

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА

Інститут комп'ютерних систем

Кафедра інформаційних систем

**МЕТОДИЧНІ ВКАЗІВКИ
ДЛЯ ПРОТОТИПУВАННЯ КОМП'ЮТЕРНИХ ІГОР
З ІГРОВІЗАЦІЇ ФІЗИЧНИХ ВПРАВ
НА БАЛАНСУЮЧИХ ДОШКАХ ЗІ СМАРТФОНОМ
У СЕРЕДОВИЩІ UNITY 3D**

Дар'я ШУМЕЙКО

(група АІ-201)

Керівник:

Олександр БЛАЖКО, к.т.н., доцент

Одеса – 2026

ЗМІСТ

1 ОГЛЯД КОМП'ЮТЕРНОЇ ІГРОВІЗАЦІЇ ФІЗИЧНИХ ВПРАВ НА БАЛАНСУЮЧИХ ДОШКАХ	4
1.1 Багатокритеріальна класифікація фізичних вправ на балансуючих дошках	4
1.2 Багатокритеріальна класифікація комп'ютерних ігор вправ на балансуючих дошках	6
1.2.1. Класифікація ігрових сценаріїв за положенням тіла користувача.	6
1.2.2. Класифікація ігрових сценаріїв за характером руху.....	7
1.2.3. Класифікація ігрових сценаріїв за ціллю тренування.....	7
1.2.4. Класифікація ігрових сценаріїв за рівнем складності.....	8
1.3 Узагальнення класифікацій фізичних вправ та ігрових сценаріїв	9
1.4 Принципи побудови ігрових сценаріїв для фізичних вправ на балансуючих дошках	10
1.4.1. Вибір ігрової механіки та способу керування.	10
1.4.3. Система нарахування балів і штрафів.	11
1.4.4. Регулювання рівня складності ігрового сценарію.....	11
1.4.5. Відповідність ігрового сценарію фізичному навантаженню.	11
1.5 Приклад реалізації ігрового сценарію та його класифікація.....	12
1.5.1 Опис ігрового сценарію.....	12
1.5.2 Класифікація ігрового сценарію.....	13
1.5.3 Узагальнення прикладу.....	15
2 СТВОРЕННЯ ПРОТОТИПУ КОМП'ЮТЕРНОЇ ГРИ З ІГРОВІЗАЦІЇ ФІЗИЧНИХ ВПРАВ НА БАЛАНСУЮЧИХ ДОШКАХ	16
2.1 Опис сценарію комп'ютерної гри	16
2.2 Ознайомлення з Unity	17
2.3 Встановлення та первинне налаштування Unity.....	18
2.3.1 Завантаження та встановлення Unity Hub.....	18
2.3.2 Вибір та встановлення версії Unity.	20
2.3.3 Налаштування середовища для мобільної розробки.....	23

2.3.4 Підготовка до подальшої роботи	24
2.4 Створення нового 3D-проєкту в Unity	25
2.4.1 Створення нового проєкту	25
2.4.2 Ознайомлення з інтерфейсом Unity Editor	27
2.4.3 Базові об'єкти та компоненти 3D-сцени.	30
2.4.4 Підготовка структури проєкту.	33
2.4.5 Збереження сцени.	33
2.5 Підготовка середовища розробки та створення проєкту	34
2.6 Початкове налаштування сцени, камери та параметрів відображення для Android	35
2.7 Організація структури проєкту, імпорт моделей та налаштування матеріалів	45
2.8 Налаштування фізики ігрового персонажа	51
2.9 Створення маршруту тунелю за допомогою сплайна	57
2.10 Реалізація руху персонажа вздовж сплайна	60
2.11 Налаштування камери для слідування за персонажем.....	65
2.12 Обробка зіткнень і визначення помилок гравця.....	67
2.13 Реалізація системи підрахунку балів	72
2.14 Створення стартового екрану гри	77
2.15 Відображення рахунку під час гри.....	84
2.16 Завершення рівня та створення фінального екрану гри	90
2.17 Побудова apk-модуля для завантаження на Android-смартфон.....	95
2.18 Рекомендації щодо впровадження методичних вказівок у навчальний процес	96
ВИСНОВКИ.....	97

1 ОГЛЯД КОМП'ЮТЕРНОЇ ІГРОВІЗАЦІЇ ФІЗИЧНИХ ВПРАВ НА БАЛАНСУЮЧИХ ДОШКАХ

У даному розділі розглядається підхід до комп'ютерної ігровізації фізичних вправ на балансуючих дошках. Спочатку аналізуються типи фізичних вправ та їх класифікація за руховими ознаками, після чого розглядаються відповідні ігрові сценарії та механіки. Така послідовність дозволяє перейти від аналізу фізичної активності до її практичної реалізації у вигляді комп'ютерної гри.

1.1 Багатокритеріальна класифікація фізичних вправ на балансуючих дошках

Дизайн ігрових сценаріїв у системах комп'ютерної ігровізації фізичних вправ повинен базуватися на структурованому підході до класифікації фізичної активності та ігрових механік. Такі системи поєднують елементи гри з виконанням вправ на балансуючій дощці, що вимагає врахування не лише ігрової логіки, а й особливостей рухової діяльності користувача, рівня фізичного навантаження та безпеки виконання вправ.

Для систематизації ігрових сценаріїв доцільно використовувати класифікацію, що ґрунтується на сукупності параметрів, які безпосередньо впливають на характер ігрового процесу та фізичне навантаження. До таких параметрів належать положення тіла користувача, характер руху, ціль тренування, рівень складності та особливості виконання рухів.

Положення тіла користувача визначає тип опори, ступінь стабільності та загальний рівень складності виконання вправ. Ігрові сценарії можуть передбачати виконання вправ у положенні стоячи, сидячи, на одній нозі, у планці або на колінах, що безпосередньо впливає на інтенсивність навантаження та вимоги до балансу. Вибір положення тіла є одним із ключових чинників при формуванні ігрової механіки.

Характер руху поділяється на динамічний і статичний. Динамічні сценарії орієнтовані на безперервні або ритмічні зміни положення тіла, реакцію на рухомі об'єкти та коригування траєкторій. Статичні сценарії, навпаки, зосереджені на утриманні стабільного положення з мінімальними коригувальними рухами. Цей параметр визначає темп гри та тип взаємодії користувача з ігровим середовищем.

