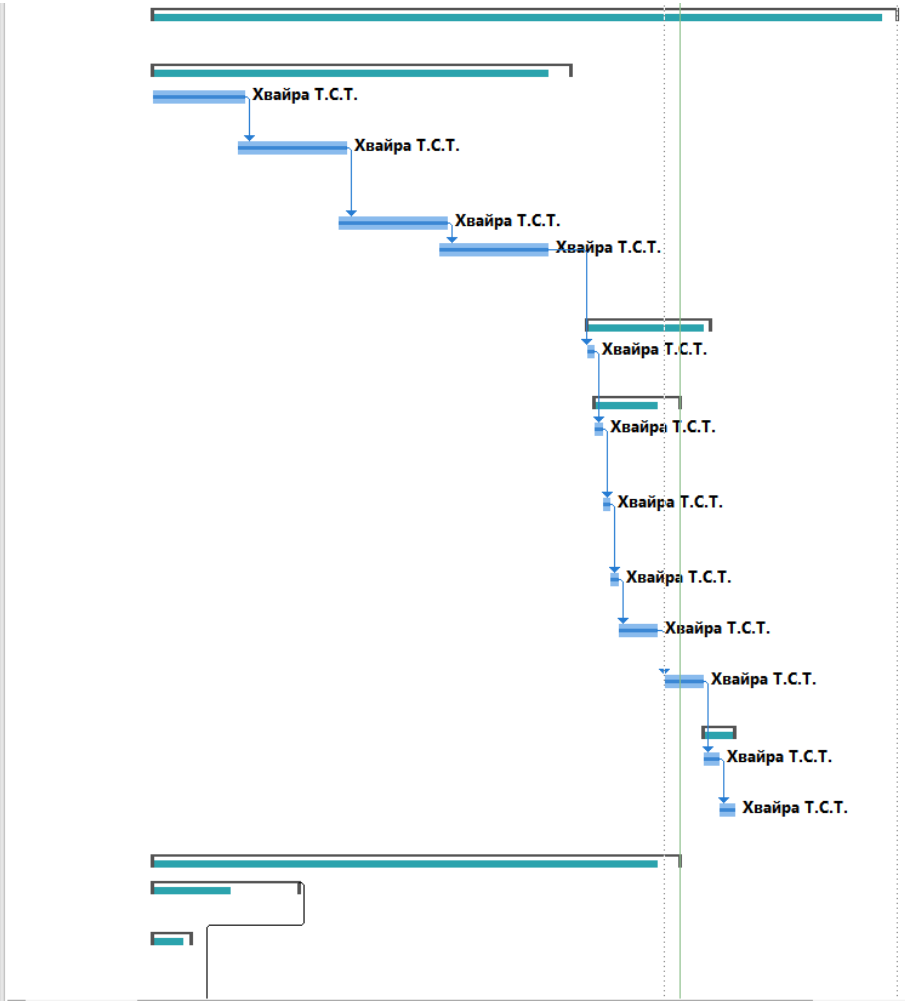

Рисунок А.3 – Календарний план та діаграма Ганта

	1.4.1 Розробка прототипу ігрового додатку	**15 days**	**Mon 9/7/20**	**Fri 9/25/20**	
	1.4.1.1 Розробка та тестування ігрових механік	**5 days**	**Mon 9/7/20**	**Fri 9/11/20**	
	1.4.1.1.1 Розробка механіки відбивання променю	2 days	Mon 9/7/20	Tue 9/8/20	
	1.4.1.1.2 Розробка механіки тач обертання скла	2 days	Wed 9/9/20	Thu 9/10/20	21
	1.4.1.2 Налаштування сцени	2 days	Fri 9/11/20	Mon 9/14/20	22
	1.4.1.3 Налаштування менеджеру рівнів	2 days	Tue 9/15/20	Wed 9/16/20	23
	1.4.2 Розробка інтерфейсу додатку	**15 days**	**Wed 9/16/20**	**Tue 10/6/20**	**19**
	1.4.2.1 Розробка інтерфейсу головного меню	5 days	Wed 9/16/20	Tue 9/22/20	24
	1.4.2.2 Розробка внутрішнього інтерфейсу гри	10 days	Wed 9/23/20	Tue 10/6/20	26
	1.4.3 Пост обробка ігрового додатку	**10 days**	**Wed 10/7/20**	**Tue 10/20/20**	**25**
	1.4.3.1 Розробка шейдерів для проміня	3 days	Wed 10/7/20	Fri 10/9/20	27
	1.4.3.2 Розробка шейдерів для ігрового оточення	3.75 days	Mon 10/12/20	Thu 10/15/20	29
	1.4.3.3 Налаштування пост процесінгу	3 days	Thu 10/15/20	Mon 10/19/20	30
	1.4.4 Збірка ігрового додатку	**16 days**	**Tue 10/20/20**	**Tue 11/10/20**	
	1.4.4.1 Оптимізація коду дода	10 days	Tue 10/20/20	Mon 11/2/20	31
	1.4.4.2 Налаштування реклами у додатку	3 days	Tue 11/3/20	Thu 11/5/20	33
	1.4.4.3 Налаштування білда, та створення електронного	2.5 days	Fri 11/6/20	Tue 11/10/20	34
	1.5 Здача роботи	**7 days**	**Tue 12/1/20**	**Wed 12/9/20**	
	1.5.1 Завантаження додатку до Google Play	1 day	Tue 12/1/20	Tue 12/1/20	
	1.5.2 Створення файлів документування	5 days	Wed 12/2/20	Tue 12/8/20	37
	1.5.3 Захист роботи	1 day	Wed 12/9/20	Wed 12/9/20	38

