МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувача кафедри

_____Є. О. Давиденко

«___»____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ІГРОВИЙ ЗАСТОСУНОК В ЖАНРІ ГОЛОВОЛОМКА

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409. 21810923

Студент _____ В. В. Пустовіт

«___»_____2022 р.

Керівник д-р. техн. наук, доцент _____ А. В. Швед

«___»_____2022 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

«___»_____2022 р.

Миколаїв 2022

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

БД - база даних

ПЗ - програмне забезпечення

ОС - операційна система

API - Application programming interface

UI - User Interface (користувальницький інтерфейс)

UML - Unified Modeling Language

ВСТУП

Мобільні ігри зараз найпопулярніше явище, але ставлення до них у людей різняться. У більшості людей існує стереотип, що ігри просто «марна трата часу», для декого це просто «розвага», а інші поринають у світ ігор і для них гра стає захопленням, відкриттям нового світу.

Інші щоб згаяти час або просто відволіктися, надають перевагу посидіти в якійсь грі, замість того щоб зробити щось корисне. І потім думають що сарно витратили свій час. Чи не так?

Мобільні чи то комп'ютерні ігри це абсолютно нормальне явище, тому що ще з дитинства у людини закладено це заняття - гратися. Завдяки іграм ти починаєш абстрактно мислити, і завдяки цьому у вас починають формуватися різні навички. Наприклад, у дитинстві збираючи лего чи конструктор, ми розвиваємо навички конструювання, під час гри "доньки-матері" ми вживаємося в інші ролі, коли граємо в шашки чи шахи, вчимося прораховувати дії. А чим складніше та цікавіше гра, тим більше людина буде намагатися її пройти і перебирати в своїй голові всі можливі варіанти проходження, таким чином тренеруючі свій мозок.

Нажаль звичайні ігри люди сприймають нормально, а от мобільні чи комп'ютерні як залежність та деградацію, але це не правильно.

Мобільні ігри – це великий прорив у світовому розвитку ігрової галузі. Зараз їх переповнює жанр гіпер-казуальних ігор – дуже прості та зрозумілі ігри, метою яких є набирання більшої кількості очок, а механіка гри заключається лиш в натисканні на екран. Такий жанр ігор очолює зараз усі чарти Google Play Market та AppStore [1].

Мета: розвиток логічного мислення та підвищення швидкості інтелектуальної реакції гравців за рахунок розв'язку складних задач під час проходження гри.

Основними результатами даної роботи є розробка ігрового застосунку в жанрі головоломка.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- розглянути основні ігрові механіки;
- проаналізувати аналоги мобільних ігор у жанрі головоломка;
- висвітлити основні проблеми та запропонувати рішення (розвиток кроків у грі);
- створити блок-схему алгоритму роботи застосунку;
- створити модель ігрового застосунку, а саме діаграми варіантів використання;
- розробити ігровий застосунок в жанрі головоломка;
- створення і розробка дизайну ігрових рівнів;
- провести тестування застосунку;
- відтворення збору інформації через парсинг тексту;
- розробити та проаналізувати стан умов праці на підприємстві, де буде розроблятись модуль.

Об’єкт: процес розробки мобільного застосунку-гри у жанрі головоломка.

Предмет: ігрові механіки та технології розробки мобільних ігор у жанрі головоломка.

Практичне значення: Покращення у гравців логічного та критичного типів мислення за допомогою мобільного ігрового застосунку у жанрі головоломка.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ РОЗРОБКИ МОБІЛЬНИХ ІГОР У ЖАНРІ ГОЛОВОЛОМКА.

1.1 Загальний аналіз жанрів мобільних ігор

Зараз, ігри відіграють значну роль у житті людей. Зараз у світі мобільних ігор жанр гіпер-казуальних ігор займає найбільший обсяг ринку. Зазвичай, люди грають у мобільні ігри аби скоротити час та просто трохи відволіктись. Заборонити людині грати в ігри, майже неможливо, а ось зробити їх корисними це реально.

З 2017 року була винайдена велика кількість ігрових механік, цей рік став розквітом гіпер-казуальних ігор. Багато ігрових механік досить цікаві для сприйняття та веселі, але на жаль, не впливають корисно на гравця [2]. Крім того, варто зазначити, що саме не корисні механіки найпопулярніші на сьогодні. Отже, потрібно змінити цей напрям та витіснити усі некорисні механіки [3].

Механіка гри для геймдизайнерів – це поле для експериментів, що постійно кидає виклики. Будучи голим набором принципів та правил, поступово геймплей обростає особливостями та унікальними рисами, які в нього впроваджують дизайнери.

Механіка є правилами, за якими працює все оточення гравця: персонажі, ігрові частинки, локації, місії, рухи та багато іншого. Розуміння механік - основна вимога для кваліфікованого геймдизайнера, оскільки без цього не вдасться створити якісний ігровий продукт. В ігровому просторі механіки комбінуються, переплітаються та накладаються один на одного. Вони можуть бути очевидними і легко розгадуватися користувачами, але зустрічаються і заплутані схеми.

До найпопулярніших механік гіпер-казуальних ігор відносяться:

- механіка спритності (Dexterity);
- механіка укладання (Stacking);
- механіка падіння (Falling);

Саме ці, вищеперечисленні механіки, не робоять мобільні ігри корисними, навпаки, затупляють розвиток гравця, через це необхідно збільшити відсоток ігор із механікою головоломка [4].

На сьогодні, сучасне суспільство мобільне, більшість своїх справ вони вирішуються через смартфон та майже усі їх спогади та життя тісно з ним пов’язані. Отже, актуальність у створенні ігрового мобільного застосунку у жанрі головоломка за мету якого є підвищення критичного та логічного мислення є неминучою.

1.2 Детальний огляд проєктів мобільних ігор на основі часто використовуваних механік

Одні з найбільших ігрових компаній світу мобільних ігор, такі як Voodoo, Ketchapp використовують велику кількість різноманітних механік гіпер-казуальних ігор. Такі ігрові механіки розроблені пару років тому і через це вони ще досі цікаві та популярні серед користувачів [5].

Нижче розглянемо найпопулярніші серед користувачів та часто використовувані серед розробників основні та самі популярні гіпер-казуальні застосунки та механіки їх гри. Проаналізуємо усі розглянуті ігрові механіки та оберемо ті що підійдуть найбільше для розробки нашого застосунку. Особливу увагу приділимо механіки, що задовольняють мету нашого проєкту та допоможуть створити застосунок не тільки для розваг, але і для розвитку корисних навичок.

Stacking mechanics (Механіка укладання)

Механіка Stacking полягає у поступовому накоплені результату, що просуває гравця на наступний раунд. Хорошим прикладом гри з такою механікою буде гра The Tower від Ketchapp [6]. Суть цієї при полягає у складанні вежі з наданих квадратів (рис. 1.1). Після кожного вдалого ідеально рівного стеку квадратів, рівень ускладнюється і вежа починає стискатися у розмірах, тобто кожний наступний квадрат все вужче и вужче з кожним успішним стеком.

Плануючи, як правильно розробити гру з механікою укладання, важливо розуміти, що користувач повинен мати певну кількість спроб (3-8)

до того, як завершити рівень, та не забувати про складність, адже занадто легкі рівні не цікаві, тому гравці повинні мати хоча б у 20-30% неідеальних спроб. Чим менше невдалих спроб тим гра стає надто складною, и так само ідеальних спроб багато - гра стає нецікавою та простою.

Рисунок 1.1 – Приклад гри з механікою укладання

Rising / Falling mechanics (Механіка підйому / падіння)

Механіка Rising / Falling заключається у створенні цікавого сюжету, невеликої подорожі для гравця. Постійне збільшення рівня надає відчуття прогресування з монотонною механікою та жодних цілей. Для того що бгравцю було цікаво, рівень повинен самостійно розвиватись. Прикладом гри з такою механікою можна виділити Helix Jump (розробник Voodoo) (рис. 1.2). Гра полягає у розвитку прогресії, при постійному русі вниз.

Рисунок 1.2 – Приклад гри з механікою падіння

Під час проходження гри гравець думає не про точність ходу, а про те як вирішити наступний виклик та продвинутися далі. На сьогоднішній день ви гадано багато способів, як можна пройти ці рівня, але удача вам не завадить. Для гравця існує одна мета – це захистити об’єкт та досягти кінця рівня.

Для успішного проходження цієї гри, гравцю необхідно сфокусуватись на вирішенні поставлених перед ним перешкод. Важливо пам’ятати, що масштаб перешкоди, залежить від того як гравець піднімається або падає під час гри.

Growth mechanics (Механіка зростання)

Мета ігор з механікою зростання - стати найбільшим об'єктом, будь то натовп, чорна діра або липка кулька. Гравці зазвичай переміщаються навколо

свого персонажа, намагаючись знайти інший об'єкт і натрапити на нього. Потім це «вбирається» у персонажа.

Slither.io від Lowtech Studios (рис. 1.3), мабуть, найпростіша з них, де ви просто коло, що намагається зіткнутися з іншими колами. Насправді це так просто, це майже ідеальна гра. Усього дві механіки. Рухайтеся і стаєте більшими. Художній стиль настільки простий, як це можливо: кольорові краплі. І кожен раунд займає лише хвилину або близько того.

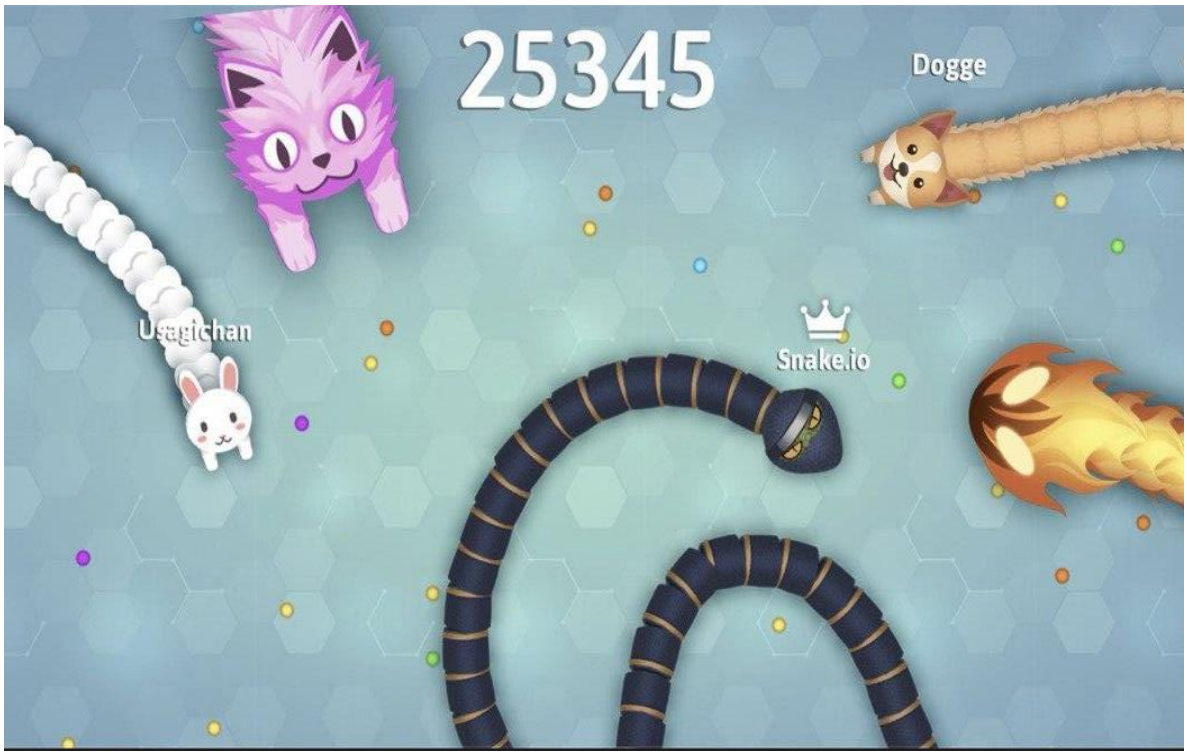

Рисунок 1.3 – Приклад гри з механікою зростання

Оскільки він потрапляє в ту золоту середину, яку шукають гіпер-казуальні ігри, він доводить, що ця механіка, мабуть, є найкращою з існуючих [8].

Puzzle mechanics (Механіка головоломки)

У кожної механіки головоломки є лише одна спільна мета: змусити гравця думати логічно. Це може бути переміщення валунів по екрану,

додавання чисел або розгадка таємниць вбивства. Ці ігри можуть бути різними, але вони часто включають переміщення об’єктів по екрану.

Звичайні головоломки з часом стають складнішими, як правило, через додавання механіки. Але в гіпер-казуальній грі ваш гравець повинен бути в змозі вирішити її за хвилину або менше.

Якісна гіпер-казуальна гра з механікою головоломка це нескінченна гра. Найкращим прикладом такої гри є 2048 від розробника Ketchapp (рис. 1.4). Правила гри дуже прості та базуються на додаванні однакових чисел, на ігровій поверхні, що розвивається під час гри [9].

Жанр головоломок – це найскладніша механіка гри у розробці, оскільки це, механіка повинна бути чітка та унікальна для розробленої гри. Тому що значно важче створити механіку, що під час проходження перетворює гру в занадто просту або складну.

Проаналізувавши вищезазначені механіки гіпер-казуальних ігор, для розробки майбутнього застосунку було обрано дві механіки – одна основна та друга допоміжна. Механіка головоломки обрана за основу, для допоміжної обрано механіку спритності. Саме ці дві механіки гарно поєднуються один з одним та допоможуть нам досягти поставленої мети.

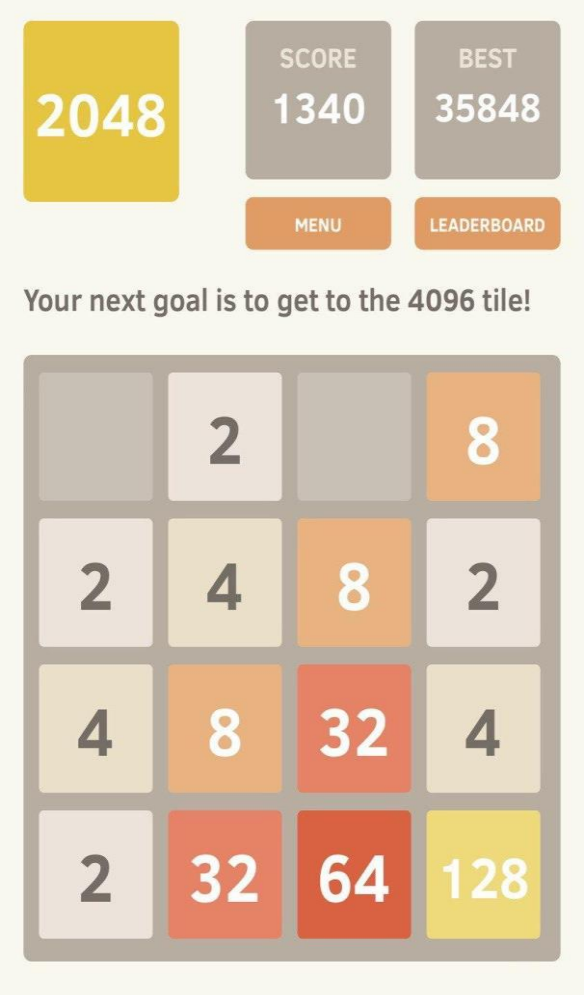

Рисунок 1.4 – Приклад гри з механікою головоломка

1.3 Аналіз ігрових застосунків у жанрі головоломка

Мета нашого мобільний застосунок – це корисне та веселе проведення часу за грою. Окрім цього, основна задача полягає у збільшенні швидкості та підвищенні реакції через додані ігрові обмеження.

Головною метою при користуванні ігровим застосунком полягає у підвищення логічного та критичного видів мислення. Крім того, не менш важливою задачею є зацікавлення гравця у проходженні у поєднанні з корисним проведенням та гаєнням часу. За статистикою, саме під час очікування люди грають у мобільні ігри [10].

Отже, для вирішення поставленої мети, нам необхідно реалізувати наступні функції:

- можливість вибору рівня гри;
- проходження попередньо обраного рівня гри;
- можливість вибору складності гри;
- зміна асистів (кольорової гами) гри;
- можливість перегляду загального рахунку за успішно пройдені рівні.

Нижче розглянемо аналоги ігор головоломок, що мають хоча б кілька вище перерахованих функцій.

Аналог 1. Laser Box – Головоломка

Laser Box – гра головоломка, що пропонує велику кількість елементів для управління лазерним променем і 120 рівнів від дуже простих до неймовірно складних (таблиця 1.1).

Таблиця 1.1 – Характеристики аналогу 1

Назва	**Laser Box - Головоломка**
Розробник	Eugene Dmitriev
Архітектура	Mobile application
Мова реалізації	Unity C#
Функції	1. Проходження рівнів 2. Переміщення елементів 3. Вирішення головоломок 4. Здійснювати покупки у додатку 5. Отримання підказок без реклами
Переваги	1. Безкоштовна 2. Не занадто складні головоломки 3. Цікавий сюжет
Недоліки	1. Складний інтерфей 2. Немає можливості обрати складність
Веб-сайт	https://apps.apple.com/ua/app/laser-box-%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BB%D0%BE%D0%BC%D0%BA%D0%B0/id942388640?l=ru

Аналог 2. Lazors

Lazors – інноваційна гра головоломка про лазери та дзеркала, що має більше 100 рівнів. (таблиця 1.2).

Таблиця 1.2 – Характеристики аналогу 2

Назва	**Lazors**
Розробник	Pyrosphere, Lda
Архітектура	Mobile application
Мова реалізації	Unity C#
Функції	1. Проходження рівнів 2. Переміщення елементів 3. Вирішення головоломок 4. Можливість перегляду загального рахунку за пройдені рівні 5. Отримання підказок, через перегляд реклами
Переваги	1. Зручний приємний інтерфейс 2. Безкоштовна 3. Дуже реалістичний інтерфейс 4. Цікавий сюжет
Недоліки	1. Вспливаюча реклама 2. Потребує підключення до інтернету
Веб-сайт	https://apps.apple.com/ua/app/lazors/id386458926?l=ru

Аналог 3. Laser Overload

Laser Overload – захоплююча гра від творців Connect. Проста але дуже гарна головоломка, в якій можна створити гарні візерунки за допомогою променів лазерів (таблиця 1.3).

Таблиця 1.3 – Характеристики аналогу 3

Назва	**Laser Overload**
Розробник	Web Avenue Unipessoal, Lda

Кінець таблиці 1.3

Архітектура	Mobile application
Мова реалізації	Unity C#
Функції	1. Проходження рівнів різної складності 2. Переміщення елементів 3. Вирішення головоломок 4. Офлайн гра (не потребує підключення Інтернету) 5. Отримання бонусів у вигляді зірок
Переваги	1. Простий і зрозумілий інтефейс 2. Безкоштовна 3. Досить творча гра 4. Не потребує підключення Інтернету
Недоліки	1. Вспливаюча реклама 2. Іноді дуже складні рівні
Веб-сайт	https://apps.apple.com/ua/app/laser-overload-%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE-%D0%BA%D0%B0%D0%B9%D1%84/id1445523670?l=ru

1.4 Специфікації вимог до програмного забезпечення.

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Даний мобільний ігровий застосунок створено для розвитку логічного мислення та підвищення швидкості інтелектуальної реакції гравців за рахунок розв’язку складних задач за час проходження гри.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення загального мобільного застосунку я та його злагодженої роботи буде використовуватися допоміжний кросплатформний ігровий рушій – Unity.

Межі проєкту ПЗ

Крайня дата завершення роботи над мобільним ігровим застосунком – 29.06.2022р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Даний мобільний ігровий застосунок не має обмежень у сферах та місцях його застосування.

Характеристики користувачів

Основні характеристики користувачів: наявність смартфону та Інтернету для завантаження застосунку (для користування Інтернет не потрібен).

Загальна структура і склад системи

Основні частини для створення програмного забезпечення: сервер; база даних; мобільний застосунок.

Загальні обмеження

Мобільний додаток розрахований лише для смартфонів.

ФУНКЦІЇ МОБІЛЬНОГО ІГРОВОГО ЗАСТОСУНКУ LASER

Функція вибору рівня

Опис функції

Функція вибору рівня допомагає користувачу обрати будь-який рівень, що для нього доступний .

Вхідна і вихідна інформація

Вхідна інформація – загальні дані про гравця;

вихідна інформація – загальні дані про доступні рівні.

Функціональні вимоги

Дані про гравця та скрипти для проходження гри.

Функція вибору складності рівня

Опис функції

Функція вибору складності рівня допомагає користувачу обрати складість рівня таку, яка йому подобається.

Вхідна і вихідна інформація

Вхідна інформація – дані про обраний рівень;

вихідна інформація – дані про складність рівню.

Функціональні вимоги

Дані про гравця та основні застосовані скрипти для проходження гри.

Функція зміни кольорової гами (налаштування рівню)

Опис функції

Функція зміни кольорової гами показує користувачу палітру кольорів для зміни гами рівня, яка буде комфортна для його очей.

Вхідна і вихідна інформація

Вхідна інформація – дані про обраний рівень та його складність;

вихідна інформація – палітра кольорів.

Функціональні вимоги

Дані про гравця та основні застосовані скрипти для проходження гри.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

В запропонованому мобільному ігровому застосунку основним джерелом вхідної інформації являється сам користувач. Він вручну авторизується, обирає рівень та робить усі наступні кроки.