Ціль тренування формує функціональну спрямованість гри та може включати розвиток координації, балансу, стабілізації, точності або їх поєднання. Саме тренувальна ціль задає пріоритетні рухові дії, які має виконувати користувач у процесі гри, та визначає вимоги до ігрових завдань.

Рівень складності ігрового сценарію визначається сукупністю параметрів ігрової механіки, зокрема швидкістю руху ігрових об'єктів, точністю керування, наявністю перешкод і штрафних санкцій за помилки. Виділення початкового, середнього та просунутого рівнів складності дозволяє адаптувати ігрові сценарії до різних груп користувачів.

Застосування цієї класифікації дозволяє розглядати кожен ігровий сценарій як поєднання окремих класів, що визначають структуру керування, динаміку гри та вимоги до користувача. Так, сценарії з динамічним характером руху передбачають активні нахили тіла та постійне коригування положення центру маси, тоді як статичні ігри орієнтовані на утримання стабільної пози з мінімальними рухами. Вибір положення тіла (стоячи, сидячи, у планці тощо) визначає рівень залучення м'язових груп і ступінь складності вправ.

Ціль тренування є одним з ключових чинників, що формує ігрову механіку. Ігри, спрямовані на розвиток координації, зазвичай включають зміну напрямку руху об'єктів, необхідність швидкої реакції та точного керування. Сценарії, орієнтовані на баланс або стабілізацію, зосереджуються на плавності рухів, точності утримання положення та поступовому ускладненні умов виконання. Рівень складності при цьому може регулюватися за рахунок швидкості ігрових об'єктів, чутливості керування або системи заохочень і штрафів.

Таким чином, дизайн ігрового сценарію формується на основі вибраних класифікаційних ознак і має відповідати їх поєднанню. Такий підхід забезпечує узгодженість між ігровою частиною та фізичною активністю, дозволяє прогнозувати рівень навантаження та адаптувати гру до різних категорій користувачів. У подальших підрозділах на основі цієї класифікації будуть сформульовані практичні рекомендації з дизайну ігрових сценаріїв і наведено приклад їх реалізації.

1.2 Багатокритеріальна класифікація комп’ютерних ігор вправ на балансуючих дошках

Для коректного проєктування ігрових сценаріїв у системах комп’ютерної ігровізації фізичних вправ доцільно використовувати багатопараметричну класифікацію, що враховує як особливості рухової активності користувача, так і характеристики ігрового процесу. Такий підхід дозволяє формалізувати вимоги до дизайну гри та забезпечити відповідність ігрових механік поставленим тренувальним цілям.

У межах даної методики ігрові сценарії розглядаються за такими основними класифікаційними ознаками: положення тіла користувача, характер руху, ціль тренування та рівень складності.

1.2.1. Класифікація ігрових сценаріїв за положенням тіла користувача.

Положення тіла користувача визначає базовий рівень стабільності та складності виконання ігрових завдань. У системах з балансуючою дошкою можуть застосовуватися такі основні положення тіла:

- стоячи — базове положення, що забезпечує природну опору та використовується в більшості ігрових сценаріїв, орієнтованих на розвиток координації та балансу;
- стоячи на одній нозі — ускладнений варіант, який підвищує вимоги до стабільності та активує м’язи-стабілізатори;

- сидячи — положення зі зниженою складністю, що використовується для початкових рівнів або реабілітаційних сценаріїв;
- у планці — положення з підвищеним силовим навантаженням, яке поєднує балансування з роботою м'язів корпусу;
- на колінах або лежачи — застосовується в спеціалізованих сценаріях, орієнтованих на контроль рухів та зниження навантаження на нижні кінцівки.

Вибір положення тіла є одним з ключових параметрів при визначені рівня складності гри та типу фізичного навантаження.

1.2.2. Класифікація ігрових сценаріїв за характером руху.

За характером рухової активності ігрові сценарії поділяються на динамічні та статичні.

Динамічні ігрові сценарії передбачають безперервні або ритмічні зміни положення тіла користувача, активні нахили та переміщення центру маси. Такі сценарії зазвичай включають керування рухомими об'єктами, уникання перешкод або проходження трас із змінною траєкторією.

Статичні ігрові сценарії зосереджені на утриманні стабільного положення протягом певного часу з мінімальними коригувальними рухами. Основна увага в таких іграх приділяється точності позиціонування та здатності підтримувати рівновагу.

Характер руху визначає динаміку ігрового процесу та інтенсивність фізичного навантаження, що має враховуватися під час розробки механік керування та системи оцінювання результатів і тривалість ігрових сесій.

1.2.3. Класифікація ігрових сценаріїв за ціллю тренування.

Ціль тренування визначає функціональне призначення ігрового сценарію та характер виконуваних вправ. Основними цілями тренування є:

- координація — розвиток здатності узгоджувати рухи тіла у відповідь на зміну ігрових умов;
- баланс — підтримання стійкого положення тіла при змінному навантаженні;

- стабілізація — утримання фіксованої пози з мінімальними відхиленнями;
- точність — виконання дрібних коригувальних рухів для досягнення заданої позиції;
- комбіновані цілі — поєднання кількох тренувальних завдань у межах одного сценарію.

Обрана ціль тренування визначає тип ігрових завдань, механіку керування та систему оцінювання результатів.

1.2.4. Класифікація ігрових сценаріїв за рівнем складності.

Рівень складності ігрового сценарію формується на основі сукупності параметрів ігрової механіки та умов виконання вправ. Виділяють такі рівні складності:

- початковий — характеризується низькою швидкістю руху об'єктів, широкими допусками на помилки та відсутністю жорстких штрафів;
- середній — передбачає підвищення темпу гри, зменшення допустимих відхилень та введення системи штрафів;
- просунутий — характеризується високою швидкістю, складною траєкторією руху об'єктів та підвищеними вимогами до точності керування.

Регулювання рівня складності дозволяє адаптувати ігрові сценарії до індивідуальних можливостей користувачів та забезпечити поступове ускладнення тренувального процесу.