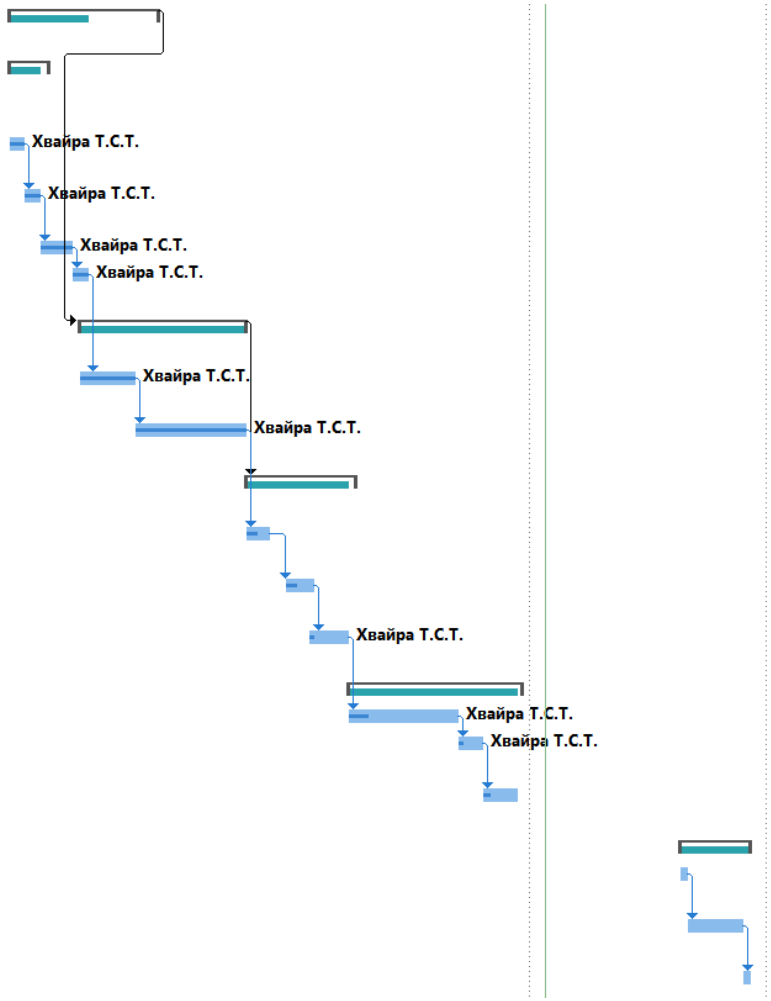

Рисунок А.4 – Календарний план та діаграма Ганта

А.5 Управління ризиками

Процес управління ризиками спрямований на виявлення та оцінку ризиків, щоб дати можливість зрозуміти ризики та ефективно управляти ними. Ключовим етапом, який пов'язує ідентифікацію / оцінку ризиків з їх управлінням, є розуміння. Побудуємо матрицю декомпозиції ризиків RBS (Risk Breakdown Structure, англ. «Структура декомпозиції ризиків») для нашого проекту (рис. А.5).