Нормативно-довідкова інформація (класифікатори, довідники тощо)

Немає вимог до даного пункту.

Вимоги до способів організації, збереження та ведення інформації

Для розробки мобільного ігрового застосунку використано кросплатформний ігровий рушій Unity. Для розробки мову програмування обрано C#.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Жорстких технічних вимог немає. Комп’ютер чи ноутбук з будь-яким процесором.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура програмного забезпечення досить проста, вона складається з клієнтської частини, серверної частини та БД.

Системне програмне забезпечення

Застосунок повинен бути побудований за допомогою Unity. Для написання скриптів обрати мову C#. Для графічної частини використовувати допоміжне ПЗ Adobe Illustrator. Обмін даними повинен відбуватися використовуючи REST API. В якості бази даних для програмного забезпечення обрати PostgreSQL.

Мережне програмне забезпечення

Для створення ПЗ знадобиться будь-яка операційна система, безкоштовний редактор Visual Studio Code, крос-платформний ігровий рушій Unity; програмний додаток Adobe Illustrator.

Програмне забезпечення ведення інформаційної бази

Все ведення інформаційної бази має відбуватися через БД PostgreSQL.

Мова і технологія розробки ПЗ

Програмне забезпечення повинно бути розроблене за допомогою крос-платформи Unity. Мови розробки – C#.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Мобільний інтерфейс повинен задовольняти усім вимогам UX/UI, що дозволить користувачу затратити найменше часу на розуміння роботи системи. Шаблон ігрового застосункку повинен складатися з декількох частин, де перша частина це авторизація користувача, друга – екран відображення рівнів, третя – шаблон відображення одного з рівнів та екран після проходження гри.

Апаратний інтерфейс

Апаратний інтерфейс – це девайс користувача, яким він буде користуватися (смартфон)

Програмний інтерфейс

Unity – крос-платформний ігровий рушій.

Комунікаційний протокол

Використання мережних протоколів Wireless Application Protocol — протокол безпроводового доступу та TCP/IP.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

Будь-який користувач, який бажає користуватися програмним забезпечення, зможе його скачати.

Супроводжуваність

Супроводження – оновлення версії.

Переносимість

Мобільний застосунок підтримує як iOS так і Android.

Продуктивність

Продуктивність залежить від процесору смартфону. Продуктивність застосунка повинна контролюватися часом виконання запитів, що не повинні перевищувати 1 секунд.

Надійність

Усі дані, які вказуються користувачем повинні бути приватними і виключити можливості їх отримання будь якими способам для інших користувачів. Кожний користувач може отримати доступ до своїх даних лише авторизуватися у системі.

Безпека

Реєстрація та вхід до кабінету користувача – авторизація повинна відбуватися з токеном.

ІНШІ ВИМОГИ

Усі вимоги описані вище.

Висновки до розділу 1

Отже, в першому розділі проведено огляд існуючих мобільних ігрових застусунків з різними механіками. Розглянуті ігрові застосунки, які містять механіку головоломка. Визначені основні переваги та недоліки оглянутих мобільних ігрових застосунків. Визначено, що більшість являються комерційними розробками. Сформульовані задачі досліджень дипломної роботи та основні вимоги щодо створення програмного забезпечення.

2 МОДЕЛЮВАННЯ МОБІЛЬНОГО ІГРОВОГО ЗАСТОСУНКУ

2.1 Алгоритм роботи ігрового застосунку.

Метою мобільного застосунку, що розробляється – є корисне та веселе проведення часу за грою. Окрім цього, основна задача полягає у збільшенні швидкості та підвищенні реакції через додані ігрові обмеження.

Мобільний застосунок призначений для розвитку логічного мислення у користувача. Основні функціональні можливості застосунку:

– можливість вибору рівня гри;
– можливість вибору складності рівня;
– проходження гри;
– перегляд рахунку за пройдені ігри.

Блок-схема допоможе схематично представити процес роботи застосунку (рис. 2.1). Варто зауважити, що мобільний ігровий застосунок безкоштовній, отже кожен користувач зможе пройти гру.

Перший крок. Ініціалізація гравця у завантаженому мобільному застосунку.

Другий крок. Обираємо будь який доступний рівень гри.

Третій крок. Обираємо складність рівня.

Четвертий крок. Проходимо головоломку гри.

П'ятий крок. Переглядаємо рахунок після пройденої гри.

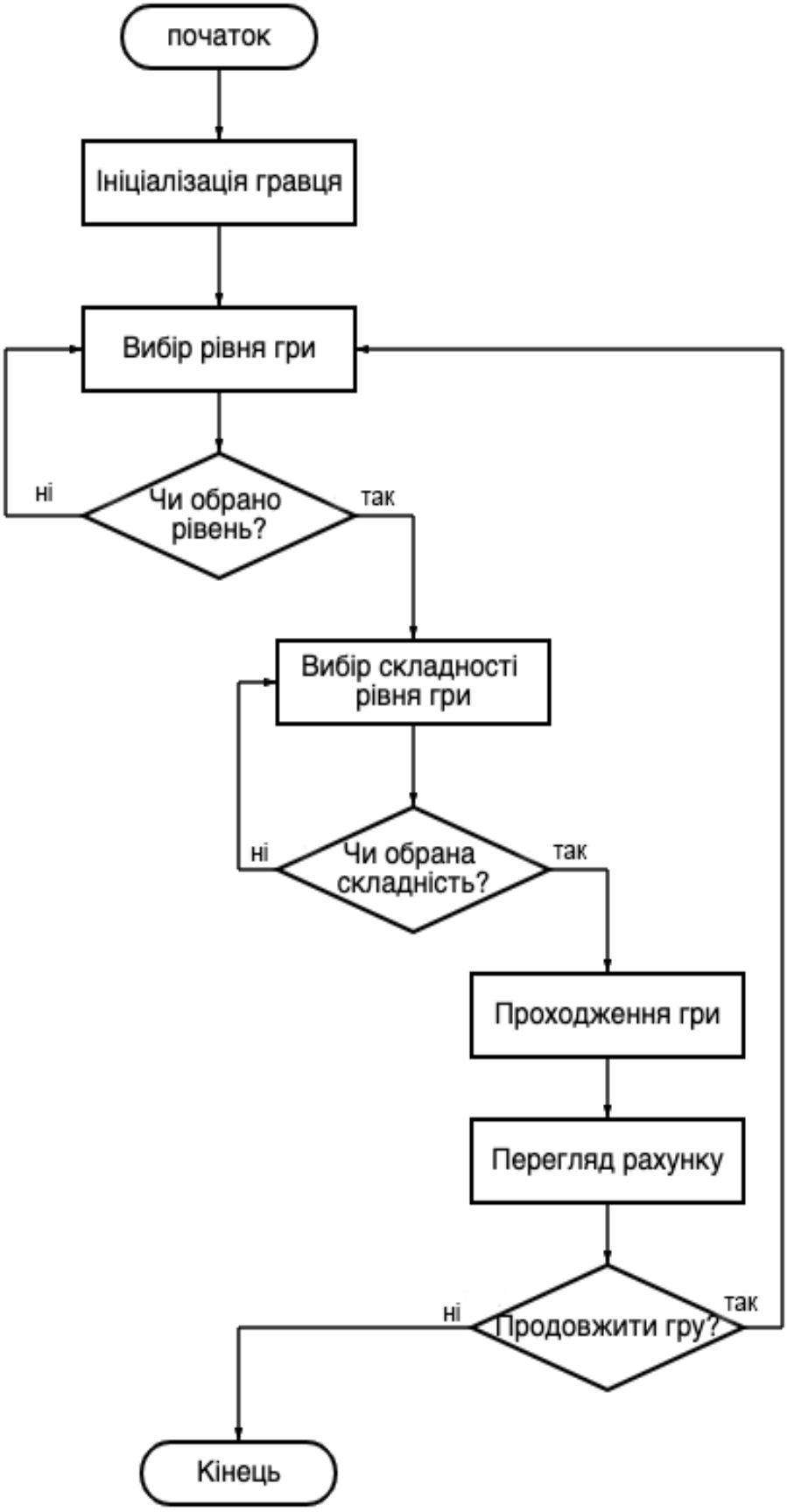

Рисунок 2.1 – Блок-схема алгоритму роботи застосунку

2.2 Діаграма прецедентів

Щоб краще зрозуміти функціонал ігрового застосунку необхідно створити модель варіантів застосування, що складається із опису акторів, усіх варіантів використання застосунку, а також взаємодії між актором та застосунком через Use Case-діаграми (діаграми прецедентів) [12].

Часто використовуваним інструментом для детального опису проєкту на сьогодні використовують мову UML, а саме Unified Modelling Language. Ця мова містить в собі велику кількість різноманітних діаграм, але нам цікава виключно діаграма прецедентів (Use Case).

На (рис. 2.2) представлена діаграма прецедентів, яка представляє собою усі можливі зв'язки акторів з усіма можливими варіантами використання.

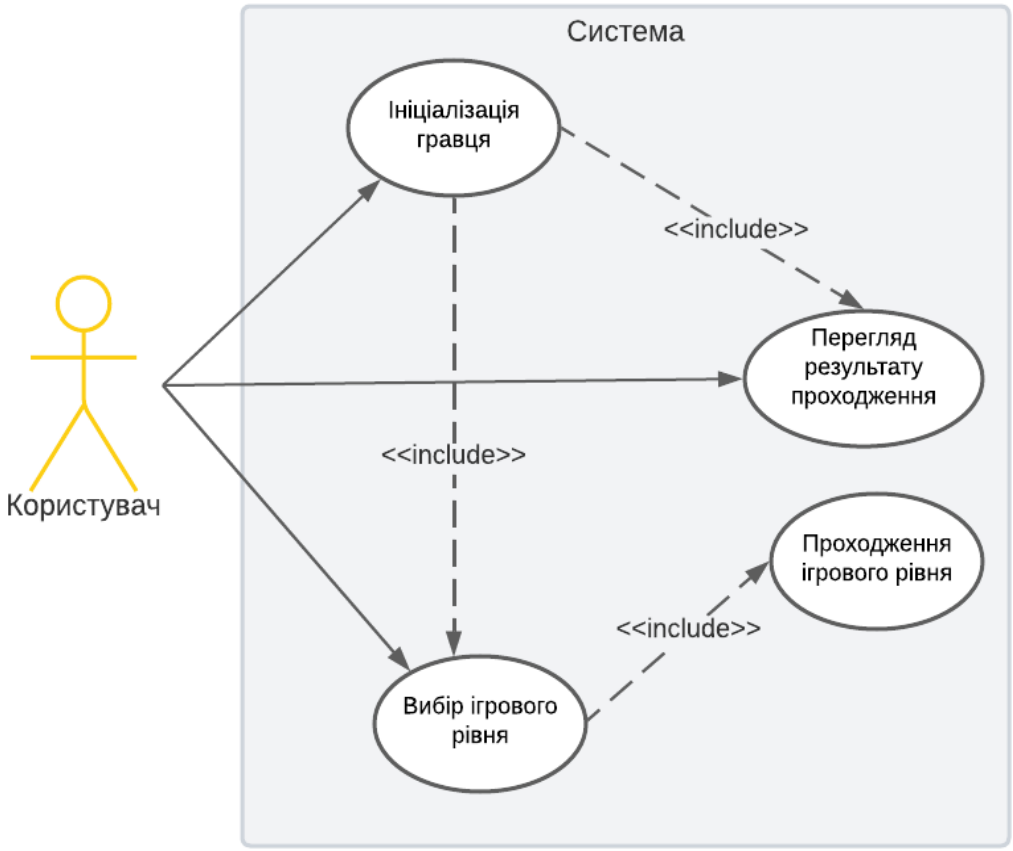

Рисунок 2.2 – UML діаграма прецендентів роботи модулю

Діаграма прецедентів проста та містить невелику кількість можливих варіантів використання. Але, якщо у діаграмі зображено більше 20 фігур, тоді ця діаграма вважається неправильно створеною. Основна мета використання діаграми це коректне відображення динамічного аспекту системи.

Щоб зібрати основні вимоги до системи, використовують діаграму варіантів використання, що включає в себе як внутрішні так і зовнішні впливи. До цих вимог, можна віднести вимоги розробки. Тобто, у той час коли систему аналізують для збору усіх можливих її функціональних можливостей, перщ зп все виділяють прецедентів та визначають усіх можливих акторів.

Щоб побудувати діаграму необхідно перш за все визначити основних діючих акторів та усі можливі варіанти використання.

Тому, у якості головного актора розробленого ігрового застосунку виділимо гравець – це актор, який використовує мобільний застосунок для проходження ігрових рівнів без можливості їх редагування.

Після цього визначимо основні прецеденти проекту:

– можливість вибору доступного ігрового рівня у меню гри;

– авторизація гравця у ігровомк застосунку отримання доступних йому рівнів та перегляду його загальних результатів проходження;

– проходження обраного ігрового рівня;

– можливість перегляду результату проходження та загального результату за усі пройдені рівні.

2.3 Діаграма розгортання

В мові UML є 12 типів діаграм:

- 4 типи діаграм представляють статичну структуру додатку;
- 5 типів представляють поведінкові аспекти системи;
- 3 представляють фізичні аспекти функціонування системи (діаграми реалізації).

Деякі з видів діаграм специфічні для певної системи і додатку. Найбільш доступними з них є:

- Діаграма прецедентів (Use-case diagram);
- Діаграма класів (Class diagram);
- Діаграма активностей (Activity diagram);
- Діаграма послідовності (Sequence diagram);
- Діаграма розгортання (Deployment diagram);
- Діаграма співробітництва (Collaboration diagram);
- Діаграма об'єктів (Object diagram);
- Діаграма станів (Statechart diagram).

Нижче представлена діаграма розгортання, що відображає графічне представлення нашого мобільного застосунку (рис. 2.3). Такий вид діаграми допомагає раціонально організувати компоненти, від цього залежить продуктивність системи та безпека.

Рисунок 2.3 – UML діаграма розгортання роботи застосунку

Детальніше розглянемо діаграму розгортання. Діаграма розгортання - це діаграма на якій розташовані обчислювальні вузли під час роботи програми, компоненти та об'єкти, котрі виконуються на цих вузлах. Діаграма розгортання в UML моделює фізичне розгортання артефактів на вузлах.

Обчислювальні вузли на діаграмі представляються у вигляді прямокутних паралелепіпедів. Також, вузли можуть містити в собі підвузли, які зображуються, як вкладені паралелепіпеди. Один вузол розгортання концептуально може представляти безліч фізичних вузлів, таких як кластер серверів баз даних.

Існує два типи вузлів: вузол пристрою та вузол середовища виконання.

Вузол пристрою - це фізичні обчислювальні ресурси, які мають свою пам'ять та сервіси для виконання ПЗ. Це може бути ПК або смартфон.

Вузол середовища виконання - це обчислювальний програмний ресурс, котрий працює всередині зовнішнього вузла та представляє собою сервіс, що виконує інші програмні елементи.

Висновки до розділу 2

Отже, в другому розділі визначено основні функціональні можливості ігрового застосунку: можливість вибору рівня гри, можливість вибору складності рівня, проходження гри та перегляд очок після проходження. Блок-схемою схематично зображено процес роботи мобільного ігрового застосунку. Для більш точної та зрозумілої роботи застосунку було створено UML діаграму прецедентів, де відображена взаємодія користувача з усіма частинами застосунку. Окрім діаграми прецендентів було представлено діаграму розгортання, що допомагає раціонально організувати компоненти, від цього залежить продуктивність системи та безпека.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ІГРОВОГО ЗАСТОСУНКУ ТА РОЗРОБКА ІНТЕРФЕЙСУ ГРИ

3.1 Проєктування ігрового інтерфейсу

Інтерфейс – це набір правил, методів та засобів взаємодії (управління, контролю) між усіма елементами системи.

Інтерфейс користувача – набір інструментів для обробки та відображення інформації, що максимально пристосований для зручності користувача [13].

Принципи, яких необхідно дотримуватись при розробці інтерфейсу:

- стандартизація (використання стандартних, перевірених інтерфейсних рішень);
- зручність та простота використання (користувачеві має бути інтуїтивно зрозумілим інтерфейс застосунку, а дії мають легко запам’ятовуватися і не вимагати складних процедур);
- зовнішній дизайн (інтерфейс має бути у збалансованій колірній палітрі, не перевантажений зайвими деталями, розрахований на довготривалу роботу користувача з додатком протягом дня).

Користувач може взаємодіяти з пристроями та програмами за допомогою наступних засобів [14]:

- інтерфейс командного рядка (комп'ютер отримує команди від користувача у вигляді рядків тексту (команд));
- графічний інтерфейс користувача (програмні функції реалізуються через графічні елементи екрану);
- діалоговий інтерфейс (програмні засоби, призначені для обміну інформацією між користувачем і ОС);

- природно-мовний інтерфейс (посередник між БД та людиною, що відбувається за допомогою засобів взаємодії з програмою або приладом природною мовою користувача);
- тактильний інтерфейс (технологія базується на основі тактильного відчуття, наприклад, джойстик, кермо і.т.д);
- нейрокомп'ютерний інтерфейс (здійснює обмін між нейронами та електронним пристроєм).

Для створення даної гри було використано графічний інтерфейс, оскільки при використанні застосунку будуть використані графічні елементи для кращого візуального контакту з користувачем та реалізації поставлених задач.

Для кращого розуміння роботи системи мобільний інтерфейс має задовольняти всі вимоги UX/UI. Таким чином, шаблон ігрового застосунку має складатися з наступних частин, а саме: перша частина – авторизація користувача, друга – екран відображення рівнів, третя – шаблон відображення одного з рівнів та екран після проходження гри.

У Unity використано стандартні компоненти розробки інтерфейсу [15], а саме:

- Canvas. – область де знаходяться усі UI об'єкти Unity. Цей компонент є стандартним в Unity та належить ігровому об'єкту. Усі UI об'єкти є його дочірніми елементами;
- Text. – неінтерактивний текстовий елемент;
- Button. – елемент натиску користувачем для запуску певної події;
- Image. – відображає спрайт в системі UI;
- Slider. – надає можливість користувачу обирати числове значення у певному діапазоні.

Проєктування ігрового інтерфейсу розпочинається зі створення нової сцени, призначеної спеціально для головного меню, меню обрання рівня, та меню результатів проходження ігрових рівнів.

Коли було створено сцену треба додати Canvas на сцену та розмістити у ньому елемент Panel де будуть розташовані кнопки та текст (рис. 3.1).

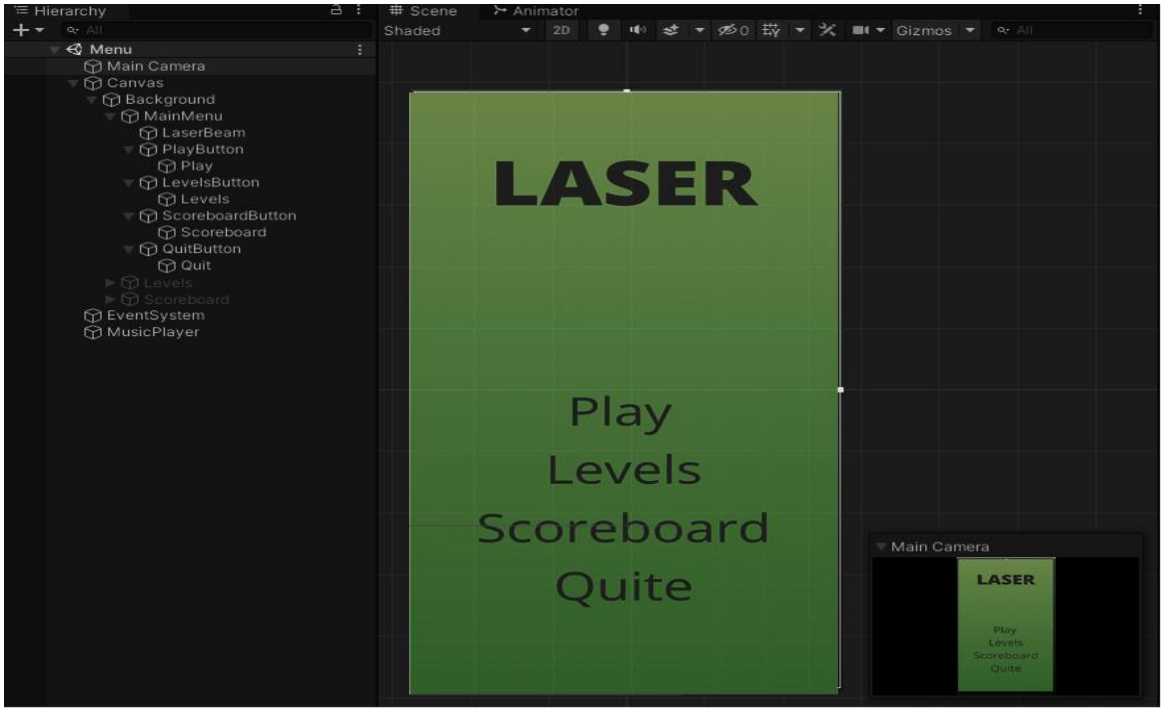

Рисунок 3.1 – Структура головного меню

Елемент TextMeshPro використовується для виведення тексту оскільки він має багато налаштувань якщо порівняти його із стандартним елементом Text.

TextMeshPro також буде використаним як елемент із бібліотеки для створення кнопки (рис. 3.2).

Створюємо asset. за допомогою елемента TextMeshPro. Це робиться для того, щоб мати можливість користуватися нестандартними шрифтами. Наступним кроком буде додавання необхідного шрифту у проєкт та використання вбудованого Font Asset Creator (рис. 3.3).