Таблиця 1.1 - Таблиця узагальнення класифікацій

Класифікаційна ознака	Класи
Положення тіла	стоячи, сидячи, на одній нозі, у планці
Характер руху	динамічний, статичний
Ціль тренування	баланс, координація, стабілізація
Рівень складності	початковий, середній, просунутий

Комбінування наведених класифікаційних ознак дозволяє описати будь-який ігровий сценарій у формалізованому вигляді та використати цей опис як основу для розробки дизайну гри. Такий підхід забезпечує методичну

послідовність, прогнозованість фізичного навантаження та можливість адаптації ігрових сценаріїв до різних груп користувачів.

1.3 Узагальнення класифікацій фізичних вправ та ігрових сценаріїв

Для розроблення ігрових сценаріїв у системах комп’ютерної ігровізації фізичних вправ доцільно розглядати класифікаційні ознаки не ізольовано, а як взаємопов’язаний набір параметрів, що в сукупності визначають дизайн гри. Кожен ігровий сценарій може бути описаний через комбінацію положення тіла користувача, характеру руху, цілі тренування та рівня складності, що дозволяє формалізувати процес проєктування ігрової механіки.

Положення тіла користувача виступає базовим параметром, який визначає тип опори, допустимий діапазон рухів і загальний рівень стабільності під час гри. На його основі обирається характер взаємодії з ігровим середовищем, зокрема напрямки керування, чутливість нахилів та допустима амплітуда рухів. Зміна положення тіла автоматично вимагає коригування інших елементів дизайну гри.

Характер руху визначає динаміку ігрового процесу та темп виконання завдань. Для динамічних сценаріїв характерні безперервні зміни ігрових умов, що потребують швидкої реакції користувача та активного переміщення центру маси. У статичних сценаріях акцент робиться на точності та утриманні стабільної пози, що зумовлює інший підхід до формування ігрових завдань і системи оцінювання результатів.

Ціль тренування задає функціональну спрямованість ігрового сценарію та визначає, які рухові дії є пріоритетними. Наприклад, ігри, орієнтовані на розвиток координації, передбачають змінні траєкторії руху об’єктів і необхідність постійного коригування положення тіла, тоді як сценарії зі спрямуванням на стабілізацію потребують мінімізації рухів і високої точності утримання пози.

Рівень складності формується шляхом регулювання параметрів ігрової механіки, таких як швидкість руху ігрових об'єктів, чутливість керування, кількість перешкод та система штрафів за помилки. Збільшення складності повинно відбуватися поступово та відповідати фізичним можливостям користувача, забезпечуючи безперервність тренувального процесу.

Узагальнення класифікаційних ознак у межах одного ігрового сценарію дозволяє перейти від описового підходу до структурованого дизайну гри. Такий підхід забезпечує передбачуваний вплив ігрової механіки на фізичне навантаження, спрощує адаптацію сценаріїв до різних умов використання та створює основу для формування практичних рекомендацій з дизайну ігрових сценаріїв, які будуть розглянуті в наступному підрозділі.

1.4 Принципи побудови ігрових сценаріїв для фізичних вправ на балансуючих дошках

Проектування ігрового сценарію в системах комп'ютерної ігровізації фізичних вправ має здійснюватися з урахуванням класифікаційних ознак рухової активності та вимог до безпеки й ефективності тренування. Рекомендації, наведені в цьому підрозділі, дозволяють сформувати збалансований дизайн гри, у якому ігрова механіка відповідає обраній тренувальній цілі.

1.4.1. Вибір ігрової механіки та способу керування.

Ігрова механіка повинна безпосередньо відображати рухи користувача на балансуючій дошці. Керування рекомендується реалізовувати через нахили платформи або смартфона в основних площинах, що забезпечує інтуїтивну взаємодію та мінімізує потребу в додатковому навчанні.

Для динамічних сценаріїв доцільно використовувати безперервне керування рухом ігрового об'єкта, тоді як у статичних сценаріях акцент слід робити на фіксації положення та точності утримання заданої пози. Чутливість керування має регулюватися залежно від рівня складності гри.

1.4.2. Формування ігрового середовища та траєкторій руху

Ігрове середовище повинно створювати умови для виконання цільових рухів користувача. Для розвитку координації та балансу рекомендується застосовувати звивисті траєкторії, змінні напрямки руху та поступове ускладнення геометрії ігрового простору.

Розміщення перешкод і меж ігрового поля має стимулювати користувача до коригування положення тіла, але не створювати надмірних ризиків втрати рівноваги. Візуальні елементи повинні бути чіткими та легко розпізнаваними.

1.4.3. Система нарахування балів і штрафів.

Система оцінювання результатів є важливим мотиваційним елементом ігрового сценарію. Рекомендується нараховувати бали за безпомилкове виконання ігрових дій, зокрема за подолання відстані або утримання об'єкта в заданих межах.

Штрафи за помилки повинні бути співмірними з рівнем складності гри та не призводити до різкого зниження мотивації. Для початкових рівнів доцільно використовувати мінімальні штрафи або попередження, тоді як для середнього та просунутого рівнів можуть застосовуватися зменшення кількості балів або обмеження часу гри.

1.4.4. Регулювання рівня складності ігрового сценарію.

Зміна складності гри повинна реалізовуватися через параметри ігрової механіки, зокрема швидкість руху ігрових об'єктів, чутливість керування, кількість перешкод та точність необхідних рухів.

Рекомендується передбачити можливість поступового підвищення складності в межах одного сценарію або надання користувачу можливості самостійно обирати рівень складності. Такий підхід забезпечує адаптацію гри до індивідуальних можливостей користувача.

1.4.5. Відповідність ігрового сценарію фізичному навантаженню.

Ігровий сценарій повинен забезпечувати рівномірне та контролюване фізичне навантаження. Тривалість ігрової сесії, інтенсивність рухів та частота

змін ігрових умов мають відповідати обраній цілі тренування та рівню підготовки користувача.

Для зниження ризику перевтоми рекомендується передбачати паузи або зміну інтенсивності ігрового процесу, особливо в динамічних сценаріях.

Для ефективного проектування ігрового сценарію доцільно враховувати класифікаційні ознаки рухової активності користувача та їхній вплив на різні елементи дизайну гри. Таблиця 1.2 демонструє відповідність між ключовими ознаками та рекомендаціями щодо побудови ігрового процесу, що дозволяє систематизувати підхід до створення збалансованих сценаріїв.