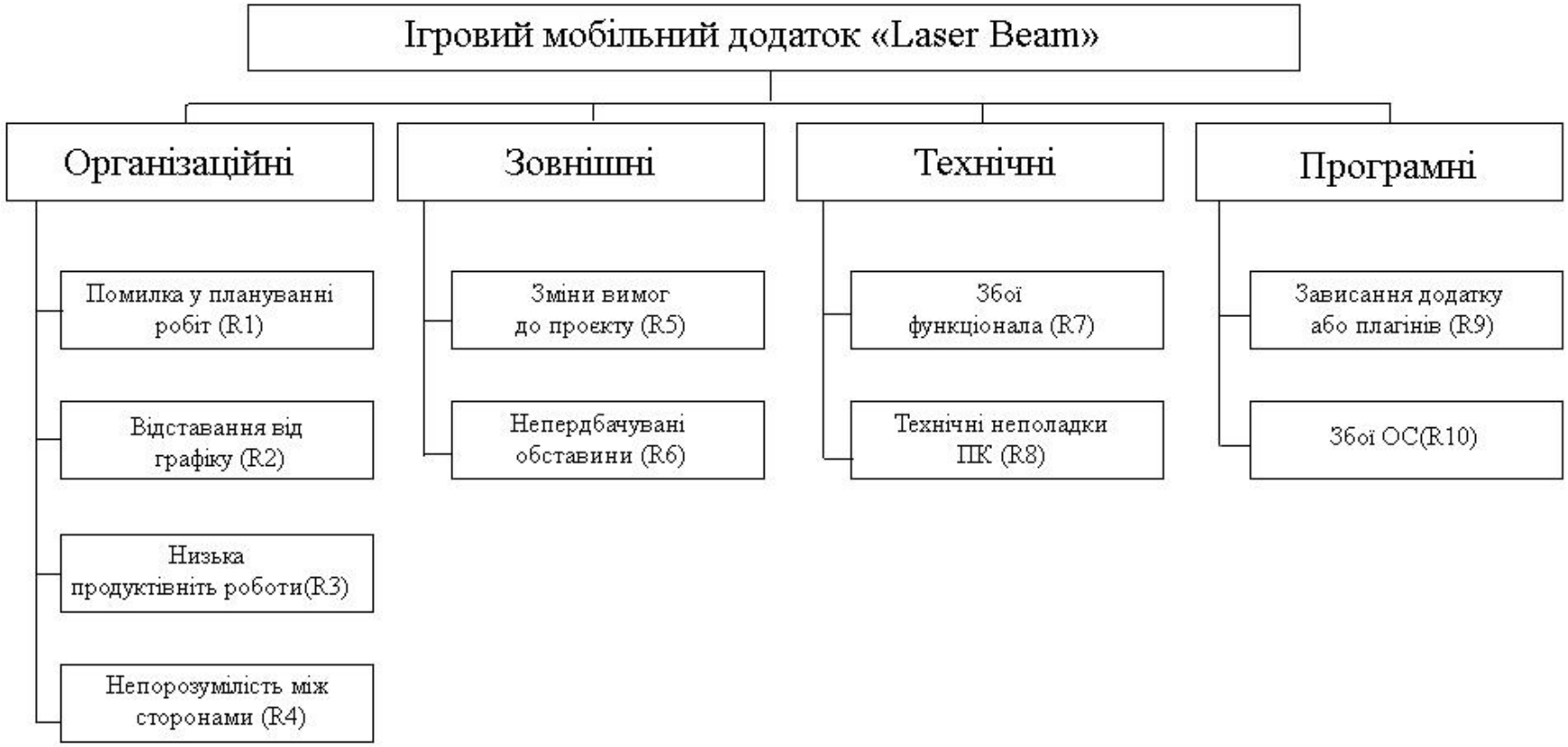

Рисунок А.5 – Матриця декомпозиції ризиків RBS

Для більш точного дослідження впливу ризиків на проект охарактеризуємо їх за показниками ймовірності виникнення (табл. А.3) та ступенями втрат (табл. А.4).

Таблиця А.3 – Ймовірність виникнення ризиків проекту

Ступінь ймовірності		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Слабо ймовірно										
2	Мало ймовірно										
3	Ймовірно										
4	Доволі ймовірно										
5	Майже ймовірно										

Таблиця А.4 – Ступінь втрат ризиків проекту

Ступінь втрат		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальний										
2	Низький										
3	Середній										
4	Високий										
5	Максимальна										

На основі показників ймовірності виникнення та ступеню втрат побудуємо матрицю «Ймовірність-втрати». За основу візьмемо значення впливу ризику R, помноживши значення ймовірності виникнення (Pq) на ступінь витрати (Iq). Відповідно за добутком визначимо ступінь впливу:

– ті, що можуть ігноруватися - $1 \leq R \leq 4$;

– незначні - $5 \leq R \leq 8$;

– помірні - $9 \leq R \leq 11$;

– суттєві - 12≤R≤17;

– критичні - 20≤R≤25;

Значення ступеню впливу та його рівень наведені у табл. А.5.