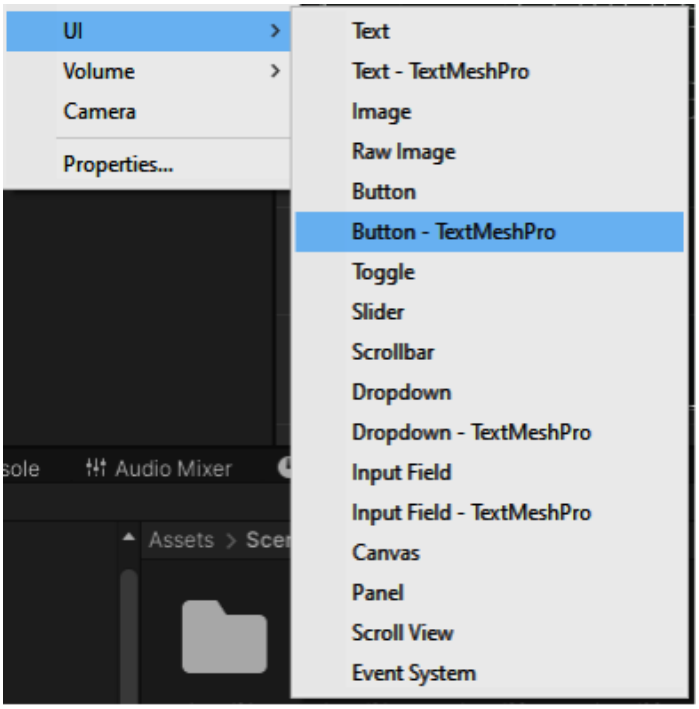

Рисунок 3.2 – Структура головного меню

Для використання шрифта у Font Asset Creator треба додати обраний шрифт та виконати налаштування, натиснувши кнопку Generate Font Atlas.

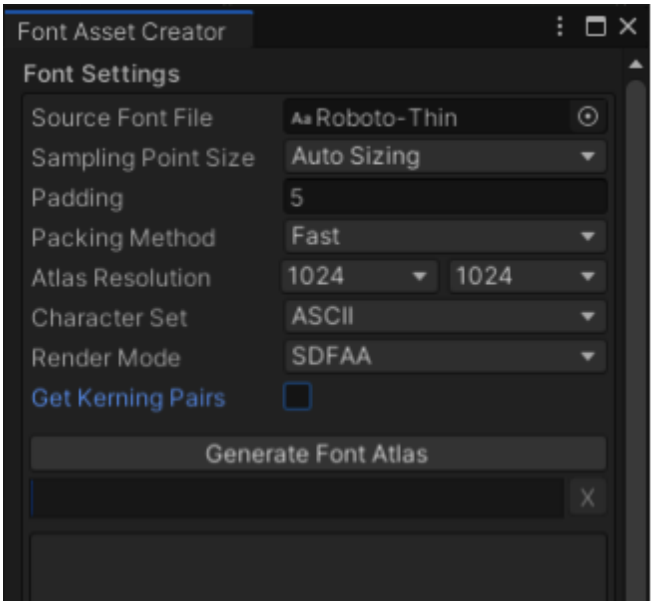

Рисунок 3.3 – Скрін вікна Font Asset Creator

Щоб розробити головне меню розміщуємо елементи на Canvas. Проєктуємо задній фон застосунку за допомогою сайту mdigi.tools (рис. 3.4). Розташовуємо елементи меню по місцям та виконуємо їх привя’зку (рис 3.5).

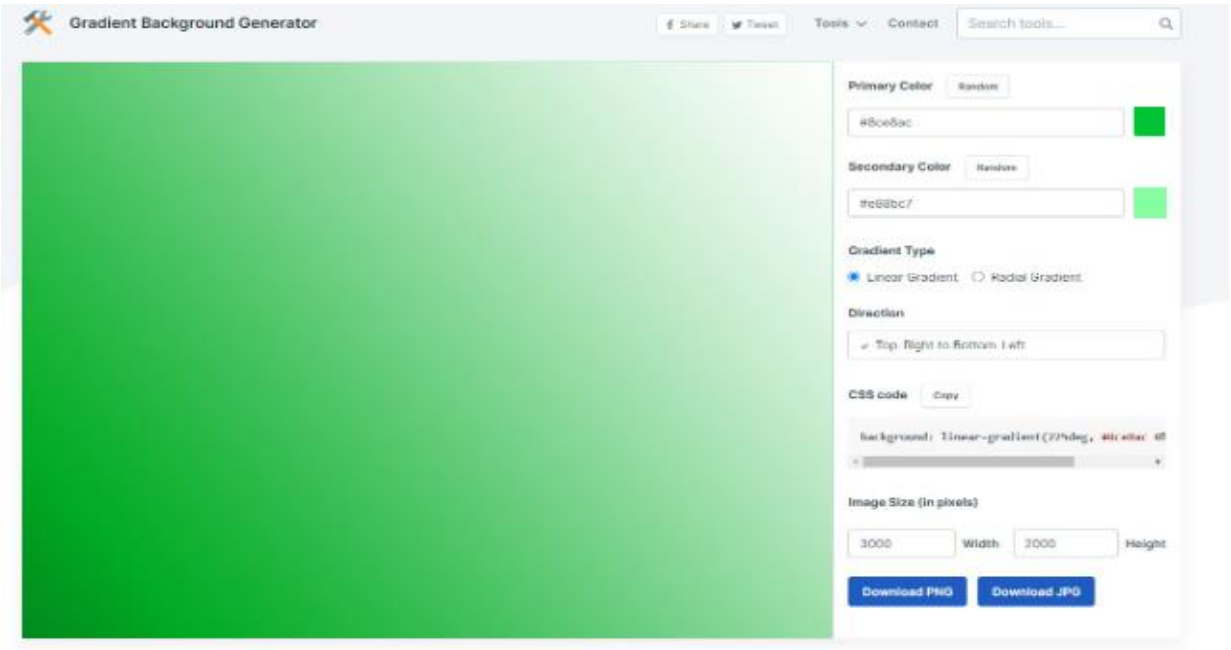

Рисунок 3.4 – Вікно для створення заднього фону додатку у вигляді градієнту

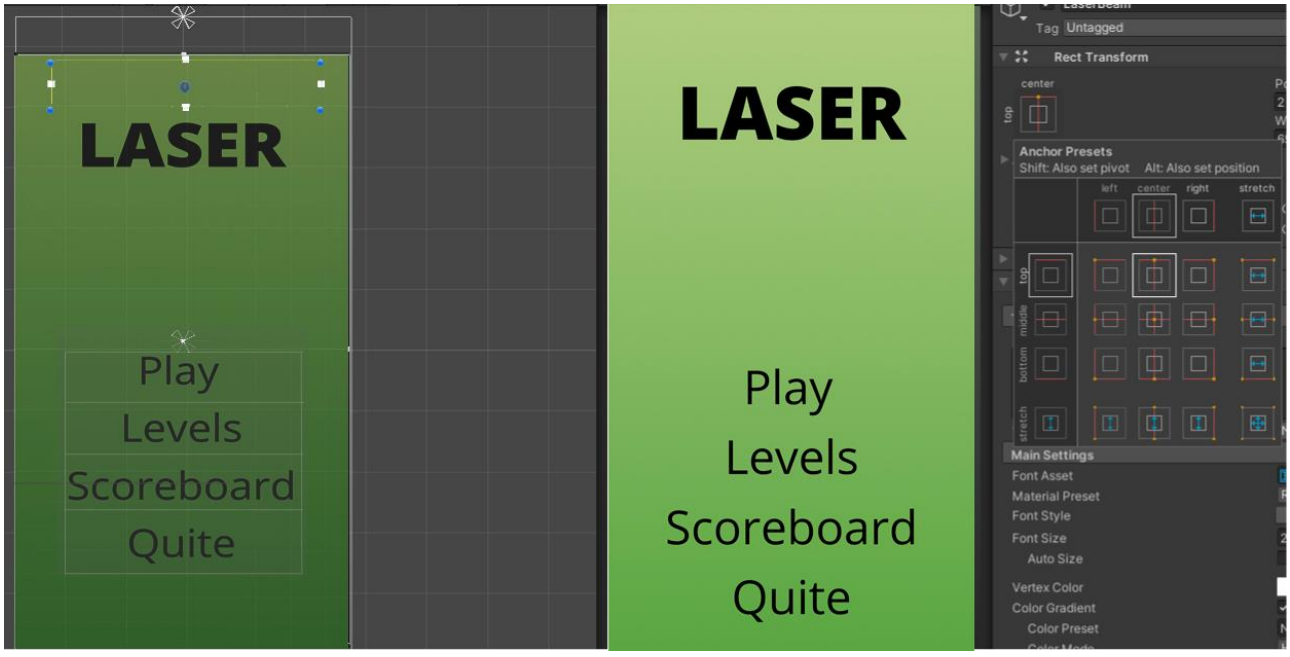

Рисунок 3.5 – Вікно створення та розміщення елементів на Canvas

Завершивши розташування елементів треба налаштувати функціонал кнопок. До елемента Panel додаємо скрипт та код очікуваної функції (рис. 3.6). У додатку А записано повний код скрипту для головного меню.

Наступним кроком здійснюємо прив’язку коду до створених кнопок. Через параметри елементу додаємо скрипт, обравши потрібну функцію (рис. 3.7).

```
rminal  Help                    mainMenu.cs - Laser Beam - Visual Studio Code
mainMenu.cs ×
Assets > Scripts > mainMenu.cs > mainMenu
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
    0 references
6   public class mainMenu : MonoBehaviour
7   {
        0 references
8       void Start()
9       {
10          if(Time.timeScale == 0f)
11              Time.timeScale = 1f;
12      }
        0 references
13      public void PlayGame()
14      {
15          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
16      }
17
        0 references
18      public void QuitGame()
19      {
20          Application.Quit();
21      }
22  }
```

Рисунок 3.6 – Код функцій створених кнопок

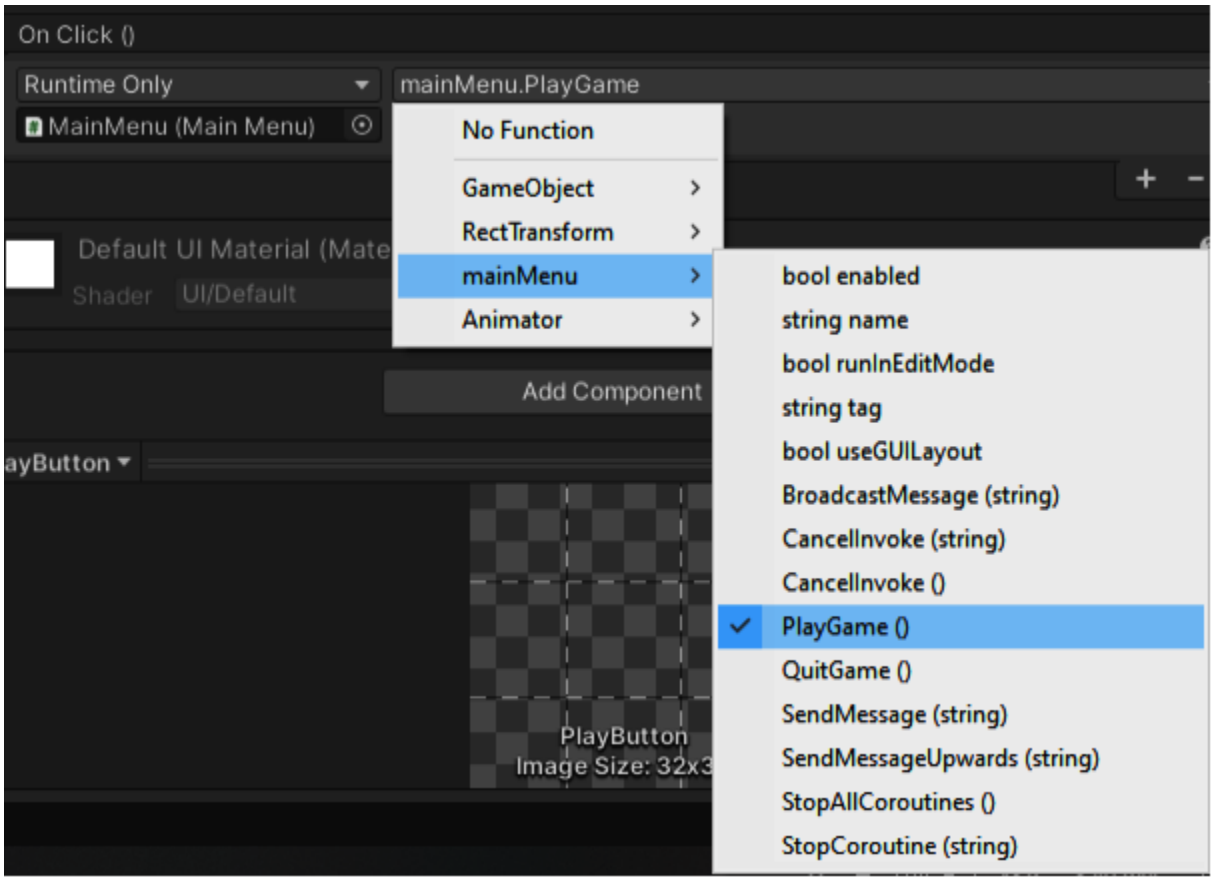

Рисунок 3.7 – Меню прив'язки написаних функцій до кнопок

Наступним кроком, після створення головного меню починаємо проєктувати інтерфейс для вибору ігрового рівня. Створюємо за аналогією елемент Panel в існуючому Canvas та додаємо до нього кнопки. Одна з кнопок буде здійснювати вибір рівню, а друга –здійснювати повернення до головного меню (рис. 3.8). За аналогією з попереднім меню, використовуємо шрифт для відображення тексту, що створили раніше.

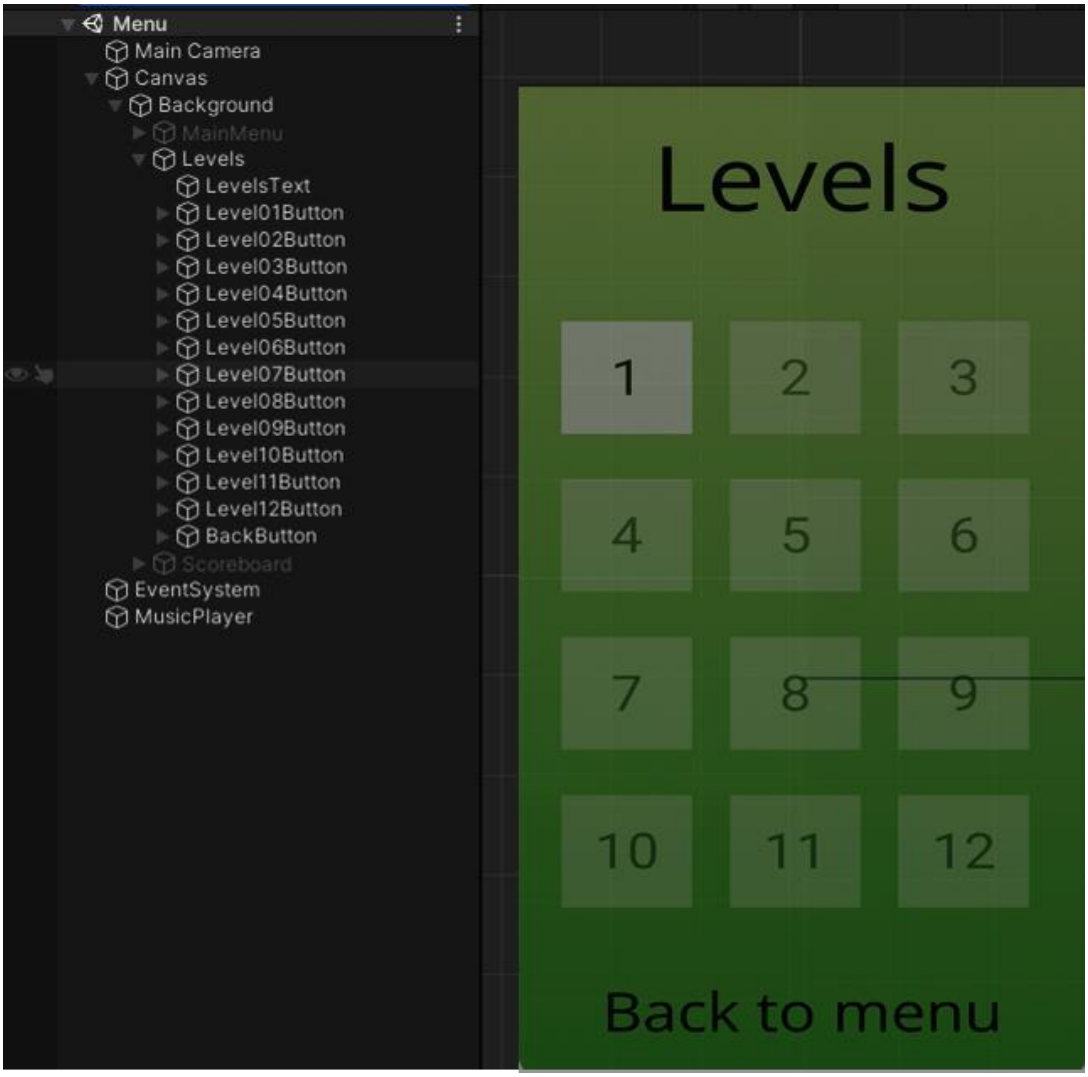

Рисунок 3.8 – Вікно створення інтерфейсу меню для вибору рівня

Для відкриття обраного рівня до кнопок прив’язуємо необхідні функції, тому створюємо скрипт менеджеру рівнів (рис. 3.9). Потім прив’язуємо функції до зазначених кнопок (рис. 3.10). У додатку А викладено повний код скрипту меню вибору рівня.

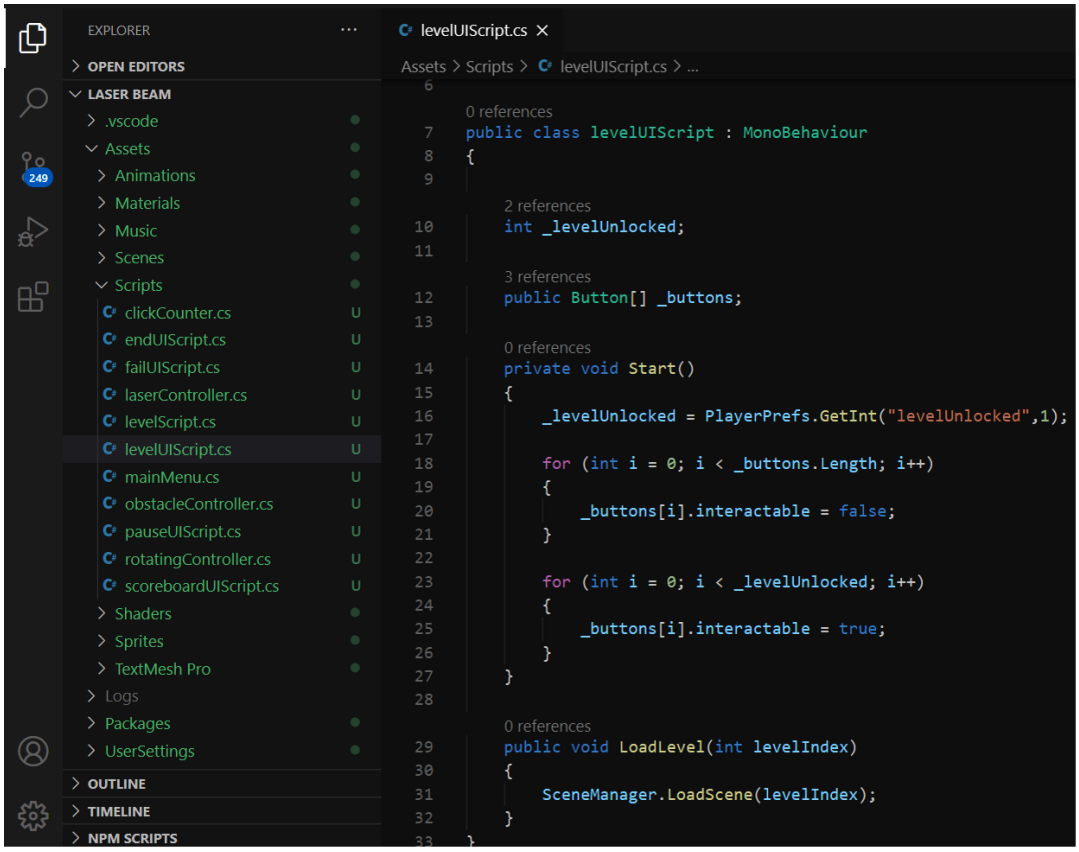

Рисунок 3.9 – Код меню для вибору рівнів гри

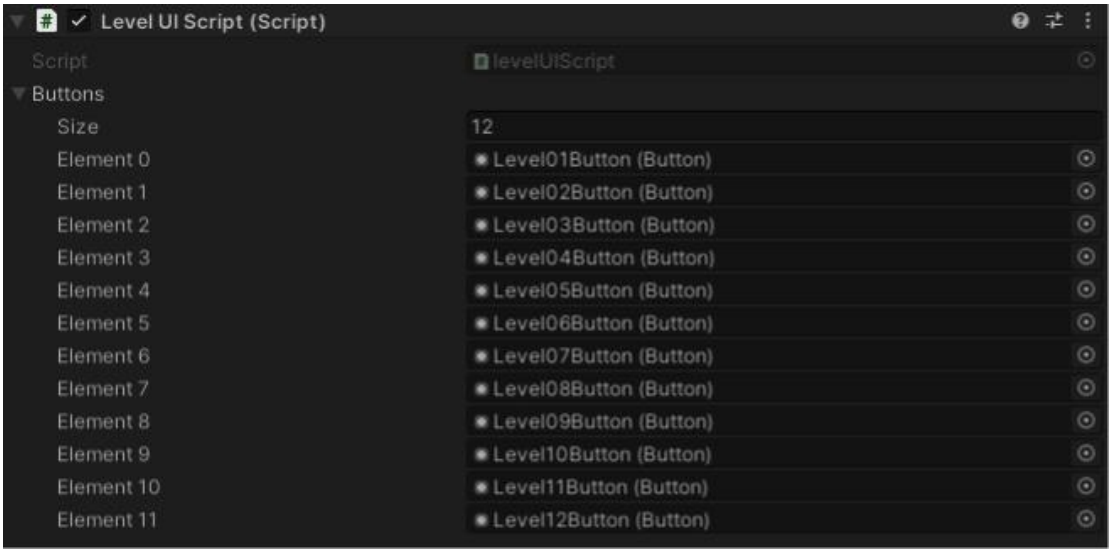

Рисунок 3.10 – Успішна прив'язка кнопок до функцій скрипта

Переходимо до наступного етапу. Створюємо інтерфейс меню результатів. Додаємо текст із назвами рівнів та кількість балів за нього. Також додаємо одну кнопку повернення у головне меню (рис. 3.11). Шрифт створений раніше використовуємо, щоб стилізувати текст.