Таблиця 1.2 – Відповідність класифікацій і елементів дизайну гри

Класифікаційна ознака	Вплив на дизайн гри
Положення тіла	тип керування, чутливість нахилів
Характер руху	темп гри, тип механіки
Ціль тренування	тип завдань
Рівень складності	швидкість, штрафи, перешкоди

Таким чином, використання класифікаційних ознак як орієнтира при проектуванні ігрового сценарію дозволяє узгодити фізичні дії користувача з ігровою механікою та рівнем складності. Це забезпечує оптимальну адаптацію тренування до індивідуальних можливостей та стимулює безпечне та ефективне виконання вправ у межах обраного сценарію.

1.5 Приклад реалізації ігрового сценарію та його класифікація

Для ілюстрації застосування класифікаційного підходу та рекомендацій з дизайну ігрових сценаріїв розглянемо приклад ігрового сценарію, орієнтованого на розвиток координації та балансу користувача з використанням балансуючої дошки.

1.5.1 Опис ігрового сценарію.

Ігровий процес полягає в керуванні м'ячем, який рухається по звивистому тунелю. Користувач, змінюючи кут нахилу смартфона або

балансуючої дошки, керує напрямком руху м'яча та повинен не допустити його зіткнення зі стінками тунелю. Рух м'яча є безперервним, а траєкторія тунелю змінюється протягом ігрової сесії.



Рисунок 1.1 – Схема ігрового сценарію керування м'ячем у звивистому тунелі шляхом нахилу пристрою

За кожен успішно пройдений метр тунелю без зіткнень користувач отримує один бал. У разі зіткнення м'яча зі стінкою нараховується штраф у вигляді зменшення кількості балів. Складність гри може регулюватися шляхом зміни швидкості руху м'яча та звуження тунелю.

1.5.2 Класифікація ігрового сценарію.

На основі описаних у попередніх підрозділах класифікаційних ознак даний ігровий сценарій може бути охарактеризований таким чином:

- положення тіла користувача: стоячи. Обране положення забезпечує природну опору та дозволяє реалізувати інтуїтивне керування через нахили тіла або платформи;
- характер руху: динамічний. Ігровий сценарій вимагає безперервних коригувальних рухів і постійного контролю положення центру маси;

- ціль тренування: координація та баланс. Користувач узгоджує рухи тіла з візуальними змінами ігрового середовища, підтримуючи стабільне положення під час руху м'яча;
- рівень складності: початковий – середній. Базовий рівень складності досягається при низькій швидкості руху м'яча, тоді як підвищення швидкості та зменшення допусків на помилки переводить гру до середнього рівня;
- коментар до фізичного навантаження: ігровий сценарій забезпечує безперервний контроль балансу з помірним навантаженням на м'язи-стабілізатори та координаційні механізми.

На основі описаних вище класифікаційних ознак можна представити конкретний приклад ігрового сценарію у вигляді узагальненої таблиці. Таблиця 1.3 демонструє ключові параметри сценарію, включаючи положення тіла користувача, характер руху, ціль тренування та рівень складності. Вона слугує наочним прикладом класифікації ігрових сценаріїв для балансувальних тренувальних систем.

Таблиця 1.3 – Класифікація прикладу ігрового сценарію

Параметр	Характеристика
Тип гри	динамічна балансова
Положення тіла	стоячі
Характер руху	динамічний
Ціль тренування	координація, баланс
Рівень складності	початковий–середній
Механіка	керування нахилами
Оцінювання	бали, штрафи

Після ознайомлення з таблицею 1.3 стає зрозуміло, що кожен параметр сценарію взаємопов'язаний із фізичною активністю користувача та вимогами до координації рухів. Такий підхід дозволяє систематизувати сценарії та забезпечити стандартизовану оцінку ефективності тренування. Для наочності механізму взаємодії користувача з ігровим об'єктом представлено схематичне зображення на рисунку 1.2. Воно ілюструє, як користувач впливає на положення ігрового об'єкта шляхом нахилів тіла на баланс-дошці.



Рисунок 1.2 – Схема керування ігровим об’єктом шляхом нахилів тіла користувача на баланс-дошці

Як висновок можемо бачити, що фізичні дії користувача безпосередньо трансформуються у зміну положення ігрового об’єкта. Це підкреслює інтеграцію фізичної активності та віртуального середовища, що є ключовим елементом гейміфікації тренувального процесу.

1.5.3 Узагальнення прикладу.

Розглянутий ігровий сценарій відповідає класу динамічних ігор з керуванням траєкторією руху об’єкта та має спільні риси з іграми типу проходження тунелів і лабіринтів. Його структура дозволяє легко адаптувати параметри складності та фізичного навантаження без зміни базової ігрової механіки. Таким чином, наведений приклад демонструє, яким чином класифікаційні ознаки можуть бути використані як практичний інструмент для формування дизайну ігрових сценаріїв у системах комп’ютерної ігровізації фізичних вправ. Запропонований підхід забезпечує методичну узгодженість між ігровою механікою та руховою активністю користувача і може бути використаний як шаблон для розроблення інших ігрових сценаріїв.

2 СТВОРЕННЯ ПРОТОТИПУ КОМП'ЮТЕРНОЇ ГРИ З ІГРОВІЗАЦІЙ ФІЗИЧНИХ ВПРАВ НА БАЛАНСУЮЧИХ ДОШКАХ

2.1 Опис сценарію комп'ютерної гри

У межах даної методичної роботи розробляється комп'ютерна гра, спрямована на ігровізацію виконання фізичних вправ на балансуючій дошці з використанням мобільного пристроя. Основною метою гри є тренування координації рухів, контролю балансу та стабілізації положення тіла користувача шляхом керування ігровим об'єктом за допомогою нахилів смартфона.

Ігровий сценарій побудовано навколо керування тривимірним колоподібним героєм, який рухається вперед усередині звивистого тунелю. Користувач, змінюючи кут нахилу мобільного пристроя, впливає на напрямок руху персонажу, не допускаючи його зіткнення зі стінками тунелю. Керування реалізується через сенсори смартфона, що забезпечує безпосередній зв'язок між рухами тіла користувача та поведінкою ігрового об'єкта.