Таблиця А.5 – Класифікація ризиків за ступенем впливу

Ризик проекту	Ступінь впливу	Класифікація рівня
Помилки у плануванні робіт (R1)	6 (незначний)	Виправданий
Відставання від графіку робіт (R2)	12 (суттєвий)	Недопустимий
Низька продуктивність роботи (R3)	9 (помірний)	Виправданий
Непорозумілість між сторонами (R4)	1 (може ігноруватися)	Прийнятний
Зміна вимог до проекту (R5)	6 (незначний)	Виправданий
Непередбачувані обставини (R6)	16 (суттєвий)	Недопустимий
Збої функціонала (R7)	6 (незначний)	Виправданий
Технічні неполадки ПК (R8)	4 (той, що може ігноруватися)	Прийнятний
Зависання додатку або плагінів (R9)	25 (критичний)	Недопустимий
Збої ОС (R10)	4 (той, що може ігноруватися)	Прийнятний

Потім на основі даної таблиці побудуємо саму матрицю «Ймовірність-втрати» (табл. А.6), віднісши ризики до кольорових областей в залежності від прийнятності ризику:

– прийнятні ризики (добуток 1≤R≤4, зелений колір);

– виправданні ризики (добуток 5≤R≤11, жовтий колір);

– недопустимі ризики (добуток 12≤ R≤25, червоний колір).

Таблиця А.6 – Матриця «Ймовірність-втрати»

Максимальна(5)					R9
Висока(4)			R2	R6	
Середня (3		R1, R7	R3		
Низька(2)		R8, R10	R5		
Мінімальна(1)	R4				
	Слабо ймовірно (1)	Мало ймовірно (2)	Ймовірно (3)	Доволі ймовірно (4)	Майже ймовірно (5)

ДОДАТОК Б. ЛІСТИНГ СКРИПТІВ

Код скрипту прототипу LaserController.cs

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

[RequireComponent(typeof(LineRenderer))]
public class LaserController : MonoBehaviour
{
    const int Infinity = 999;
    int maxReflections = 100;
    int currentReflections = 0;

    [SerializeField]
    Vector2 startPoint, direction;
    List<Vector3> Points;
    int defaultRayDistance = 100;
    LineRenderer lr;

    void Start()
    {
        Points = new List<Vector3>();
        lr = transform.GetComponent<LineRenderer>();
    }

    private void Update()
    {
        var hitData = Physics2D.Raycast(startPoint, (direction -
 startPoint).normalized, defaultRayDistance);

        currentReflections = 0;
        Points.Clear();
        Points.Add(startPoint);

        if (hitData)
        {
            ReflectFurther(startPoint, hitData);
        }
        else
        {
            Points.Add(startPoint + (direction -
 startPoint).normalized * Infinity);
        }

        lr.positionCount = Points.Count;
        lr.SetPositions(Points.ToArray());
    }

    private void ReflectFurther(Vector2 origin, RaycastHit2D hitData)
    {
        if (currentReflections > maxReflections) return;
```

```
        Points.Add(hitData.point);
        currentReflections++;

        if(hitData.collider.CompareTag("Obstacle"))
        {
            Vector2 inDirection = (hitData.point - origin).normalized;
            Vector2 newDirection = Vector2.Reflect(inDirection, hitData.norma
l);

            var newHitData = Physics2D.Raycast(hitData.point + (newDirection
* 0.0001f), newDirection * 100, defaultRayDistance);
            if (newHitData)
            {
                ReflectFurther(hitData.point, newHitData);
            }
            else
            {
                Points.Add(hitData.point + newDirection * defaultRayDistance)
;
            }
        }

        else if(hitData.collider.CompareTag("Target"))
        {
            Debug.Log("End");
            //_endScreen.gameObject.SetActive(true);
        }
    }
}
```

Код скрипту clickCounter.cs