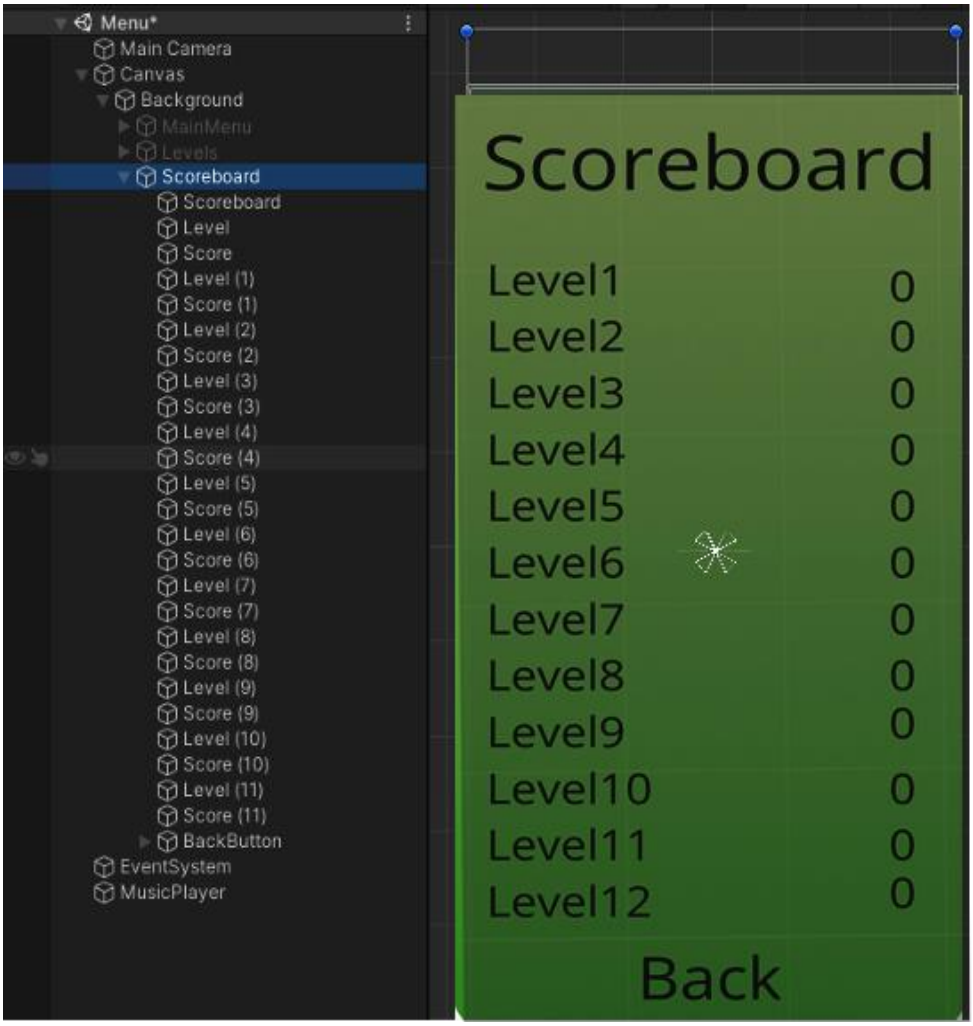

Рисунок 3.11 – Структура меню вікна результатів проходження рівнів

Наступним кроком створюємо скрипт для виведення результатів на екрані (рис. 3.12). У додатку Б наведено код скрипту меню перегляду результатів проходження.

Таким чином, створивши інтерфейси головного меню, наступним кроком є розробка ігрових інтерфейсів. Необхідно створити інтерфейси гри, паузи, вдалого та невдалого проходження рівнів.

Елементи TextMeshPro та раніше створений шрифт були використані в зазначених інтерфейсах.

Текст із кількістю доступних натискань на люстерка та кнопку паузи містить у собі ігровий інтерфейс, структуру якого зображено на рис. 3.13.

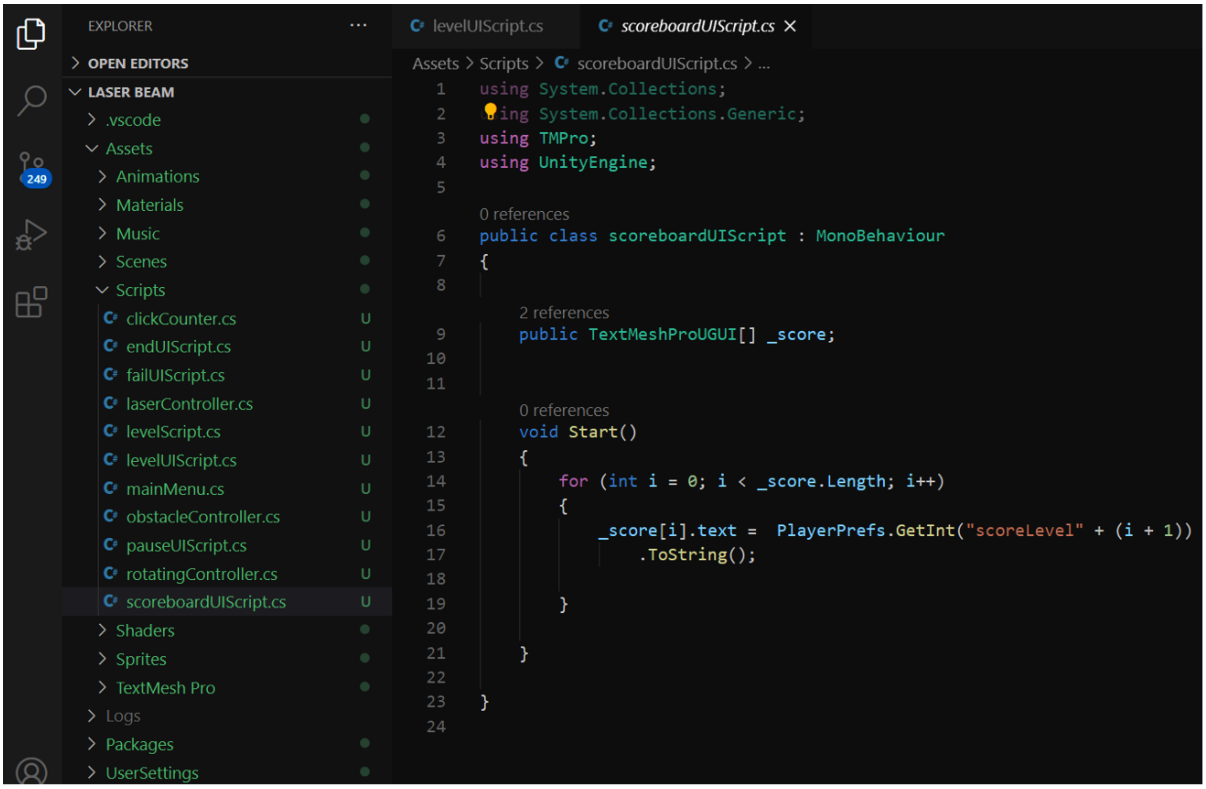

Рисунок 3.12 – Код відображення результатів проходження

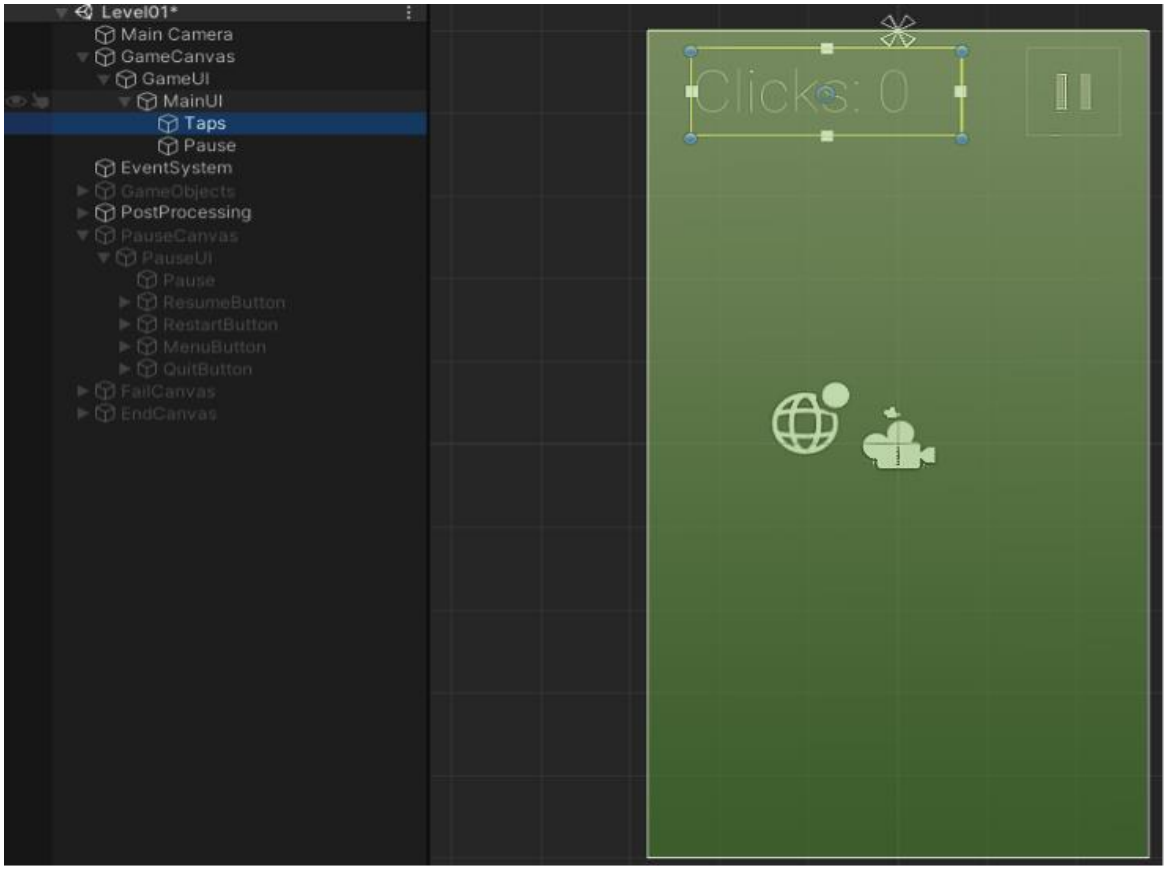

Рисунок 3.13 – Структура ігрового інтерфейсу

Потім було створено скрипт (рис. 3.14), який відображає кількість натискань та функціювання кнопки паузи, та додано його до кнопки.

Зазначений параметр було винесено у параметри ігрового об'єкта меню (рис. 3.15), що надало можливість налаштувати кількість доступних натискань.

Також, варто виділити випадки коли гавець пройшов рівень невдало, у такому випадку скрипт повинен відповідати за виклик інших інтерфейсів. У додатку А додано повний код даного скрипту.

obstacleController.cs | levelScript.cs

Assets > Scripts > levelScript.cs > levelScript > Pause()

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

0 references
private void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        _clickText.text = "Clicks: " + globalClickController.sharedInstance._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);
    }
}

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

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

Рисунок 3.14 – Код скрипта ігрового інтерфейсу

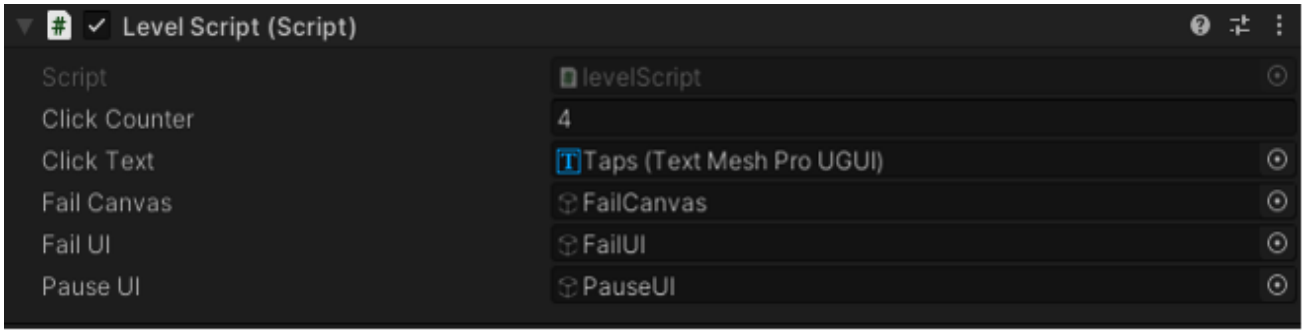

Рисунок 3.15 – Основні параметри об’єкту інтерфейсу

Наступним є інтерфейс паузи. Він містить у собі тільки його назву та кнопки з функціями (рис. 3.16). Розробляємо скрипт для функціювання кнопок (рис. 3.17). Далі прив’язуємо кожну функцію до відповідної кнопки (рис. 3.18).

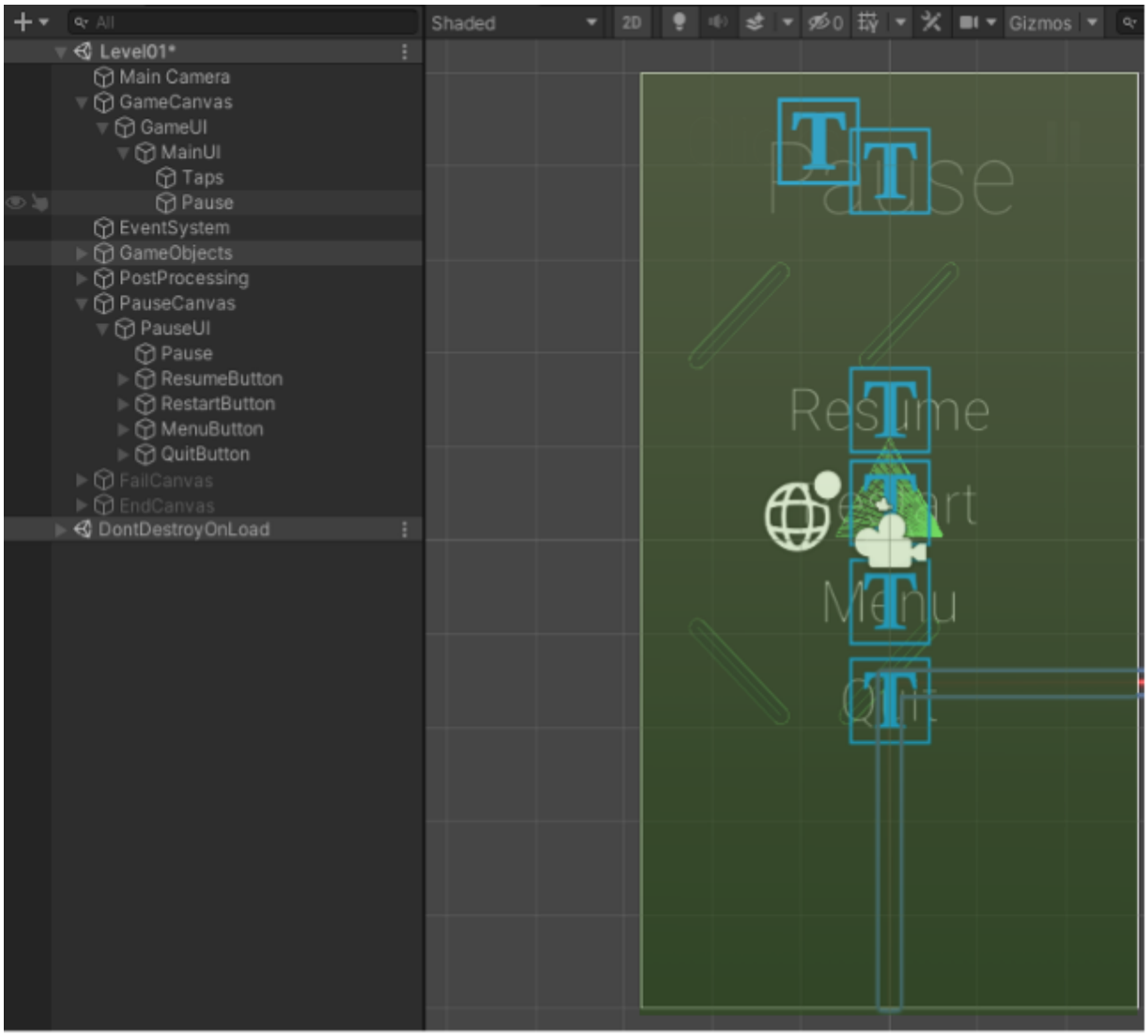

Рисунок 3.16 – Вікно розробки інтерфейсу паузи

```
obstacleController.cs   pauseUIScript.cs ×
Assets > Scripts > pauseUIScript.cs > pauseUIScript > ResumeGame()
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
    0 references
6   public class pauseUIScript : MonoBehaviour
7   {
8
        1 reference
9       public GameObject _pauseUI;
10
        0 references
11      public void ResumeGame()
12      {
13          Animator animator = _pauseUI.GetComponent<Animator>();
14
15          if(animator != null)
16          {
17              animator.SetBool("pauseTransition", false);
18          }
19          Time.timeScale = 1f;
20      }
21
        0 references
22      public void RestartGame()
23      {
24          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
25      }
26
        0 references
27      public void BackToMain()
28      {
29          SceneManager.LoadScene(0);
30      }
31
        0 references
32      public void QuitGame()
33      {
34          Application.Quit();
```

Рисунок 3.17 – Скрипт інтерфейса паузи

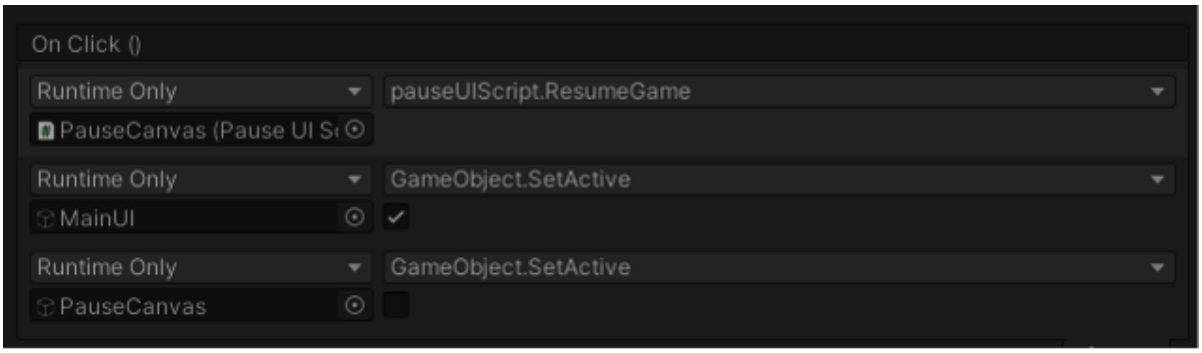

Рисунок 3.18 – Основні параметри кнопки «Resume»

Наступним кроком є проєктування інтерфейсу невдалого проходження рівня, який, подібно до інтерфейсу паузи, містить у собі тільки назву та функціональні кнопки. Його структура зображена на рис. 3.19.

Також було розроблено скрипт для функціювання кнопок (рис. 3.20).