Ігровий процес передбачає систему оцінювання успішності виконання вправ. За кожен відрізок пройденої дистанції без зіткнень користувачеві нараховуються ігрові бали. У випадку зіткнення м'яча зі стінками тунелю передбачено штраф у вигляді зменшення кількості набраних балів. Такий підхід стимулює користувача до точного та плавного керування рухами, що позитивно впливає на тренування координації та балансу.

Однією з ключових особливостей сценарію є можливість зміни складності гри шляхом регулювання швидкості руху м'яча. На початковому рівні м'яч рухається з невеликою швидкістю, що дозволяє користувачу адаптуватися до механіки керування. На середньому та високому рівнях швидкість поступово зростає, підвищуючи вимоги до реакції, точності рухів і контролю положення тіла.

Згідно з класифікацією ігрових сценаріїв, наведеною в науковій частині роботи, даний сценарій належить до ігор з такими характеристиками:

- положення тіла користувача — стоячи;
- характер рухів — динамічний;
- ціль тренування — координація та баланс;
- рівень складності — змінний (від початкового до просунутого);
- фізичне навантаження — безперервний контроль центру маси тіла.

Описаний ігровий сценарій є основою для подальшої практичної реалізації гри в середовищі Unity та слугує прикладом застосування рекомендацій з дизайну ігрових сценаріїв, розглянутих у першому розділі методичних вказівок.

2.2 Ознайомлення з Unity

Unity є одним із найбільш поширених та універсальних середовищ розробки комп’ютерних ігор та інтерактивних застосунків. Даний ігровий рушій використовується для створення дво- та тривимірних ігор, симулаторів, освітніх програм, а також мобільних застосунків для платформ Android та iOS. Завдяки підтримці фізичних моделей, зручній системі компонентів і кросплатформенності, Unity широко застосовується у проектах, пов’язаних з ігровізацією фізичних вправ і тренуванням моторних навичок користувача.

Особливістю Unity є компонентно-орієнтований підхід до розробки, за якого кожен об’єкт сцени (GameObject) може містити набір компонентів, що визначають його поведінку, фізичні властивості та взаємодію з іншими об’єктами. Такий підхід дозволяє швидко створювати ігрові прототипи та поступово ускладнювати функціональність гри без повної перебудови структури проекту.

При розробці ігор для мобільних платформ Unity надає вбудовані засоби роботи з сенсорами смартфона, зокрема акселерометром та гіроскопом. Це дозволяє реалізовувати керування ігровими об’єктами за допомогою нахилів

мобільного пристрою, що є доцільним для ігрових сценаріїв, спрямованих на тренування балансу та координації рухів. Крім того, Unity забезпечує інструменти для оптимізації продуктивності та створення інсталяційних файлів (APK) для подальшого запуску гри на реальному мобільному пристрой.

Для початку роботи з Unity необхідно скористатися офіційними ресурсами розробника:

- Unity Download — офіційна сторінка для завантаження середовища Unity Hub, за допомогою якого здійснюється встановлення та керування версіями Unity;
- Unity Learn — платформа з навчальними матеріалами, прикладами та інтерактивними курсами, орієнтованими на початківців.

Дані методичні вказівки орієнтовані на користувачів, які раніше не працювали з Unity або мають значну перерву в роботі з цим середовищем. Тому всі подальші етапи розробки гри будуть описані покроково, з поясненням базових понять, налаштувань та дій, необхідних для створення простого ігрового проєкту, формування APK-файлу та запуску гри на смартфоні.

2.3 Встановлення та первинне налаштування Unity

Перед початком розробки ігрового проєкту необхідно встановити середовище Unity та виконати його первинне налаштування. Для цього використовується спеціальний інструмент Unity Hub, який дозволяє керувати версіями рушія, створювати та відкривати проєкти, а також встановлювати додаткові модулі для різних платформ.

2.3.1 Завантаження та встановлення Unity Hub.

Для встановлення Unity необхідно виконати наступні кроки.

Крок 1. Перейти на офіційну сторінку завантаження Unity (Unity Download) (дивіться рисунок 2.1).

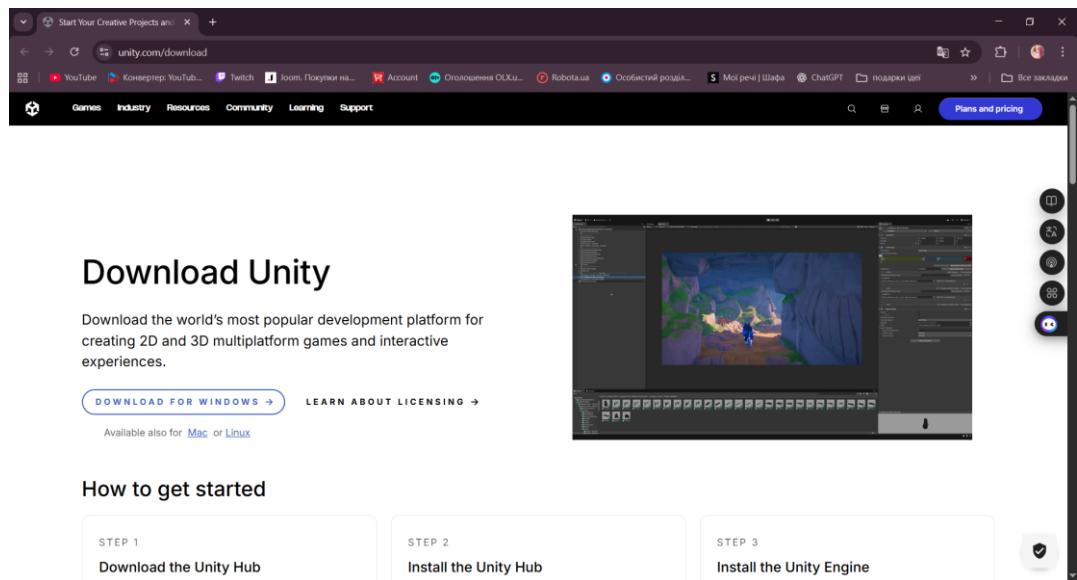


Рисунок 2.1 – Знімок екрану офіційної сторінки завантаження Unity

Крок 2. Натиснути кнопку Download та завантажити інсталяційний файл Unity Hub (дивіться рисунок 2.2).