```
public class globalClickController {
    private static globalClickController _instance = null;
    public static globalClickController sharedInstance {
        get {
            if (_instance == null) {
                _instance = new globalClickController ();
            }
            return _instance;
        }
    }
    public int _clickCounter;
}
```

Код скрипту endUIScript.cs

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

public class endUIScript : MonoBehaviour
{
```

```
    public TextMeshProUGUI _currentScoreText;
    public GameObject _endUI;

    int _currentScore;

    private void Start()
    {
        _currentScore = globalClickController.sharedInstance._clickCounter *
1267;
        if(PlayerPrefs.GetInt("scoreLevel" + SceneManager.GetActiveScene().bu
ildIndex) < _currentScore)
            PlayerPrefs.SetInt("scoreLevel" + SceneManager.GetActiveScene().b
uildIndex, _currentScore);

        _currentScoreText.text = _currentScore.ToString();

    }

    public void ReloadLevel()
    {
        Animator animator = _endUI.GetComponent<Animator>();

        if(animator != null)
        {
            animator.SetBool("transition", false);
        }

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    public void BackToMain()
    {
        Animator animator = _endUI.GetComponent<Animator>();

        if(animator != null)
        {
            animator.SetBool("transition", false);
        }

        SceneManager.LoadScene(0);
    }

    public void NextLevel()
    {
        Animator animator = _endUI.GetComponent<Animator>();

        if(animator != null)
        {
            animator.SetBool("transition", false);
        }


        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

}
```

Код скрипту failUIScript.cs

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

public class failUIScript : MonoBehaviour
{

    public void ReloadLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    public void BackToMain()
    {
        SceneManager.LoadScene(0);
    }

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

Код скрипту laserController.cs

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

[RequireComponent(typeof(LineRenderer))]

public class laserController : MonoBehaviour
{
    const int _infinity = 999;
    int _maxReflections = 100;
    int _currentReflections = 0;

    public GameObject _endUI;
    public GameObject _endCanvas;
    bool _isRun;

    [SerializeField]
    Vector2 _startPoint, _direction;
    List<Vector3> _points;
    int _defaultRayDistance = 100;
    LineRenderer _lineRenderer;

    void Start()
    {
        _isRun = true;
        _points = new List<Vector3>();
        _lineRenderer = transform.GetComponent<LineRenderer>();
    }
```

```
    private void Update()
    {
        //if(!globalRotatingController.sharedInstance._isRotating)
            CountReflection();
    }

    public void CountReflection()
    {
        var hitData = Physics2D.Raycast(_startPoint, (_direction -
 _startPoint).normalized, _defaultRayDistance);

        _currentReflections = 0;
        _points.Clear();
        _points.Add(_startPoint);

        if (hitData)
        {
            ReflectFurther(_startPoint, hitData);
        }
        else
        {
            _points.Add(_startPoint + (_direction -
 _startPoint).normalized * _infinity);
        }

        _lineRenderer.positionCount = _points.Count;
        _lineRenderer.SetPositions(_points.ToArray());
    }

    void TimeStop()
    {
        Time.timeScale = 0f;
    }

    private void ReflectFurther(Vector2 origin, RaycastHit2D hitData)
    {
        if (_currentReflections > _maxReflections) return;

        _points.Add(hitData.point);
        _currentReflections++;

        if(hitData.collider.CompareTag("Obstacle"))
        {
            Vector2 inDirection = (hitData.point - origin).normalized;
            Vector2 newDirection = Vector2.Reflect(inDirection, hitData.norma
l);

            var newHitData = Physics2D.Raycast(hitData.point + (newDirection
* 0.0001f), newDirection * 100, _defaultRayDistance);
            if (newHitData)
            {
                ReflectFurther(hitData.point, newHitData);
            }
            else
```

```
            {
                _points.Add(hitData.point + newDirection * _defaultRayDistanc
e);
            }
        }

        else if(hitData.collider.CompareTag("Target"))
        {
            if(_isRun && !globalRotatingController.sharedInstance._isRotating
)
            {
                int currentLevel = SceneManager.GetActiveScene().buildIndex;

                if(currentLevel >= PlayerPrefs.GetInt("levelUnlocked"))
                {
                    //PlayerPrefs.SetInt("levelUnlocked", currentLevel + 1);
                }

                _endCanvas.SetActive(true);

                Animator animator = _endUI.GetComponent<Animator>();

                if(animator != null)
                {
                    animator.SetBool("transition", true);
                }

                Invoke("TimeStop", 1f);