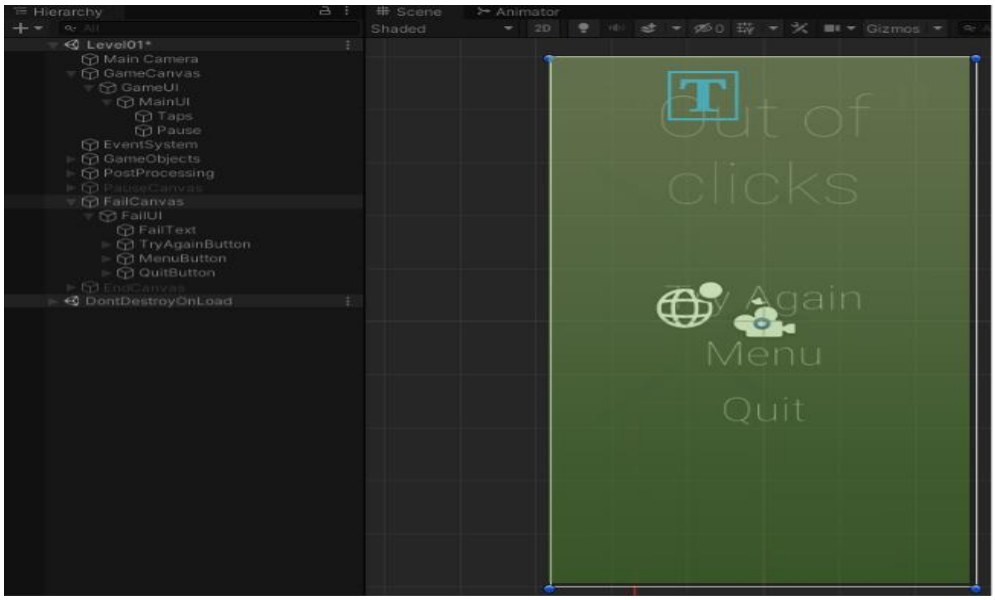

Рисунок 3.19 – Вікно створення інтерфейсу у разі невдалого проходження

```
obstacleController.cs   failUIScript.cs ×
Assets > Scripts > failUIScript.cs > ...
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
    0 references
6   public class failUIScript : MonoBehaviour
7   {
8
        0 references
9       public void ReloadLevel()
10      {
11          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
12      }
        0 references
13      public void BackToMain()
14      {
15          SceneManager.LoadScene(0);
16      }
17
        0 references
18      public void QuitGame()
19      {
20          Application.Quit();
21      }
22  }
23
```

Рисунок 3.20 – Код інтерфейсу у разі невдалого проходження

Останнім кроком було розроблено інтерфейс у разі вдалого проходження ігрового рівня, що містить у своїй загальній структурі

функціональні кнопки та текст, що зображають загальний результат проходження рівня (рис. 3.21).

Було створено скрипт (рис. 3.22), що відображає загальний результат проходження та здійснює функціонування кнопок та відповідає за кількість зроблених натисків гравцем, що на основі цих даних формує проміжний результат. Ігровий об’єкт тексту було передано у скрипт для виведення на екран результату гри (рис. 3.23).

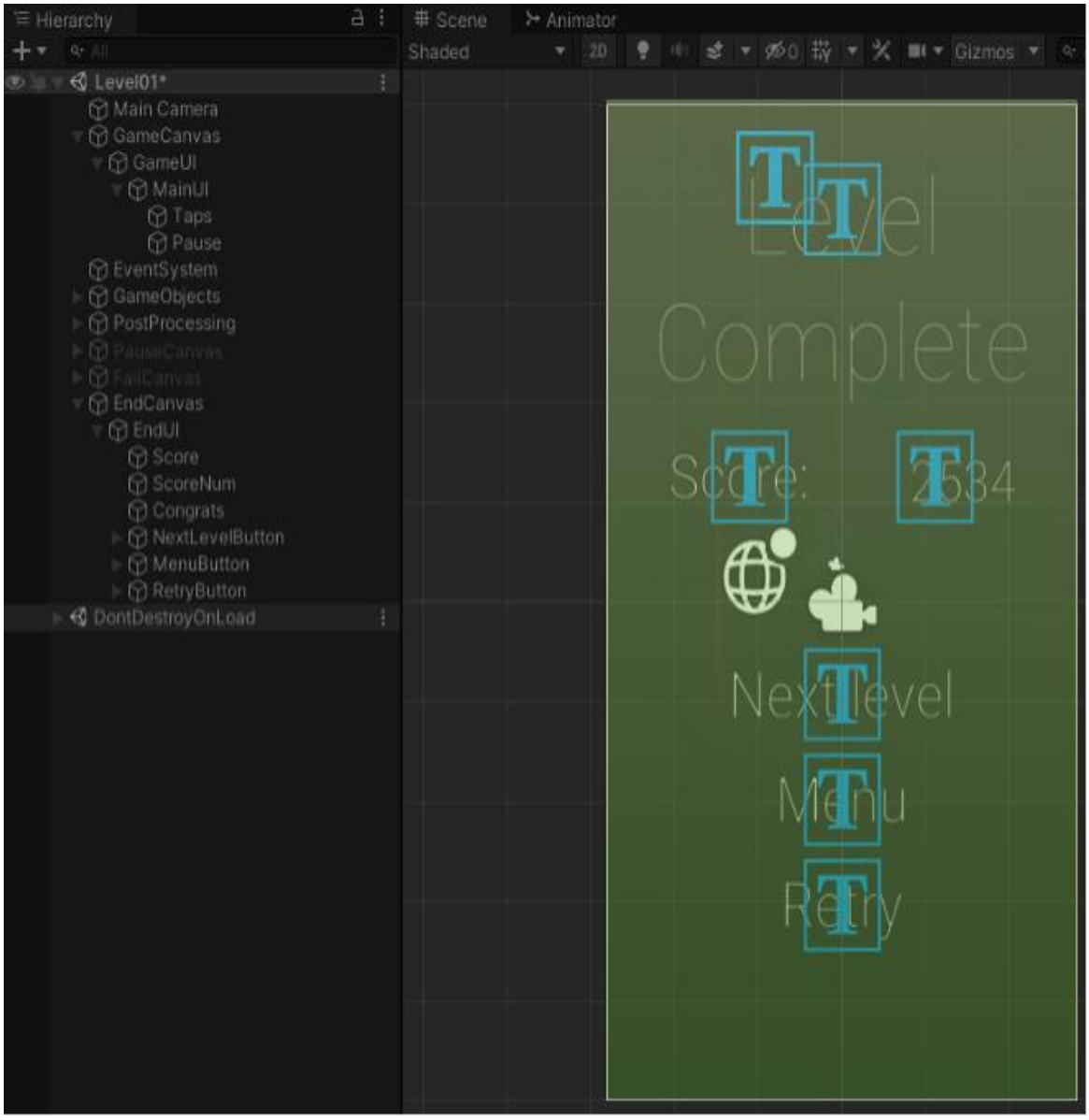

Рисунок 3.21 – Створення інтерфейсу у разі вдалого проходження

```
obstacleController.cs   endUIScript.cs ×
Assets > Scripts > endUIScript.cs > endUIScript
using UnityEngine;
using UnityEngine.SceneManagement;

0 references
public class endUIScript : MonoBehaviour
{

    1 reference
    public TextMeshProUGUI _currentScoreText;
    3 references
    public GameObject _endUI;

    4 references
    int _currentScore;

    0 references
    private void Start()
    {
        _currentScore = globalClickController.sharedInstance._clickCounter * 1267;
        if(PlayerPrefs.GetInt("scoreLevel" + SceneManager.GetActiveScene().buildIndex) < _currentScore)
            PlayerPrefs.SetInt("scoreLevel" + SceneManager.GetActiveScene().buildIndex, _currentScore);

        _currentScoreText.text = _currentScore.ToString();

    }

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

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

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
```

Рисунок 3.22 – Код інтерфейсу у разі вдалого проходження

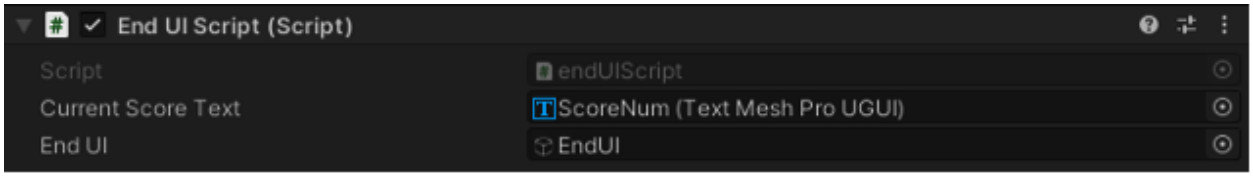

Рисунок 3.23 – Відправка ігрового об’єкту у скрипт

3.2 Вибір технологій розробки гри

На сьогодні найбільш доступними, комфортними у роботі, простими та застосовуваними є два основні ігрові рушія та прив’язані до них мови програмування для створення мобільних ігрових додатків, це: **Unreal Engine та Unity.**

Unreal Engine – ігровий рушій, створений компанією Epic Games [16] та вперше застосований в шутері від першої дійової особи у 1998 році. Даний рушій напочатку був створений для шутерів від першої особи. Але потім

його застосовували в багатьох різних жанрах, таких як файтинги, платформери, MMORPG та інші RPG. Unreal Engine написаний на C ++. Рушій має високий ступінь портативності, підтримує багато платформ та розробку за допомогою блупринтів [14].

Unity — це 2D та 3D ігровий рушій, який існує з 2005 року. Розроблений Unity Technologies, він був створений для того, щоб надати більшій кількості розробників доступ до інструментів розробки ігор, що в ті часи було новим підприємством. Протягом свого довгого терміну служби двигун різко змінився та розширився, зуміючи йти в ногу з найновішими практиками та технологіями. Станом на 2018 рік Unity підтримує більш ніж 25 платформ (рис. 3.24).

Unity пропонує дуже прозорий метод для створення архітектури вашої гри. Кожен «рівень» в ігровому проекті Unity поділено на сцену, і кожна сцена містить усі ігрові об’єкти, необхідні гравцеві для використання рівня – будь то фон, персонаж гравця, ворог, куля чи щось інше [15].

Unity – це не тільки рушій, а й середовище розробки комп'ютерних ігор. У даному рушії поєднані різноманітні програмні засоби, які використовуються під час створення програмного забезпечення, а саме: текстовий редактор, компілятор, відладчик тощо. Unity робить розробку ігор максимально простою та комфортною, завдяки зручності використання, а його мультиплатформність дає можливість охопити велику кількість ігрових платформ та операційних систем [15].

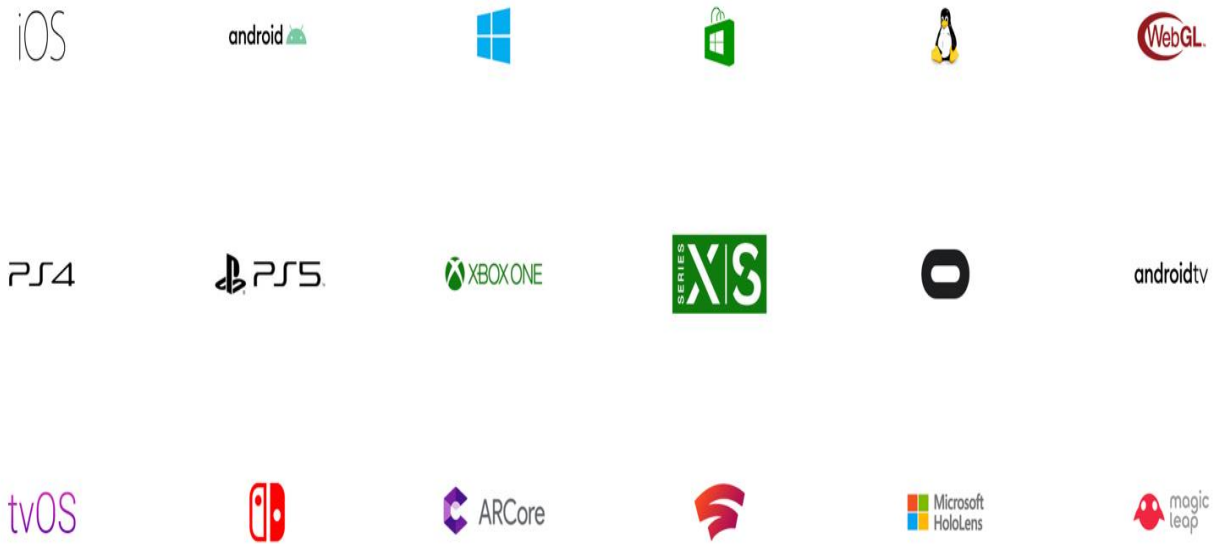

Рис. 3.24 – Платформи та технології, які підтримує Unity

Оскільки Unity є універсальним редактором він сумісний із Windows, Linux, Mac, IOS, Android, Tizen, Xbox, PS4, Switch та іншими платформами, а його інтуїтивно зрозумілий інтерфейс спрощує розробку та зменшує потребу у навчанні.

У Unity проєктам необхідно мати хоча б одну сцену, тобто окремий файл, який містить у своєму складі ігрові об’єкти, скрипти, налаштування (рис. 3.25). Об’єкти містять різноманітні набори компонентів з якими взаємодіють скрипти. Кожен об’єкт має назву, шар, де він відображається, та тег. Рушій може допускати існування декількох об’єктів з однаковими назвами.

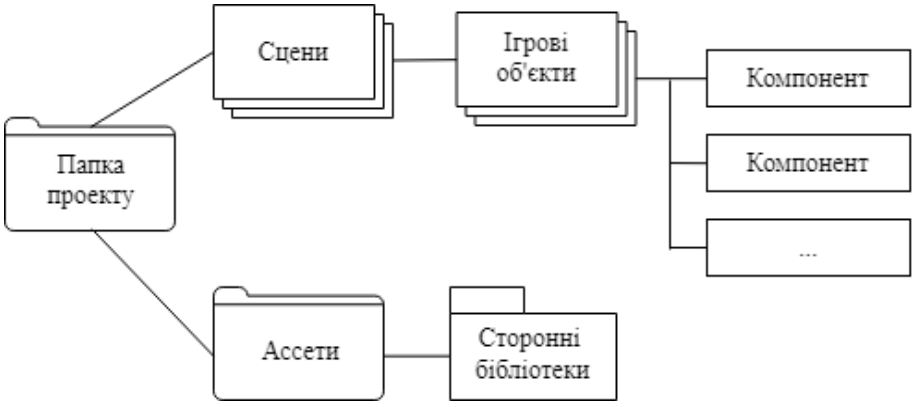

Рисунок 3.25 − Структура проекту на Unity

Кожнен об’єкт має містити компонент Transform. Цей компонент виконує зберігання інформації про йoго координати положення на сцені, повороти, розміри об’єкту на трьох осях. Mesh Renderer − один з важливиих компонентів у Unity. Цей компонент робить модель об’єкта видимою.

Написання та редагування шейдерів також підтримує редактор Unity. Він також має компоненти для створення анімації, яку попередньо можна розробити в зручному для себе 3D-редакторі, імпортувавши її разом з моделлю. Для легкої передачі моделей, скриптів, матеріалів та звуків можна запакувати їх у формат unityassets. Unity Asset Store використовує цей формат, в який розробники викладають свої ассети, заробляючи на цьому.

Порівняльний огляд платформ для створення ігрових додатків зображено на рис. 3.26

Таким чином, Unity був обраний у якості ігрового рушія, оскільки він має ряд переваг: простота розробки, зручність використання, сучасний рівень графіки, кросплатформність та ін. Unity дає можливість створення гри на різних платформах з мінімальним витрачанням часу, що буде необхідним для подальшого розвитку та розширення функціоналу додатка.

Мова програмування – це формальна знакова система, призначенням якої є запис комп'ютерних програм. Набір лексичних, синтаксичних і семантичних правил визначає мову програмування, а також зовнішній вигляд програми і дій, які виконує комп'ютер під її управлінням [17].

Платформа	Unity	UE4
Мова програмування	C#, JavaScript	C++
Ціна	Free/ 1500$ pro	Free/ 3000$ pro
Мульти-платформенність	+	+
Відкритий код	-	-
Зручність використання	5/5	4/5
Масштабність проектів, що розробляються	Дрібні, середні, великі	Середні, великі

Рисунок 3.26 − Переваги та недоліки Unity

Людство вигадало більше 2500 мов програмування з моменту створення перших програмованих машин. Їх кількість щороку збільшується. Серед усіх мов на сьогодні найвідомішими та найбільш використовуваними можна назвати C, C++, C#, Java, PHP, Delphi, Visual Basic та ін.

Проаналізувавши всі плюси та мінуси зазначених вище мов програмування, робимо висновок, що для розробки найефективнішою мовою програмування буде C#. Рушій Unity підтримує мову C# для написання скриптів, а також декілька шейдерних мов зі спеціальними надбудовами. При вирішенні поставленої задачі оптимально використовувати мову C#, оскільки вона є мовою високого рівня і дає можливість швидко та ефективно створювати програми. Отже, для створення гри будемо використовувати мову C#, про яку далі.

Мова програмування C# − це об’єктно орієнтована мова програмування загального призначення, створена компанією Microsoft

спільно з платформою .NET. Відноситься до мов з С-подібним синтаксисом, який подібний до синтаксису мов C++ та Java. Оскільки C# створили під впливом зазначених мов, таким чином вона містить у собі лише їх найкращі сторони та фічі, і разом з тим не має всіх недоліків своїх попередників. Має строгу статичну типізацію, оскільки її розробили спеціально для програмування з використанням платформи .Net [18].

Так як програмування в Unity базується на мові C#, необхідно розглянути переваги та недоліки використання даної мови.

Переваги:

- регулярно розвивається, додаються нові можливості у синтаксисі та у конструкціях для користувачів;
- використовує ООП до всього, що дозволяє робити абстрактні конструкції; для поступової роботи над проєктом;
- бібліотека шаблонів, яка дозволяє користуватися уже давно створеними конструкціями, які ідеально налагоджені та стабільно працюють;
- універсальність.

Недоліки:

- сталі конструкції інколи починають працювати некоректно;
- під час першого запуску програми можуть виникати проблеми з затримками в обробці, фрізи та лаги;
- програми можуть працювати повільніше ніж ті, які написані на спеціально розроблених для цього мовах програмування.

Таким чином, маючи свої плюси та мінуси, C# є однією з найпопулярніших мов програмування, що підходить для розробки мобільних ігрових застосунків.

Так як мову C# було обрано як мову програмування можна виокремити такі основні можливі середовища розробки: **Visual Studio Code та Visual Studio**.

Для розробки на C# існує багато середовищ, та найпопулярнішим є Visual Studio (рис 3.27). Unity надає можливість користувачам при встановлені Unity встановлювати разом із нею Microsoft Visual Studio, щоб можна було робити скрипти на C# та одразу імпортувати їх у проєкт [19].

Microsoft Visual Studio – це програмне середовище для розробки, що надає можливість створювати різноманітні додатки для для будь-яких ОС, для консольних включно. Програмне середовище має багато фунціоналу, такого як: редактор вихідного коду, відладчик коду, редактор форм, веб-редактор, дизайнер класів та схем баз даних та ін. Visual Studio дає можливість розширити функціонал середовища за допомогою підключення сторонніх додатків (плагінів). За допомогою плагінів для роботи на предметно-орієнтованих мовах програмування можна редагувати та візуально проектувати код інших мов (окрім C, C++,C#).

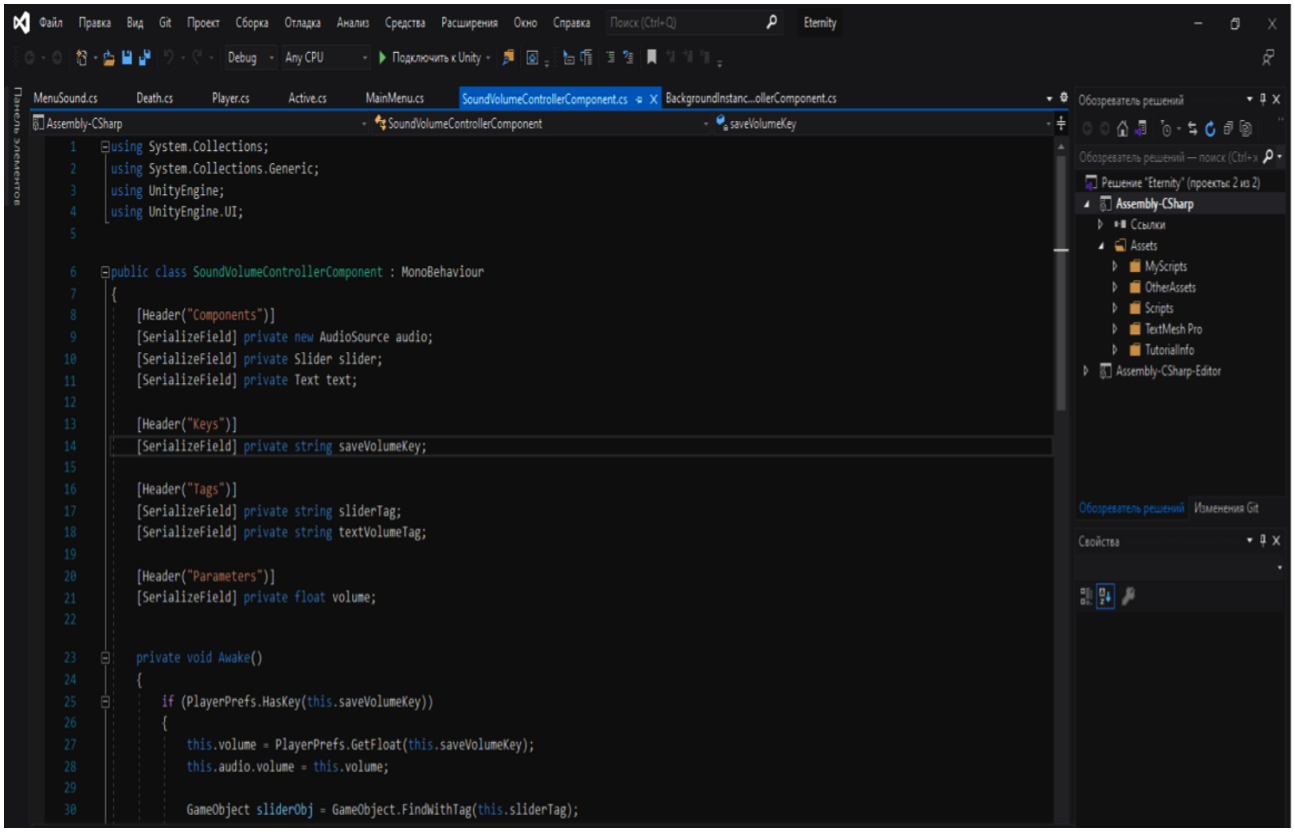

Рис. 3.27 – Середовище розробки Microsoft Visual Studio.