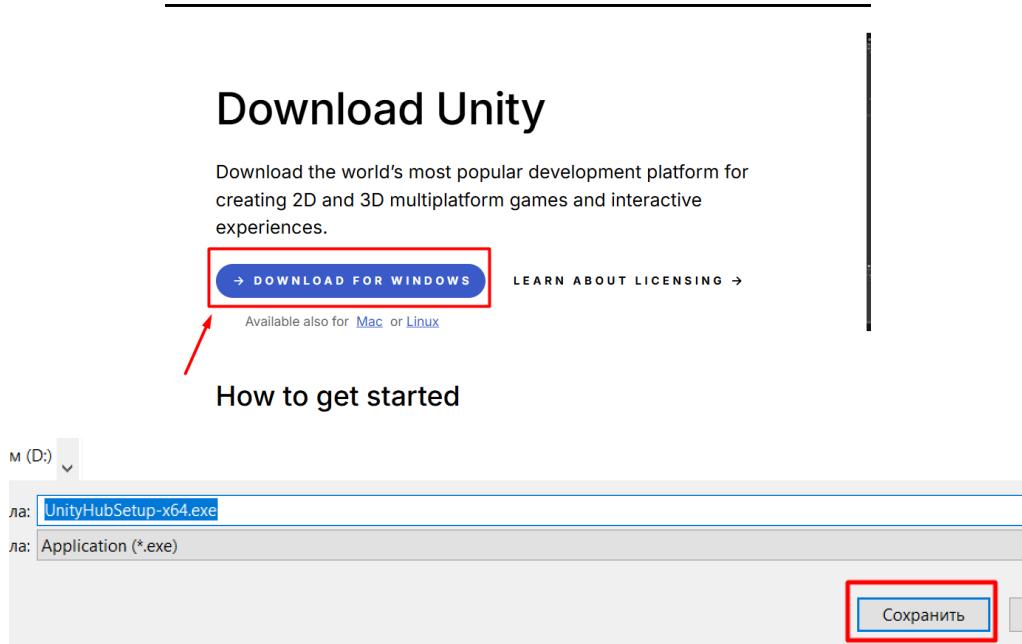


Рисунок 2.2 – Знімок екрану із завантаженням інсталяційного файлу Unity Hub

Крок 3. Запустити інсталятор та завершити встановлення Unity Hub, дотримуючись стандартних інструкцій майстра встановлення (дивіться рисунок 2.3).

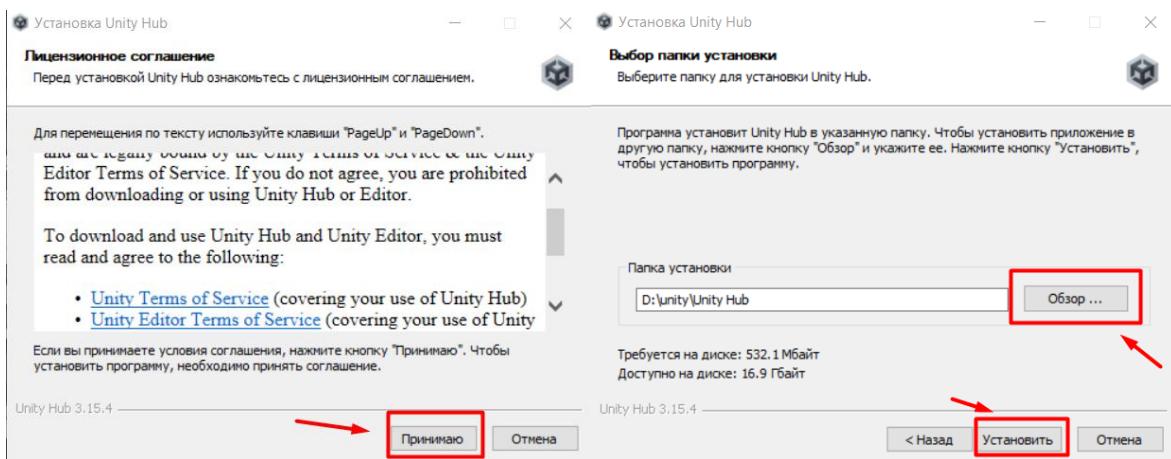


Рисунок 2.3 – Знімок екрану із описом кроку 3

Крок 4. Після завершення інсталяції запустити Unity Hub (дивіться рисунок 2.4).

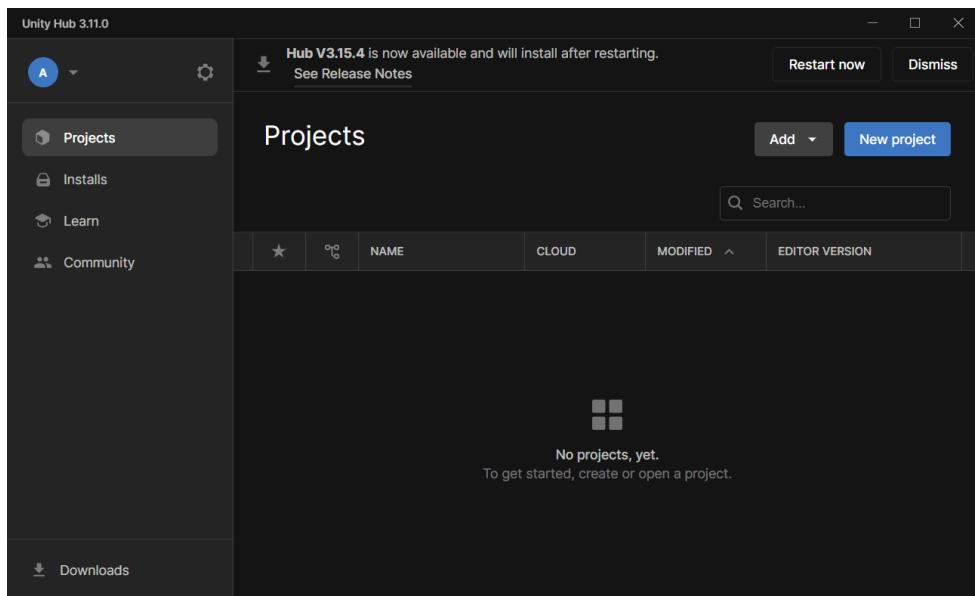


Рисунок 2.4 – Знімок екрану із описом кроку 4

Unity Hub є основною точкою входу до середовища розробки та використовується для подальших етапів роботи.