                _isRun = false;
            }
        }
    }

}
```

Код скрипту levelScript.cs

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

public class levelScript : MonoBehaviour
{
    public int _clickCounter;
    public TextMeshProUGUI _clickText;

    public GameObject _failCanvas;
    public GameObject _failUI;
    public GameObject _pauseUI;

    private void Start()
```

```
    {
        if( Time.timeScale == 0f) Time.timeScale = 1f;
        globalClickController.sharedInstance._clickCounter = _clickCounter;
        _clickText.text = "Clicks: " + _clickCounter.ToString();
    }

    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            _clickText.text = "Clicks: " + globalClickController.sharedInstan
ce._clickCounter.ToString();
        }

        if(globalClickController.sharedInstance._clickCounter == 0)
        {
            _failCanvas.SetActive(true);

            Animator animator = _failUI.GetComponent<Animator>();

            if(animator != null)
            {
                animator.SetBool("failTransition", true);
            }

            Invoke("TimeStop", 2f);
        }
    }

    public void Pause()
    {
        Animator animator = _pauseUI.GetComponent<Animator>();

        if(animator != null)
        {
            animator.SetBool("pauseTransition", true);
        }

        Invoke("TimeStop", 1f);

    }

    void TimeStop()
    {
        Time.timeScale = 0f;
    }

}
```

Код скрипту levelUIScript.cs

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

```
using UnityEngine.UI;

public class levelUIScript : MonoBehaviour
{

    int _levelUnlocked;

    public Button[] _buttons;

    private void Start()
    {
        _levelUnlocked = PlayerPrefs.GetInt("levelUnlocked",1);

        for (int i = 0; i < _buttons.Length; i++)
        {
            _buttons[i].interactable = false;
        }

        for (int i = 0; i < _levelUnlocked; i++)
        {
            _buttons[i].interactable = true;
        }
    }

    public void LoadLevel(int levelIndex)
    {
        SceneManager.LoadScene(levelIndex);
    }
}
```

Код скрипту mainMenu.cs

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

public class mainMenu : MonoBehaviour
{
    void Start()
    {
        if(Time.timeScale == 0f)
            Time.timeScale = 1f;
    }
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

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

Код скрипту obstacleController.cs

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

public class obstacleController : MonoBehaviour
{

    public GameObject _obstacle;
    private Animator _animator;

    void OnMouseDown()
    {
        if(Time.timeScale == 1f){
            _animator = _obstacle.GetComponent<Animator>();

            if(_animator != null)
            {
                if(_obstacle.transform.rotation.eulerAngles.z == 0 || _obstac
le.transform.rotation.eulerAngles.z == 180)
                {
                    _animator.SetTrigger("rotation_0");
                    globalRotatingController.sharedInstance._isRotating = tru
e;
                }

                else if(_obstacle.transform.rotation.eulerAngles.z == 90)
                {
                    _animator.SetTrigger("rotation_90");
                    globalRotatingController.sharedInstance._isRotating = tru
e;
                }

                else
                {
                    _animator.SetTrigger("rotation_90");
                    _animator.SetBool("rotation", true);
                    globalRotatingController.sharedInstance._isRotating = tru
e;
                }
            }
            Invoke("SetRotatingFalse", 1f);
            globalClickController.sharedInstance._clickCounter--;
        }

    }

    void SetRotatingFalse()
    {
        globalRotatingController.sharedInstance._isRotating = false;
    }
}
```

Код скрипту pauseUIScript.cs

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

public class pauseUIScript : MonoBehaviour
{

    public GameObject _pauseUI;

    public void ResumeGame()
    {
        Animator animator = _pauseUI.GetComponent<Animator>();

        if(animator != null)
        {
            animator.SetBool("pauseTransition", false);
        }
        Time.timeScale = 1f;
    }

    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void BackToMain()
    {
        SceneManager.LoadScene(0);
    }

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

Код скрипту rotatingController.cs

```
public class globalRotatingController {
    private static globalRotatingController _instance = null;
    public static globalRotatingController sharedInstance {
        get {
            if (_instance == null) {
                _instance = new globalRotatingController ();
            }
            return _instance;
        }
    }
    public bool _isRotating;
}
```

Код скрипту scoreboardUIScript.cs

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

public class scoreboardUIScript : MonoBehaviour
{

    public TextMeshProUGUI[] _score;

    void Start()
    {
        for (int i = 0; i < _score.Length; i++)
        {
            _score[i].text =  PlayerPrefs.GetInt("scoreLevel" + (i + 1))
                .ToString();

        }

    }

}
```