Visual Studio – це інтегроване кросплатформене середовище, яке може підтримувати багато мов програмування [26]. Серед середовища переваг такі:

- безкоштовне розповсюдження;
- велика база плагінів;
- автодоповнення та виправлення коду;
- доступна та проста документація;
- вбудована система контролю версій;
- ком’юніті розробників, що готові допомогти;
- можливість використання середовища на різних ОС.

Visual Studio Code – редактор вихідних кодів, розроблений Microsoft для операційних систем Windows, Linux та macOS [20]. Дане середовище використовується для розробки програмних засобів, а саме: програмних застосунків, web додатків, API та мобільних додатків. Включає в себе засоби

для створення схем та моделей, редактор коду, засоби тестування. Підтримує 36 мов програмування в тому числі і C#.

При використанні можна виділити такі переваги:

- безкоштовне розповсюдження;
- підсвічування синтаксису;
- інтелектуальне доповнення коду;
- рефакторинг коду;
- дебагінг;
- вбудований Git.

Також користувачі можуть додавати та розширювати функціональність шляхом встановлення розширень, що дозволяють змінити теми, комбінації клавіш, налаштування.

Таким чином, для виконання поставлених для розробки задач найкращим середовищем програмування буде Visual Studio Code. Програма Adobe Illustrator буде використана для розробки спрайтів, а плагін Google AdMob буде використаний для подальшого встроювання реклами.

Adobe Illustrator – це програмний додаток для створення ілюстрацій та малюнків. Створений у 1987 році, регулярно оновлюється, і на сьогодні входить до складу Adobe Creative Cloud [21]. По всьому світі для створення високоякісних ілюстрацій Illustrator використовується графічними дизайнерами, веб-дизайнерами, художниками-візуалістами та професійними ілюстраторами. Дана програма має велику кількість складних інструментів малювання, які скорочують час на створення ілюстрацій (рис. 3.28).

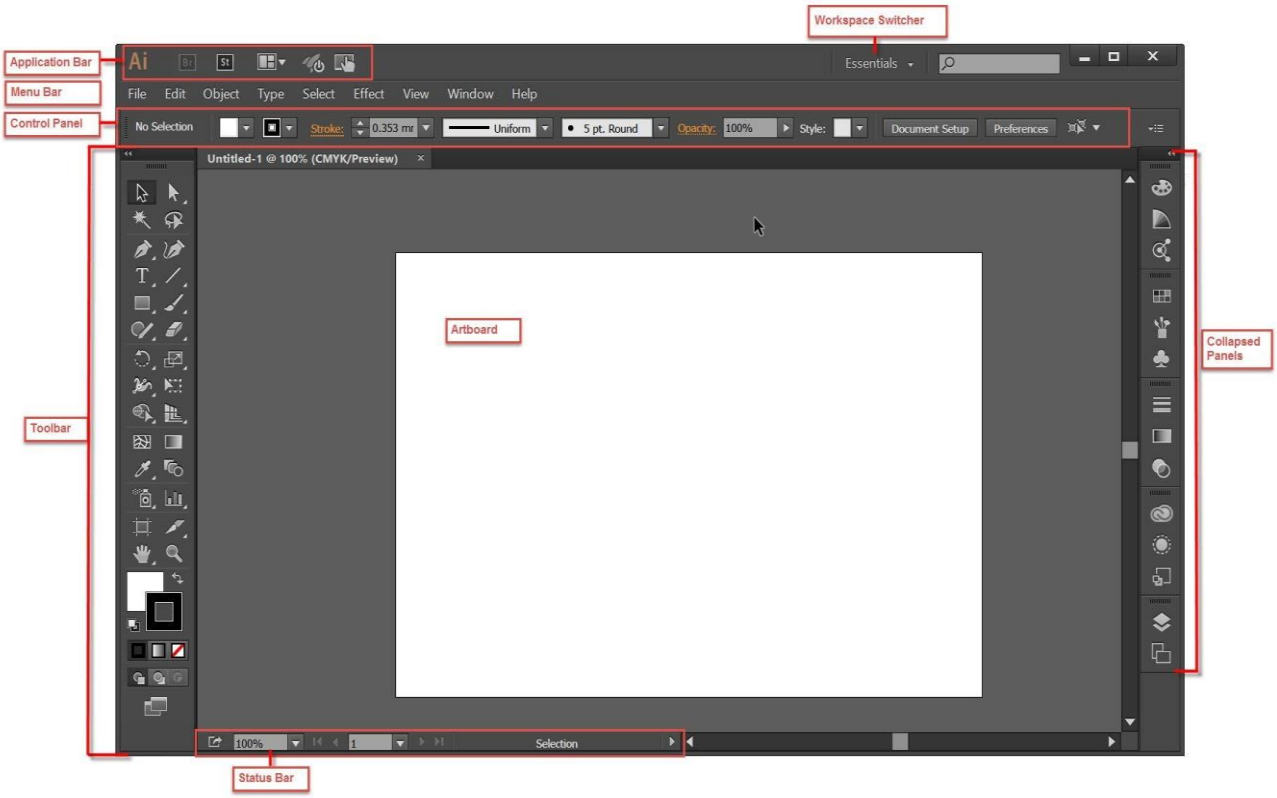

Рис. 3.28 – Інтерфейс Adobe Illustrator

Google AdMob – це плагін Admob SDK для Unity3d для легкого додавання реклами Google у мобільний додаток на Unity3d. Даний плагін підтримує Android та IOS, а також підтримує інтерстицію та банер AdMob.

3.3 Розробка прототипу ігрового додатку

Створення прототипу гри є важливим та необхідним етапом розробки. На основі ігрового прототипу, який є необхідним для тестування механіки і геймплея гри, можна планувати наступні етапи роботи та мати відправну точку усього проєкту.

Механіка головоломки була обраною як основна механіка додатку. У зв’язку з цим гра має реалізуватись у вигляді задач, які будуть вирішуватись гравцем. Головна задача гравця – налаштувати лазер таким чином, щоб він влучив у ціль за допомогою люстерок.

Після планування основної механіки гри можна починати її розробку, під час якої будуть використовуватись лише основні дії без ефектів та інтерфейсів, тільки геймплей [21].

У рамках роботи над прототипом необхідно виокремити дві основні механіки, які треба реалізувати у проєкті:

- механіка лазеру;
- механіка ротації люстерок.

Для цього треба створити проєкт та зробити налаштування збірки (рис. 3.29), обравши Android як платформу розробки.

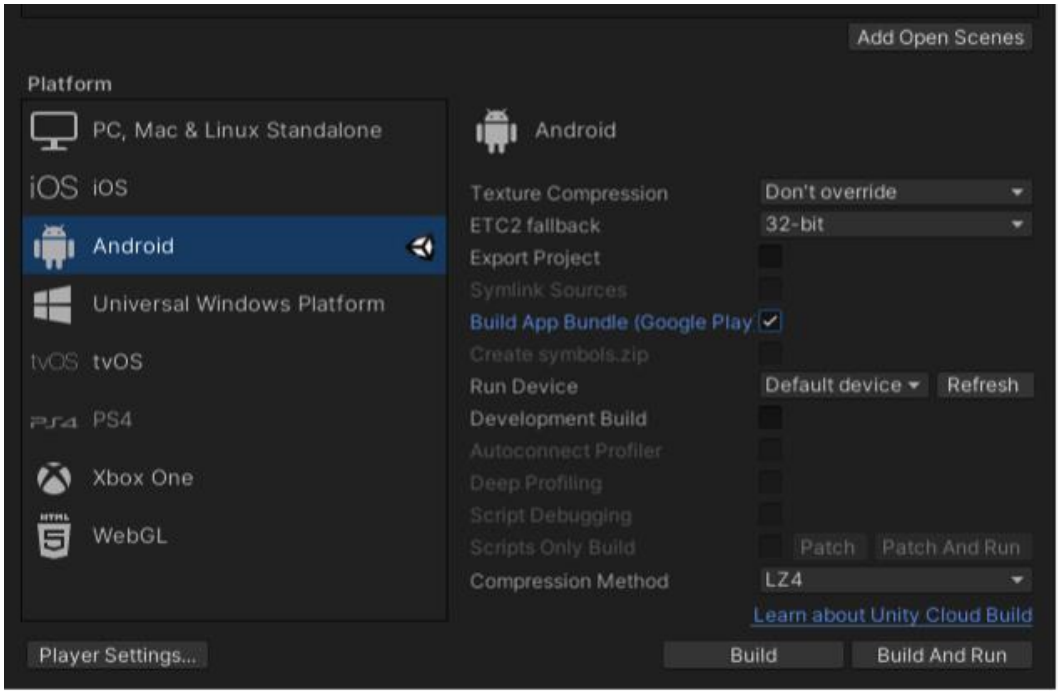

Рисунок 3.29 – Попереднє налаштування збірки проекту

Наступним кроком буде розробка механік. Елемент Line використовуємо як лазер. Даний елемент має у своєму складі компонент LineRenderer, що відповідає за відображення лазеру. У структуру додатку додаємо елемент та налаштовуємо його (рис. 3.30).

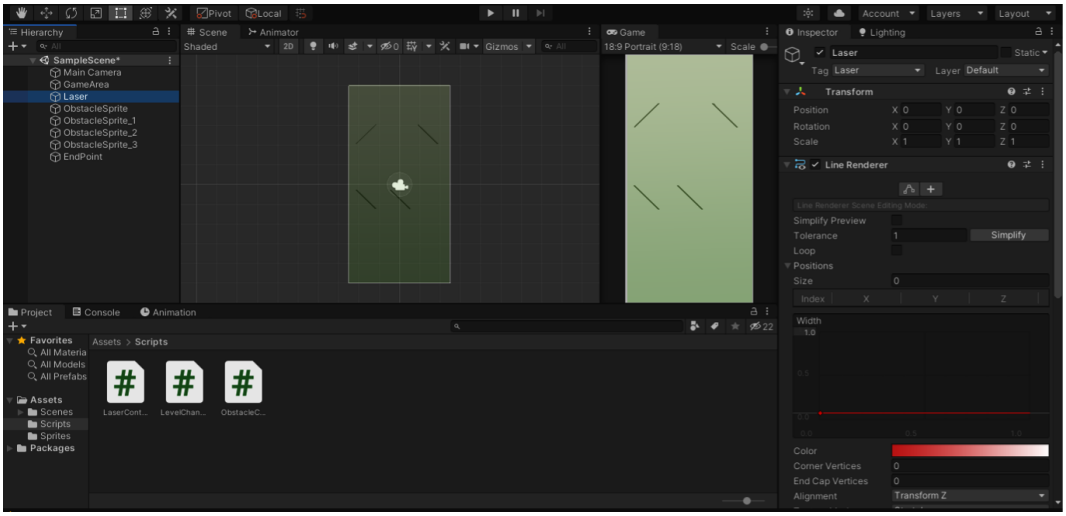

Рисунок 3.30 – Налаштування елементу Line

До створеного раніше елемента додаємо новий компонент у вигляді скрипта.

Пишемо код відбивання лазеру від примітивів. Далі у середовищі розробки відкриваємо створений скрипт та розпочинаємо роботу над кодом (рис. 3.31).

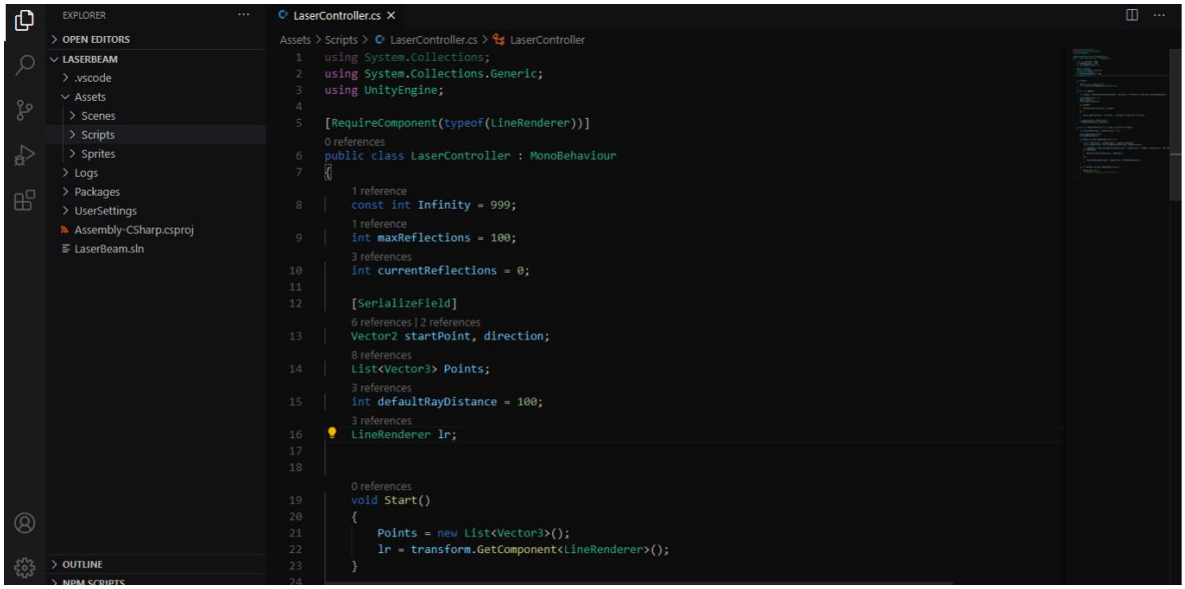

Рисунок 3.31 – Процес створення лазеру

Physics2D.Raycast., що у сцені створює промінь проти колайдерів, був використаний для імітування відбиваття лазеру. Концептуально промінь

подібний до лазерного, що випускається вздовж певного напрямку з точки, та під час контакту з колайдером про це буде надіслане повідомлення.

У додатку знаходиться повний код скрипту лазеру.

До ігрової області додаємо примітив, щоб протестувати лазер. У Adobe Illustrator робимо лінію, яка потім буде замінена на повноцінний спрайт. Додаємо в ігрове середовище зроблену лінію та компонент EdgeCollider2D (рис. 3.32) для того, щоб від її поверхні міг відбиватися лазер.

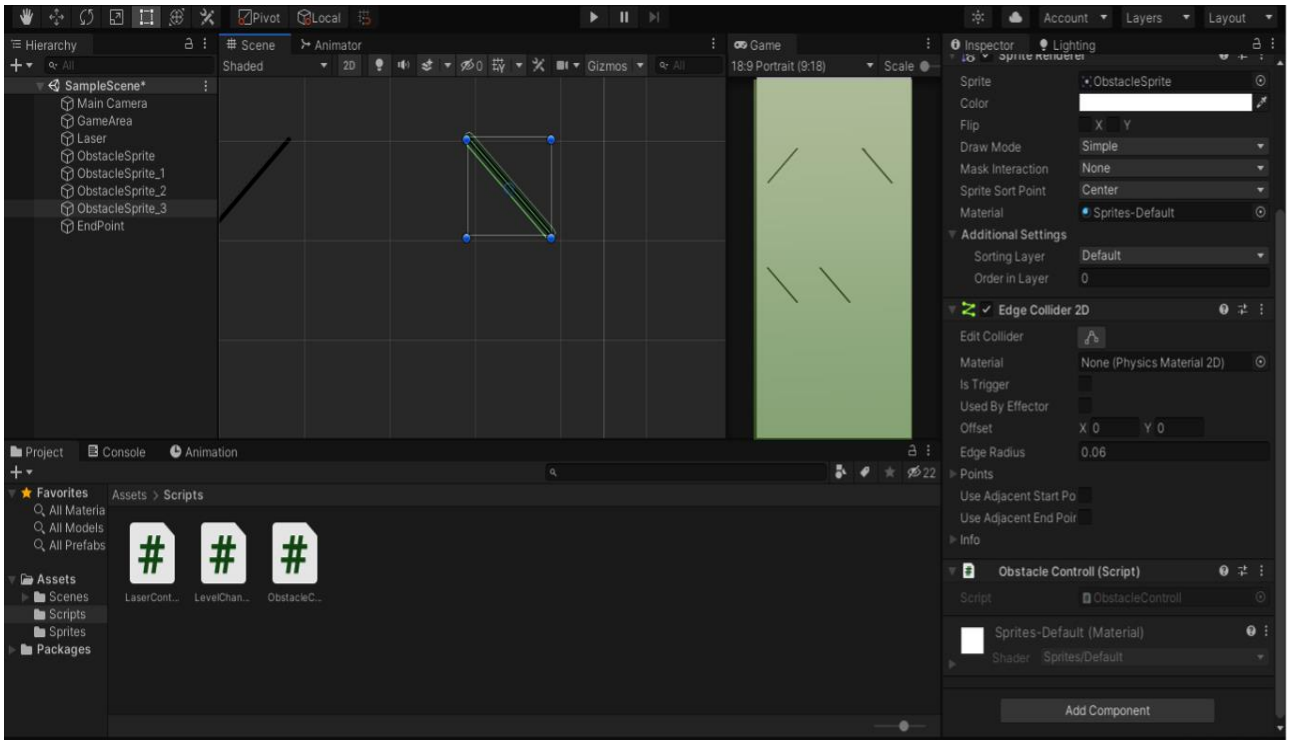

Рисунок 3.32 – Додавання компоненту до ігрового об’єкта

Переходимо до тестування механіки лазеру після додавання до ігрової області всіх примітивів з налаштованими колайдерами. На рис. 3.33 зображено встановлення у початкову позицію лазеру та запуск попереднього перегляду додатку.

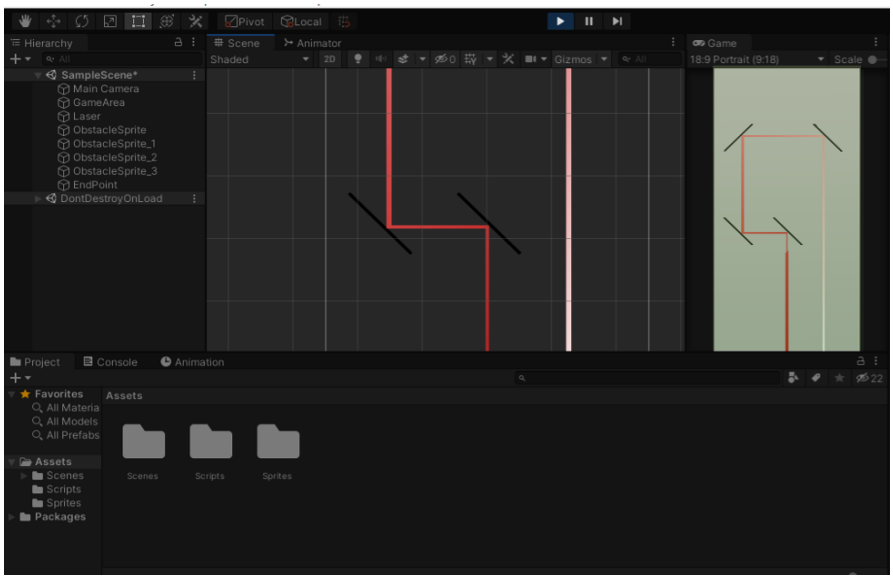

Рисунок 3.33 – Процес тестування механіки відбивання лазеру

Для керування напрямком лазера було розроблено механіку обертання примітивів, до яких було додано скрипт зміни їх положення (рис. 3.34).

У майбутньому код обертання можна модифікувати у більш складний. У додатку А розміщено лістинг скрипта обертання.