2.3.2 Вибір та встановлення версії Unity.

Після запуску Unity Hub необхідно встановити версію Unity Editor за наступними кроками.

Крок 1. У вікні Unity Hub перейти до вкладки Installs (дивіться рисунок 2.5).

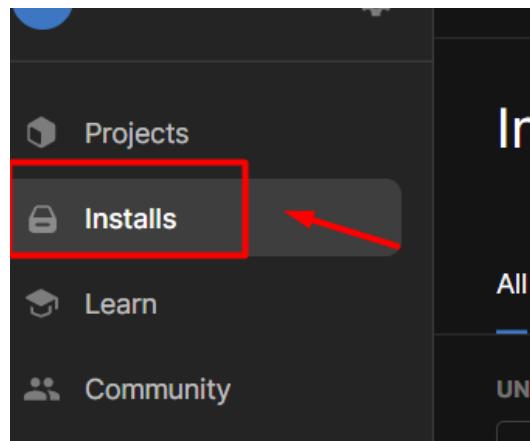


Рисунок 2.4 – Знімок екрану із описом кроку 1

Крок 2. Натиснути кнопку Install Editor (дивіться рисунок 2.5).

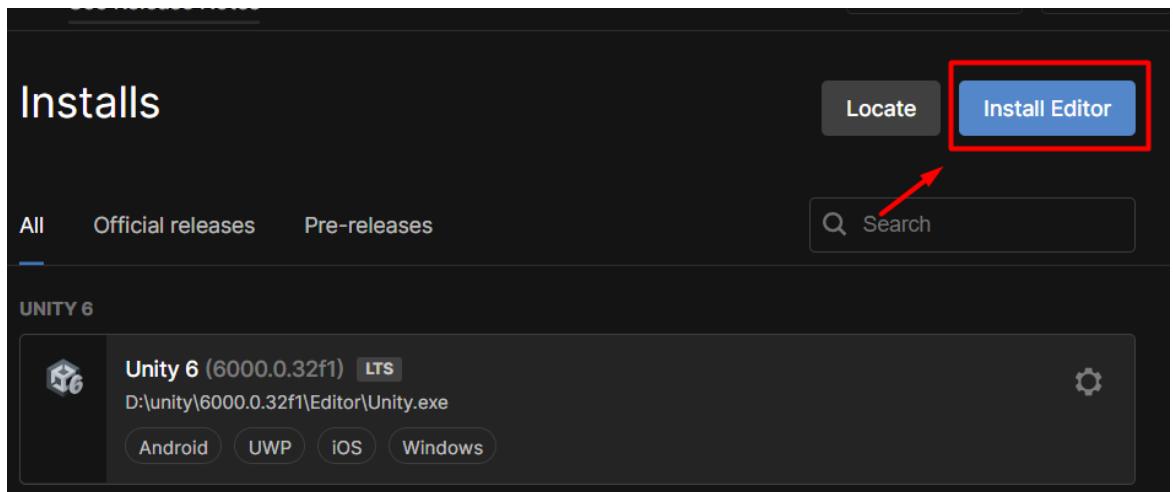


Рисунок 2.5 – Знімок екрану із описом кроку 2

Крок 3. Обрати рекомендовану стабільну версію Unity (LTS — Long Term Support) (дивіться рисунок 2.6).

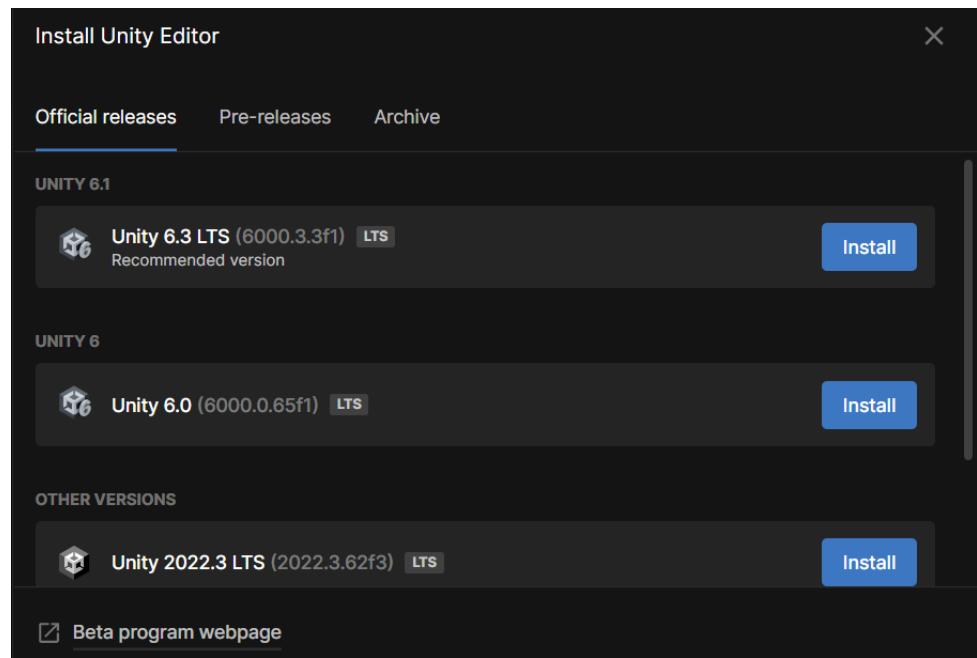


Рисунок 2.6 – Знімок екрану із описом кроку 3

Крок 4. Під час вибору компонентів встановлення обов'язково позначити модулі iOS Build Support та Android Build Support, який включає: Android SDK, Android NDK, OpenJDK (дивіться рисунок 2.7).