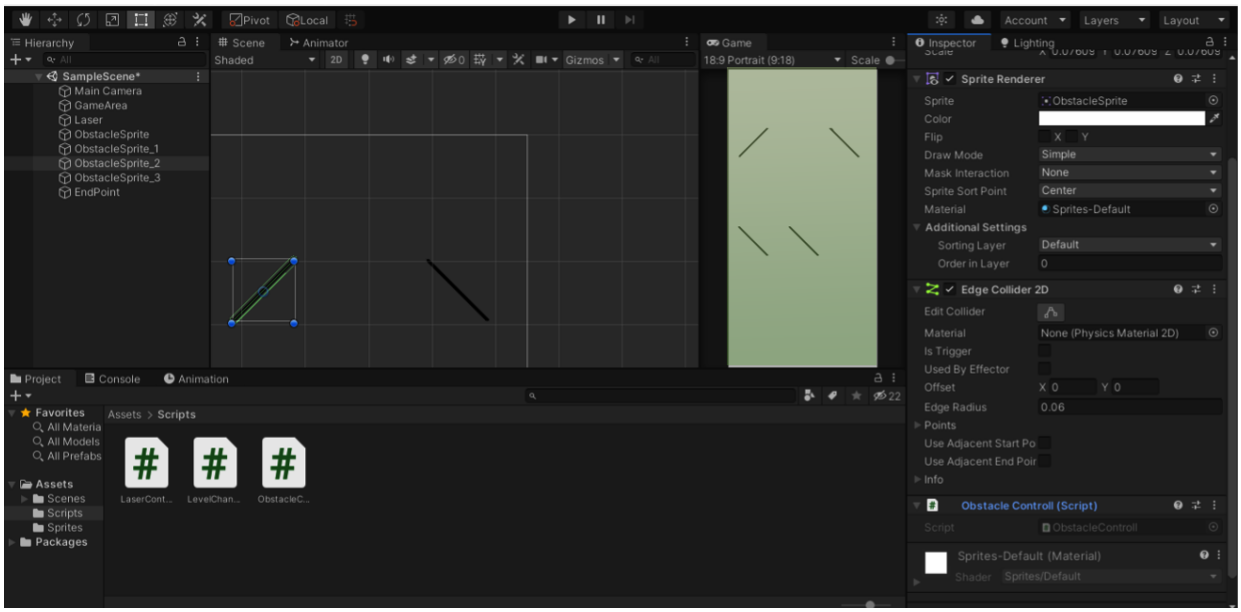

Рисунок 3.34 – Додавання до елементу скрипту

Запустимо попередній перегляд додатку, щоб протестувати даний прототип (рис. 3.35).

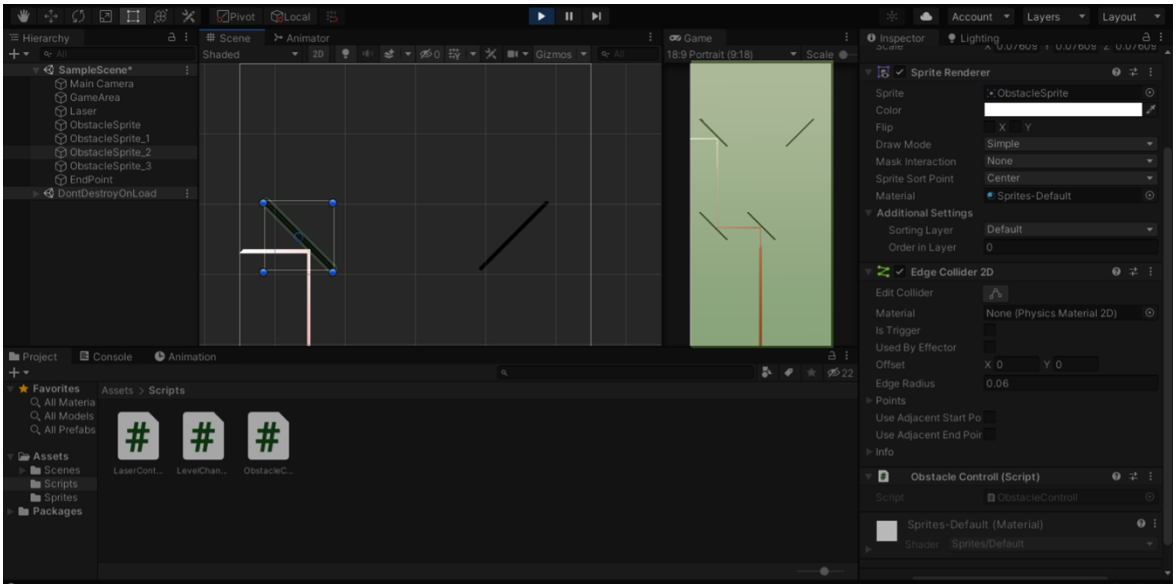

Рисунок 3.35 – Тестування прототипу

Тестування пройшло успішно. Основні ігрові механіки було розроблено. Переходимо до налаштування файлу рівнів, це важлива частиноа розробки прототипу. Модифікуємо скрипт лазеру, щоб реалізувати менеджер рівнів. На ігрове поле додаємо невидимий колайдер. Перехід на наступний рівень відбувається при перетині колайдеру лазером. На рис. 3.36 показано створення колайдеру.

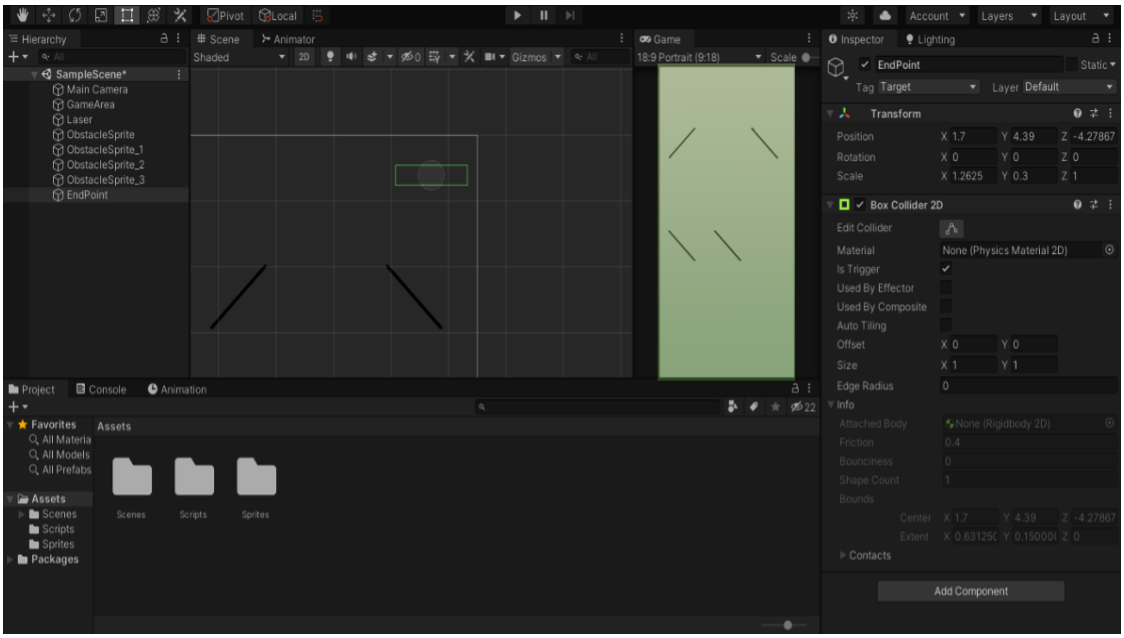

Рисунок 3.36 – Додавання колайдеру

Наступним кроком налаштовуємо збірку та додаємо новий рівень (рис.3.37).

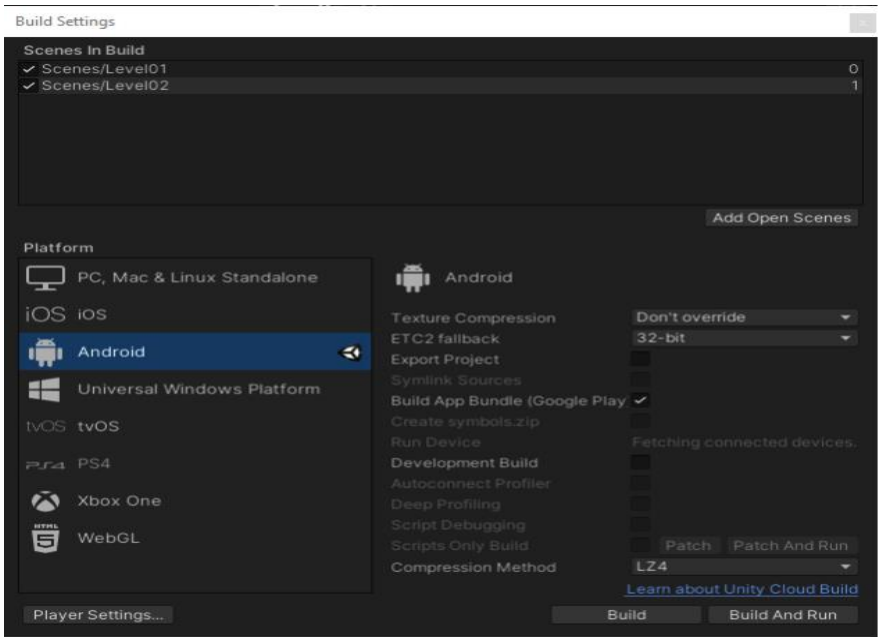

Рисунок 3.37 – Зміна налаштувань

Модифікуємо скрипт лазеру для завантаження наступного рівня таким чином, щоб рівень змінювався (рис. 3.38) після перетиннання колайдера з відповідним тегом «Target».

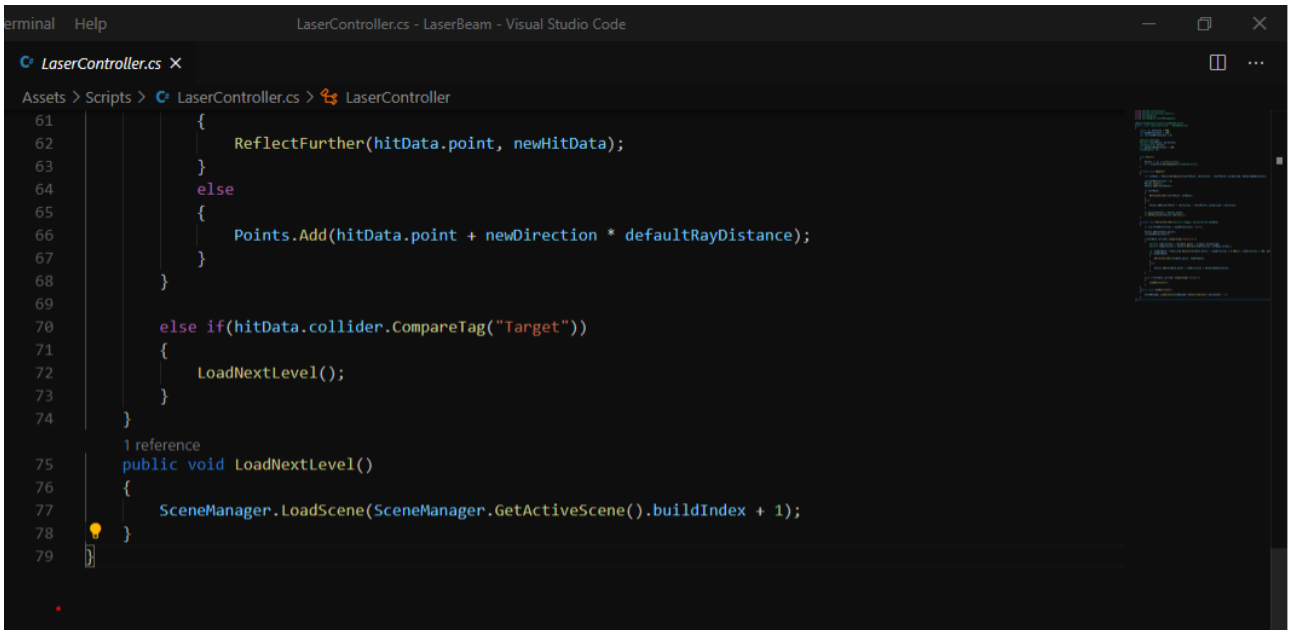

Рисунок 3.38 – Модифікація коду скрипта лазера

Висновки до розділу 3

У третьому розділі було розроблено дизайн інтерфейсу гри, проведено аналіз та обґрунтування вибору засобів та технологій розробки програмного продукту. Unity був обраний у якості ігрового рушія через ряд переваг. При вирішенні поставленої задачі оптимально використовувати мову C#, оскільки вона є мовою високого рівня і дає можливість швидко та ефективно створювати програми. Таким чином, для виконання поставлених задач найкращим середовищем програмування було обрано Visual Studio Code. Програма Adobe Illustrator використана для розробки спрайтів, а плагін Google AdMob використаний для подальшого встроювання реклами.

Було розроблено прототип гри, що є важливим та необхідним етапом розробки. На основі ігрового прототипу, який є необхідним для тестування механіки і геймплея гри, можна планувати наступні етапи роботи та мати відправну точку усього проєкту.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Діаграма класів програмного забезпечення

Діаграми класів – це такі діаграми, які відображають класи системи, їх атрибути, зміст та відношення, методи і взаємозв'язок між ними. Дані діаграми є одними з основних UML-діаграм і відображають статичне представлення структури моделі. Діаграми класів можуть містити позначення для пакетів та для вкладених пакетів. У термінології класів ООП для представлення статичної структури моделі системи служать діаграми класів. На них відображуються класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

Дані діаграми використовуються на різних етапах проектування і будуються з різним ступенем деталізації.

Діаграми класів можуть використовуються при прямому та зворотному проєктуванні, тобто у першому випадку у процесі розробки нової системи, а у другому – при описі існуючих та використовуваних систем [11].

У вихідному коді програми безпосередньо відображається інформація з діаграми класів. Кодогенерація для певних мов програмування (C++ чи Java) можлива в більшості існуючих інструментів UML-моделювання. Отже, можна зазначити, що діаграми класів представляють собою відправну точку процесу розробки та кінцевий результат проєктування [12].

У проєктуванні об'єктно-орієнтованих систем діаграми класів займають центральне місце. На різних етапах проєктування використовується нотація класів. Мова UML також застосовується для створення ескізів проєктів.

Основними елементами є класи і зв'язку між ними. За допомогою атрибутів і операцій характеризуються класи.

Властивості об'єктів класу описують атрибути. У класі більшість об'єктів мають індивідуальність через відмінності в їх атрибутах і взаємозв'язках з іншими об'єктами. У межах класу ім'я атрибуту має бути

унікальним. За ім'ям атрибута може слідувати його значення за замовчуванням та тип.

Діаграма основних класів для гри складається з чотирьох класів, а саме:

- Клас Player. Створює об’єкти взаємодії з грою, які мають можливість вносити в неї зміни. Має два приватні атрибути – логін і пароль. Авторизація гравця відображається одним публічним атрибутом – + loged_In. Операторами даного класу є: - get_login (),-set_login (),-get_pass (),-set_pass (),-authorization (),-registration ().

- Клас Game. Його атрибути відповідають за правильний доступ до параметрів ігрового поля та чергу ходу. Клас відповідає за зміну параметрів поля, виконання ігрової логіки та взаємодію між ігровим полем та гравцем.

- Клас Field. Ігрове поле формують та контролюють стан комірок параметри, які має клас Field. Під впливом класу Game клас Field змінює статуси клітинок, що характеризують належність гравця. Він має композиційний зв’язок з класом Game. Коли знищується другий, руйнується і перший.

- Клас Statatistic. Коли надходить запит на відображення статистики, вся інформація може бути доступна в будь-який час, оскільки клас Game має доступ до класу Statastic. Даній клас має лише один приватний параметр, що надає відповідь користувачеві на запит зпівставлення з його інформаційним полем у БД.

Діаграма класів для проекту розробки представлено на рис. 4.1.

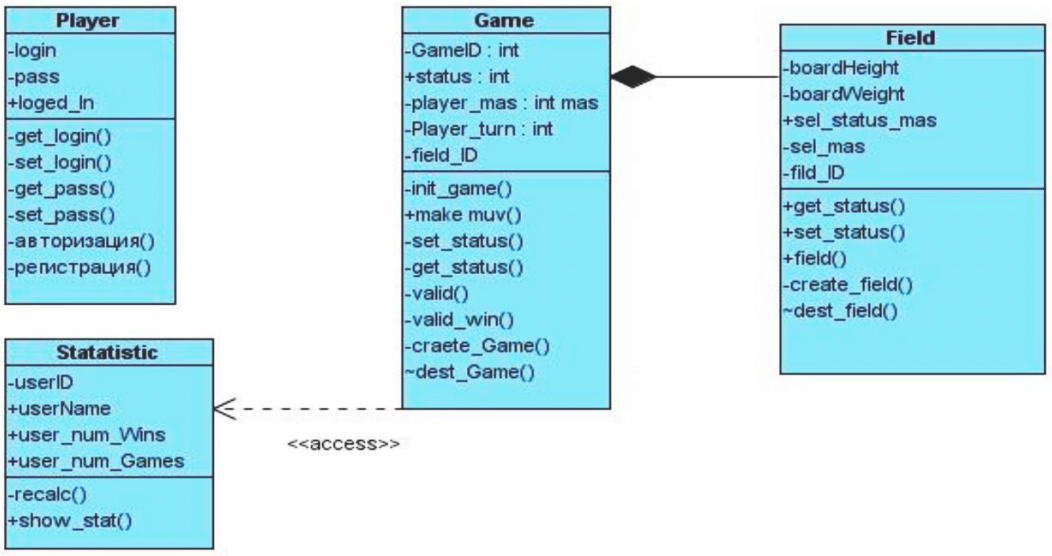

Рисунок 4.1 – Діаграма класів програмного забезпечення

4.2 Тестування комп’ютерної гри

Тестування є одним із розділів діагностики. Тестування використовується, щоб визначити, чи відповідає випробуваний елемент зазначеним специфікаціям. Визначення причин невідповідності зазначеним вимогам не входить у завдання тестування. Тестування використовується, щоб визначити чи придатний об’єкт тестування для виконання певних функцій. Від тестера багато в чому залежить достовірність тестування та якість його результатів.

Технології тестування мають у своєму складі такі частини:

- зовнішньої дії;
- реакцію суб'єкта;
- оцінка реакції та висновки.

Неодноразово проводяться випробування, з різними зовнішніми діями, поки тестувальник не прийме рішення щодо придатності випробуваного програмного продукту для виконання зазначених функцій.

Тестування програмного забезпечення [22] – це процес перевірки ПЗ на якість продукту та пошук у ньому багів.

Сучасні методи тестування ПЗ не дозволяють повністю та однозначно виявити всі дефекти та виявити коректне функціонування розглянутої

програми, тому всі наявні методи тестування діють в рамках офіційного процесу верифікації програмного забезпечення, що досліджується або розробляється.

З точки зору використаного методу, такий процес верифікації може довести, що немає недоліків. Таким чином, неможливо точно встановити чи гарантувати відсутність дефектів у програмному продукті, враховуючи людський фактор, який присутній на всіх етапах життєвого циклу ПЗ.

До вирішення проблеми тестування та верифікації програмного засобу існує багато підходів. але ефективне тестування складних програмних продуктів є надзвичайно творчим процесом, який не зводиться до дотримання суворих і чітких процедур [23].

Класифікувати види тестування прийнято за наступними критеріями:

За об’єктом тестування: функціональне тестування, тестування продуктивності, навантажувальне тестування, стрес-тестування, тестування стабільності, тестування юзабіліті, тестування UI, тестування безпеки, тестування локалізації, тестування на сумісність [24].

В якості компонентів виділяють: компонентне (модульне) тестування, інтеграційне тестування, тестування системне.

Таким чином, мобільний додаток було протестовано після успішного проходження усіх етапів розробки.

Спочатку протестовано головне меню, його дисплей, анімація, функціональність кнопок (рис. 4.2).

Рисунок 4.2 – Головне меню

Наступним кроком є перевірка меню вибору рівня гри, коректність відображення доступних рівнів та функціонування кнопок (рис. 4.3).

Рисунок 4.3 – Меню вибору рівня гри

У меню результатів проходження перевірено правильне відображення результатів та коректність функціонування кнопок (рис. 4.4).

Рисунок 4.4 – Меню для перегляду результатів проходження

Завершивши повне тестування інтерфейсів головного меню переходимо до перевірки інтерфейсів ігрового меню та рівнів гри в цілому.

Спочатку тестуємо інтерфейс на рівні гри, враховуючи кнопку паузи та текст, який показує значення кількості наявних ходів (рис. 4.5).

Далі перевіряємо коректне відображення спрайтів на екрані.

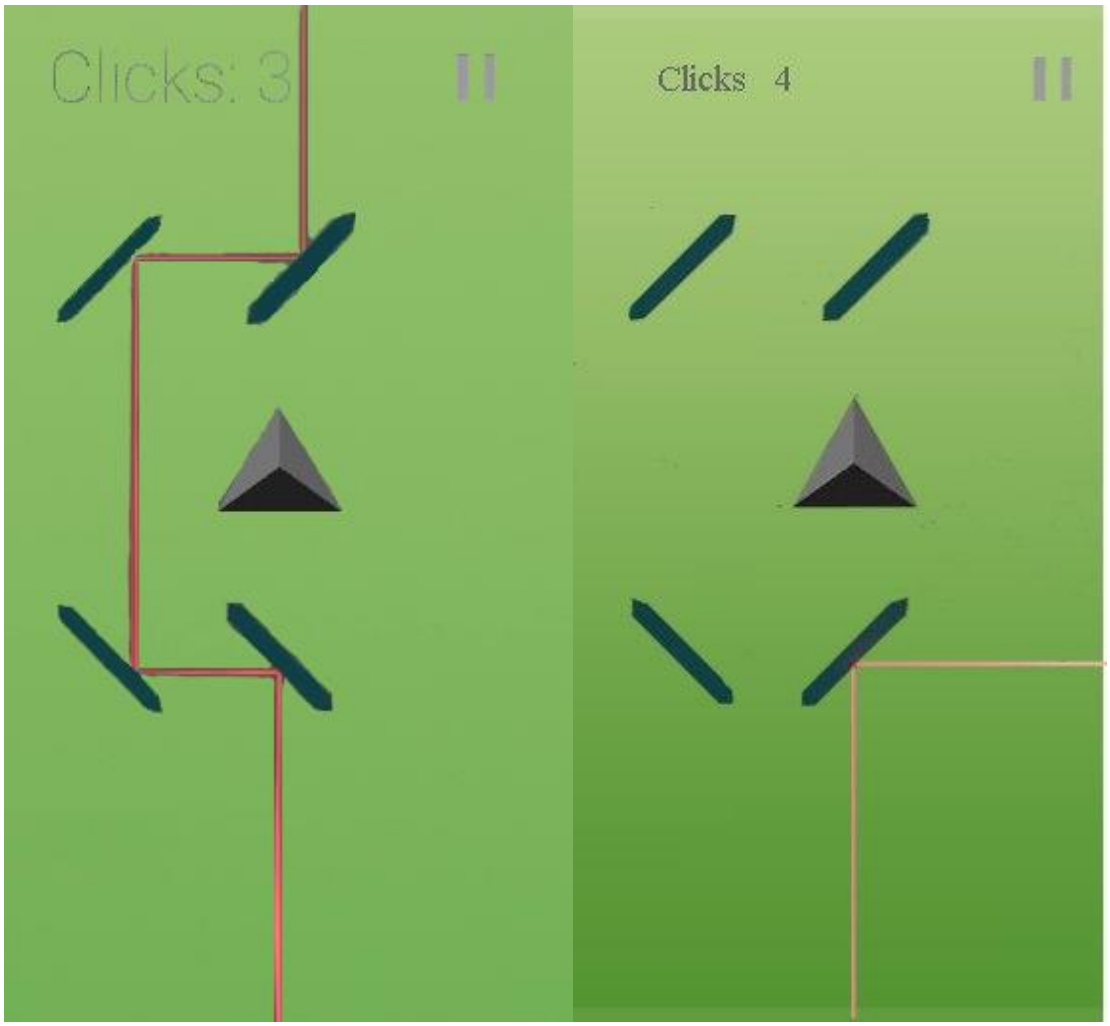

Рисунок 4.5– Інтерфейс рівнів гри

Наступним етапом є перевірка інтерфейсу меню паузи та роботу його складових [25]. У даному меню анімація та кнопки функціонують коректно (рис. 4.6).

Перевіряємо поведінку програми під час виклику меню паузи. Коли меню паузи на екрані, час гри зупиняється, що повністю відповідає завданню.

Рисунок 4.6 – Інтерфейс ігрового рівня

Тестуємо меню, яке відповідає за невдале проходження рівня. Кнопки та анімація функціонують коректно, меню відображається правильно (рис. 4.7).

Рисунок 4.7 – Інтерфейс меню у разі невдалого проходження

Перевіряємо меню успішного проходження рівня. Кнопки та анімація функціонують коректно, меню відображається правильно (рис. 4.8).

Меню викликається коректно, правильно відображаючи поточну оцінку за пройдений рівень.

Рисунок 4.8 – Інтерфейс меню у разі успішного проходження

Після проведення тестування всіх інтерфейсів починаємо тестувати основну ігрову механіку та перевіряємо як працює меню зберігання результатів [26].

На рис. 4.9, рис. 4.10 та рис. 4.11 наведено результати перевірки роботи шляху проходження першого рівня гри.

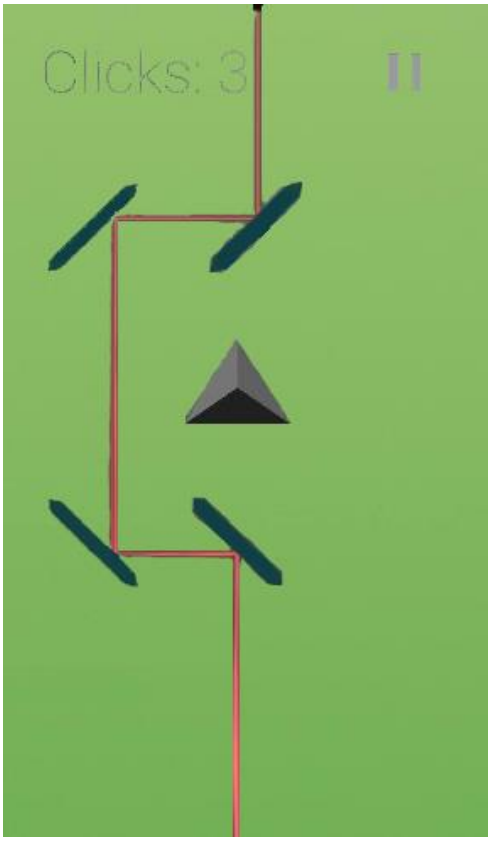

Рисунок 4.9– Перший варіант проходження першого рівня гри

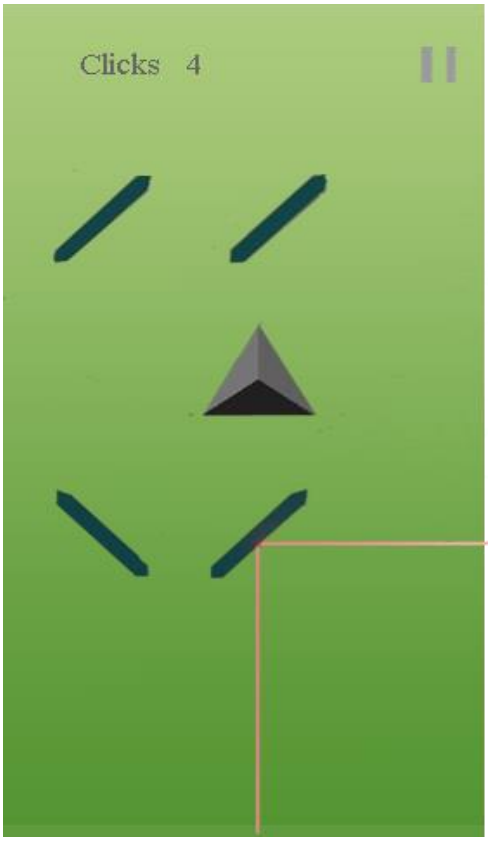

Рисунок 4.10 – Другий варіант проходження першого рівня гри

Рисунок 4.11 – Загальний результат проходження першого рівня

Перевіряємо як зберігається прогрес проходження гри. Умовно поділяємо цей процес на два етапи. На першому перевіряємо збереження результатів у меню (рис. 4.12), а на другому – у меню з вибором рівня гри тестуємо як зберігається прогрес (рис. 4.13).

Рисунок 4.12 – Меню результатів проходження усіх рівнів гри

У меню результатів збережено та правильно відображається результат проходження рівня гри.

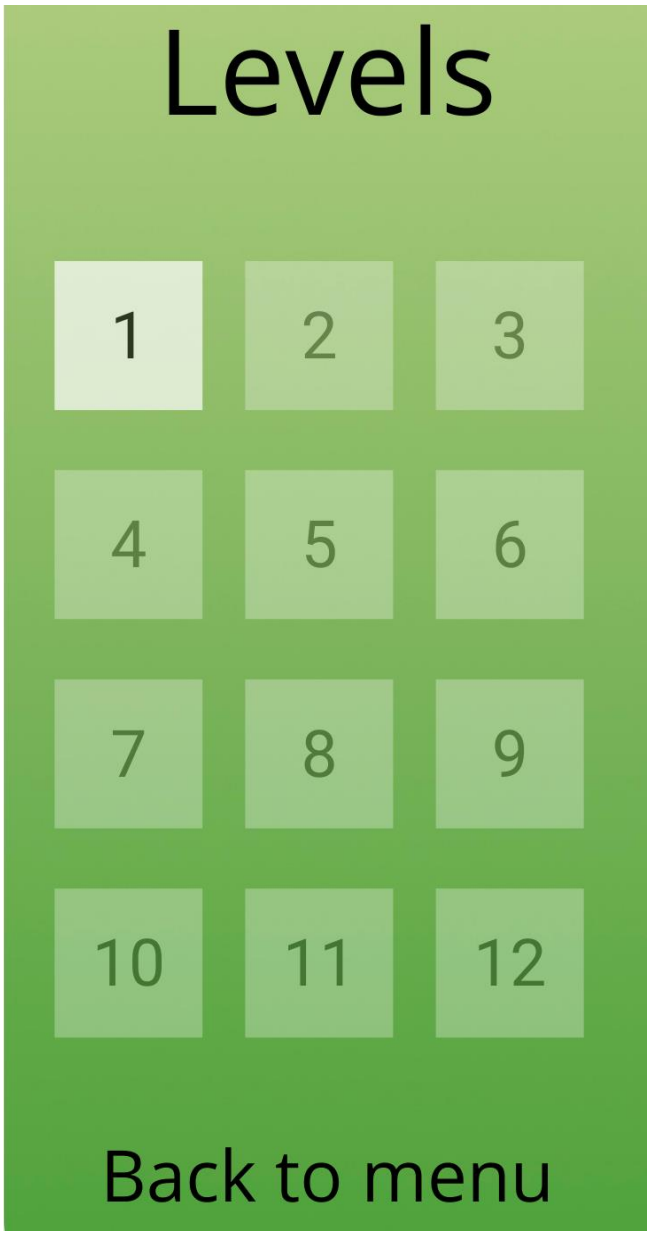

Рисунок 4.13 – Розблоковані рівні у меню вибору рівня гри

Гравець отримує доступ до другого рівня після успішного завершення першого.

Функціональні кнопки, анімації та механіки працюють коректно, без збоїв.

Таким чином, у ході даного тестування було перевірено відповідність програмного продукту функціональним вимогам.

4.3 Інструкції користувача

Дана гра написана у жанрі – головоломка, для вирішення якої, потрібна кмітливість, а не тільки спеціфічні знання високого рівня. Мобільний застосунок призначений для розвитку логічного мислення користувача, а також для забезпечення веселого та корисного проводження часу у грі [27]. Крім того, однією із задач є підвищення швидкості та реакції через встановлені у додатку ігрові обмеження.

Створювана гра має наступний функціонал:

- вибір рівня гри;
- проходження гри;
- перегляд рахунку за пройдені ігри.

Мобільний інтерфейс гри задовольняє всі вимоги UX/UI, є інтуїтивно зрозумілим, що дозволяє користувачу витрачати менше часу на розуміння та налаштування гри. Орієнтація гри при запуску була налаштована розробником для її коректного відображення на екрані мобільного пристрою. У якості стандартної орієнтації розробником у налаштуваннях проєкту було встановлено портретну, без можливості зміни (рис. 4.14).

Гра складається із декількох частин: головне меню, екран відображення рівнів, відображення одного з рівнів та екран результатів після проходження гри. Поетапно взаємодія користувача із застосунком відбувається наступним чином:

- ініціалізація гравця у завантаженому мобільному застосунку;
- обрання доступного рівня гри;
- проходження головоломки;
- перегляд результату після пройденого рівня гри.

Гра має один режим – за рівнями. Даний режим надає можливість переглянути особистий рекорд та почати гру. У такому випадку гра може тривати доти, поки користувач буде успішно проходити рівні, так як після досягнення цілі рівня запускається наступний рівень зі збільшенням складності. Найбільший останній номер рівня і є персональним рекордом. Він записується у пам’ять, коли гравець покидає рівень.

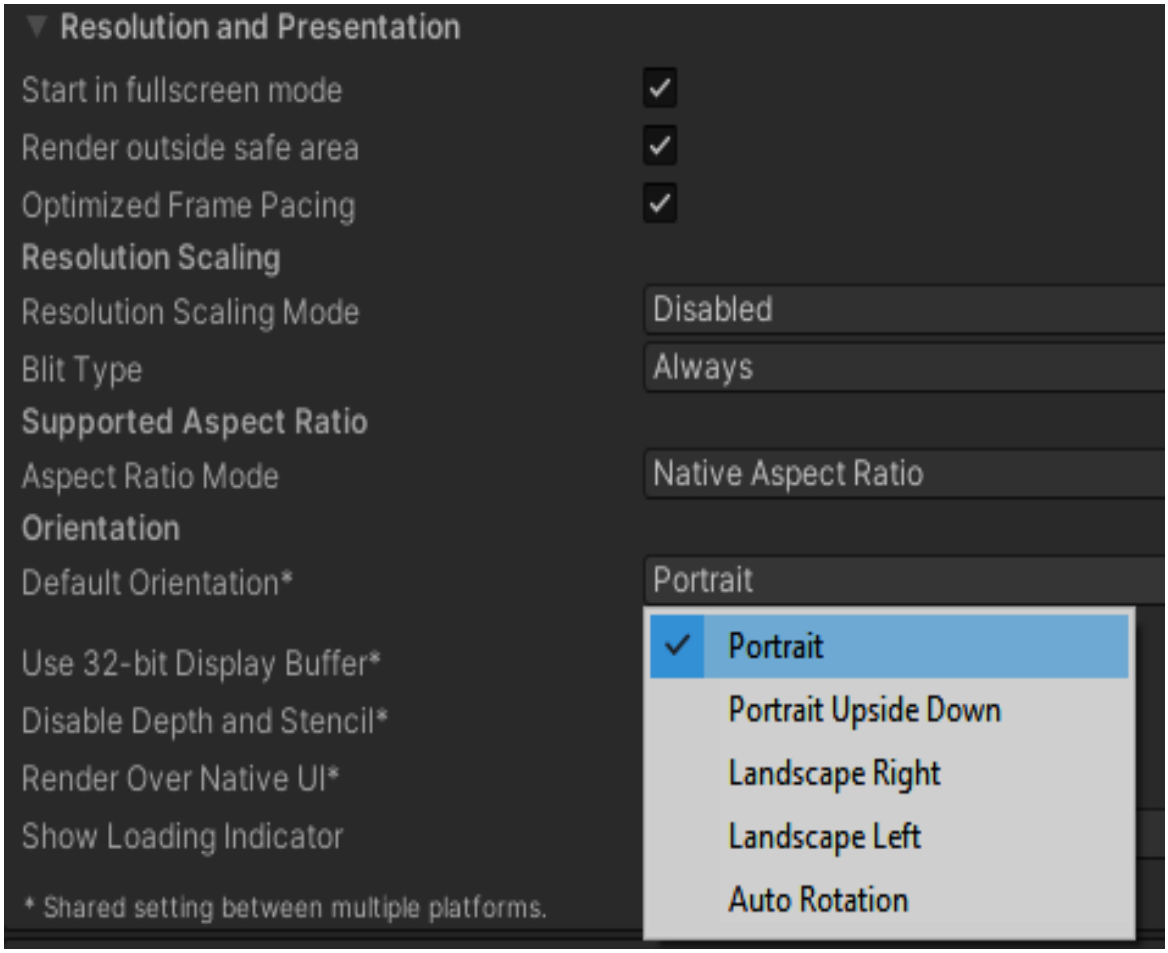

Рисунок 4.14 – Налаштування орієнтації додатку

Оскільки механіка головоломки є основною механікаою додатку, то гра реалізовується у через задачі, які повинен вирішити гравець. За задачу у реалізованій гри гравцю потрібно правильно направити промінь лазеру рухаючи люстерка (дзеркала), щоб лазео влучив у ціль (рис. 4.15).

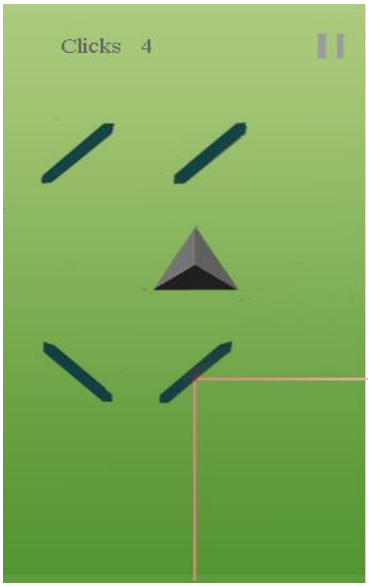

Рисунок 4.15 – Проходження рівня гри

Після проходження рівня відкривається можливість переходу на наступний рівень з більш складним завданням, де вказана інша доступна гравцю кількість натискань.

Гравцеві доступна можливість вибрати рівень проходження гри. Якщо гравець почав нову гру, доступний тільки перший рівень, після проходження якого стає доступним другий і так далі. Якщо гравець продовжує збережену гру, то він може вибрати будь-який з уже пройдених рівнів або останній доступний, після проходження якого стає доступним наступний [28].

Варто зауважити, що мобільний ігровий застосунок безкоштовній, отже кожен користувач зможе пройти гру.

Висновки до розділу 4

У даному розділі було продемонстровано загальну структуру ієрархії класів системи, її атрибутів (полів), методів, інтерфейсів і взаємозв'язків між ними. У ході тестування було перевірено відповідність програмного продукту функціональним вимогам. Тестування було успішно виконано. Дана гра повністю відповідає всім відповідним вимогам. Також було наведено інструкцію користування додатком.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено ігровий застосунок в жанрі головоломка. Проведено огляд існуючих мобільних ігрових застусунків з різними механіками. Розглянуті ігрові застосунки, які містять механіку головоломка. Проаналізовані основні переваги та недоліки оглянутих мобільних ігрових застосунків. Визначено, що більшість являються комерційними розробками. Сформульовані задачі досліджень дипломної роботи та основні вимоги щодо створення програмного забезпечення.

Розлянуто основні функціональні можливості ігрового застосунку: можливість вибору рівня гри, складності рівня, проходження гри та перегляд очок після проходження. Блок-схемою схематично зображено процес роботи мобільного ігрового застосунку. Також створено UML діаграму прецедентів, де відображена взаємодія користувача з усіма частинами застосунку. Окрім діаграми прецендентів було представлено діаграму розгортання, що допомагає раціонально організувати компоненти, від цього залежить продуктивність системи та безпека.

Розроблено дизайн інтерфейсу гри, проведено аналіз та обґрунтування вибору засобів та технологій розробки програмного продукту, таким чином був обраний основний інструментарій, необхідний для реалізації проєкту. У ході роботи над кваліфікаційною було зроблено прототип гри, з урахуванням бажаної механіки, на основі якого на платформі Unity було створено ігровий мобільний додаток «Laser» із використанням механіки головоломки. Продемонстровано загальну структуру ієрархії класів системи, її атрибутів, інтерфейсів і взаємозв'язків між ними. Задля досягнення мети у проєкті були створені ігрові інтерфейси та розроблені ігрові рівні.

У ході тестування було перевірено відповідність програмного продукту функціональним вимогам. Тестування було успішно виконано. Розроблена гра повністю відповідає всім відповідним вимогам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 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 23.04.22).

2. 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.

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

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

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

6. Hyper-casual games: mobile gaming’s greatest genre: веб-сайт. URL: https://clevertap.com/blog/hyper-casual-games/ (дата звернення 26.04.22).

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

8. Lovell N. The pyramid of game design: designing, producing and launching service games. CRC Press; 1st Edition, 2018. 340 с.

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

10. Atkins M. F. Design a game (rookie get ready to code). Children's press, 2019. 32 с.

11. Методичні рекомендації до виконання кваліфікаційних робіт студентами спеціальності 121 «Інженерія програмного забезпечення» ступеня вищої освіти «Бакалавр» / Фісун М. Т. та ін. Миколаїв: ЧНУ ім. П. Могили, 2020. 73 с.

12. Unified Modeling Language (UML) | An Introduction. URL: у: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/ (дата звернення: 03.05.2022).

13. J. Schell The Art of Game Design. Carnegie Mellon University, 2008. 518 p.

14. Fullerton T., Swain Ch., Hoffman S. Game design workshop: a playcentric approach to creating innovative game. 2nd ed. Morgan Kaufmann, 2008. 491 p.

15. Hocking J. Unity in Action: Multiplatform Game Development in C# with Unity 5. 1st Edition. Manning Publications, 2015. 352 p.

16. UNet Overview. URL: https://docs.unity3d.com/Manual/UNet.html (дата звернення 06.05.22).

17. Фурсова Н. А., Козак О. Є. Розробка мережевої комп'ютерної гри з використанням Unity Engine. Тези 70-ої ювілейної наук. конф. проф., викл., наук. прац., аспір. та студ. університету. Том 2 / ПолтНТУ, Полтава, 23 квітня – 18 травня 2018 р. Полтава: ПолтНТУ, 2018. С. 244–245.

18. Троэлсэн Э., Джепикс Ф. Язык программирования C# 6.0 и платформа .NET 4.6. М.: Вильямс, 2016. 1440 с.

19. Albahari J. C# 7.0 in a Nutshell: The Definitive Reference. 1st ed. Sebastopol: O'Reilly Media, 2017. 1090 c.

20. Android - Architecture / Tutorialspoint. URL:

http://www.tutorialspoint.com/android/android_architecture.htm (дата звернення 09.05.22).

21. The Perfect Platform for Game Developers. Android URL: http://www.developer.com/ws/android/client/the-perfect-platform- for-gamedevelopers-android.html (дата звернення 09.05.22).

22. Testing Guide. URL: https://owasp.org/www-project- web-security-testing-guide/ – Посібник з тестування (дата звернення 10.05.22).

23. Лайза Криспин, Джанет Грегори Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. М.: «Вильямс», 2010. 464 с.

24. Голощапов А. Л. Google Android програмування для мобільних пристроїв. М.: BHV Санкт-Петербург, 2011. 1549 с.

25. Хашимі С. Розробка додатків для Android. М.: Біном, 2011. 2125 с.

26. Тестування програмного забезпечення. Основні поняття і визначення. URL: http://www.protesting.ru/testing/. (дата звернення: 12.05.2022).

27. Software Testing. URL: https://www.tutorialspoint.com/software_testing/software_testing_lev els.htm. (дата звернення: 14.05.2022).

28. Software Testing - API testing. URL: http://www.tutorialspoint.com/software_testing_dictionary/api_testin g.htm. (дата звернення: 14.05.2022).

ДОДАТОК

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();

        }

    }

}
```