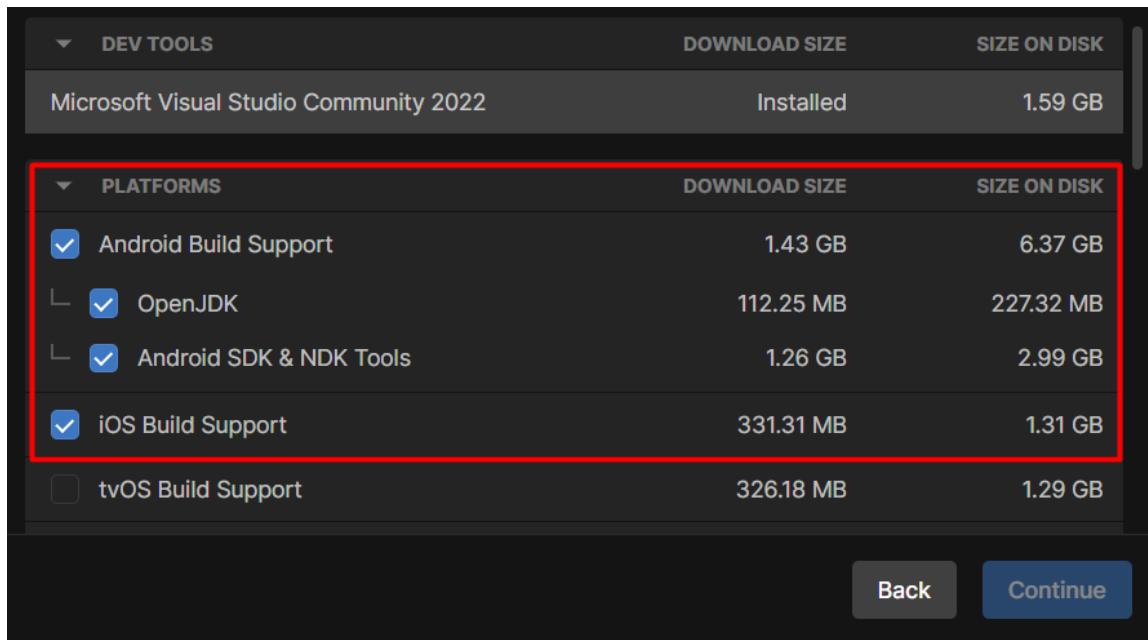


Рисунок 2.7 – Знімок екрану із описом кроку 4

Крок 5. Продовжити встановлення та дочекатися його завершення (дивіться рисунок 2.8).

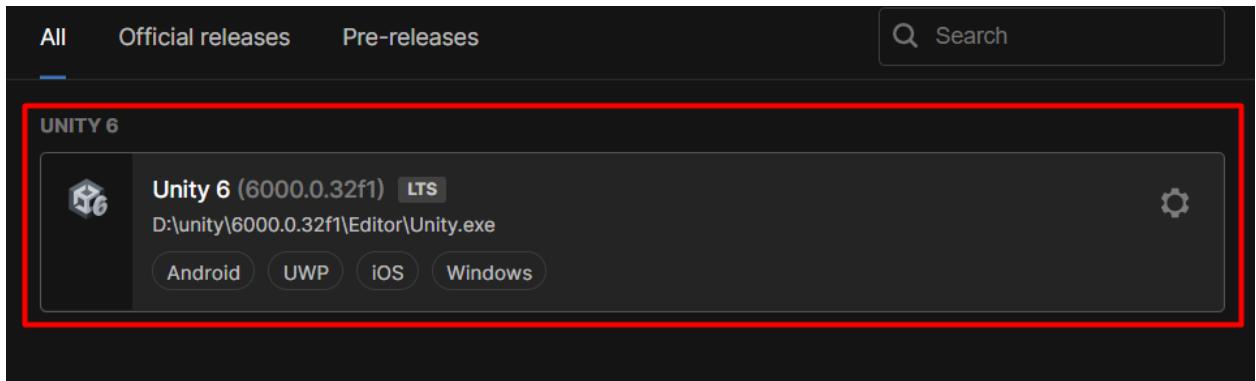


Рисунок 2.8 – Знімок екрану із описом кроку 5

Використання стабільної версії Unity є важливим для забезпечення сумісності проекту між різними користувачами, які виконують завдання за даними методичними вказівками.

2.3.3 Налаштування середовища для мобільної розробки.

Після завершення встановлення Unity Editor необхідно перевірити готовність середовища до створення мобільних застосунків: у Unity Hub переконатися, що встановлена версія Unity має активний модуль Android Build Support (дивіться рисунок 2.9 та 2.10).

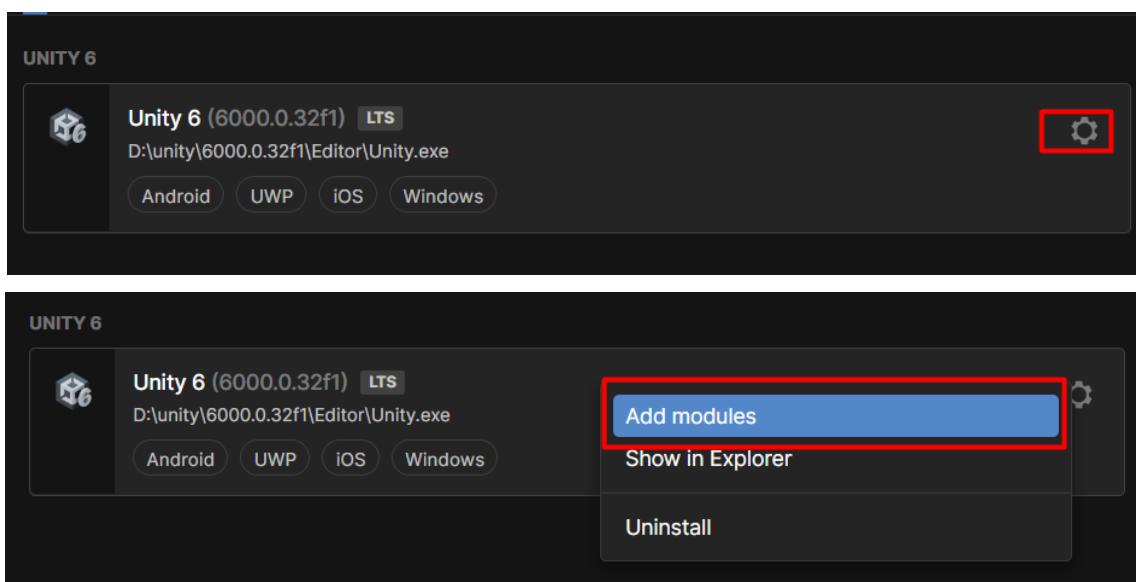


Рисунок 2.9 – Знімок екрану, щоб перевірити готовність середовища до створення мобільних застосунків

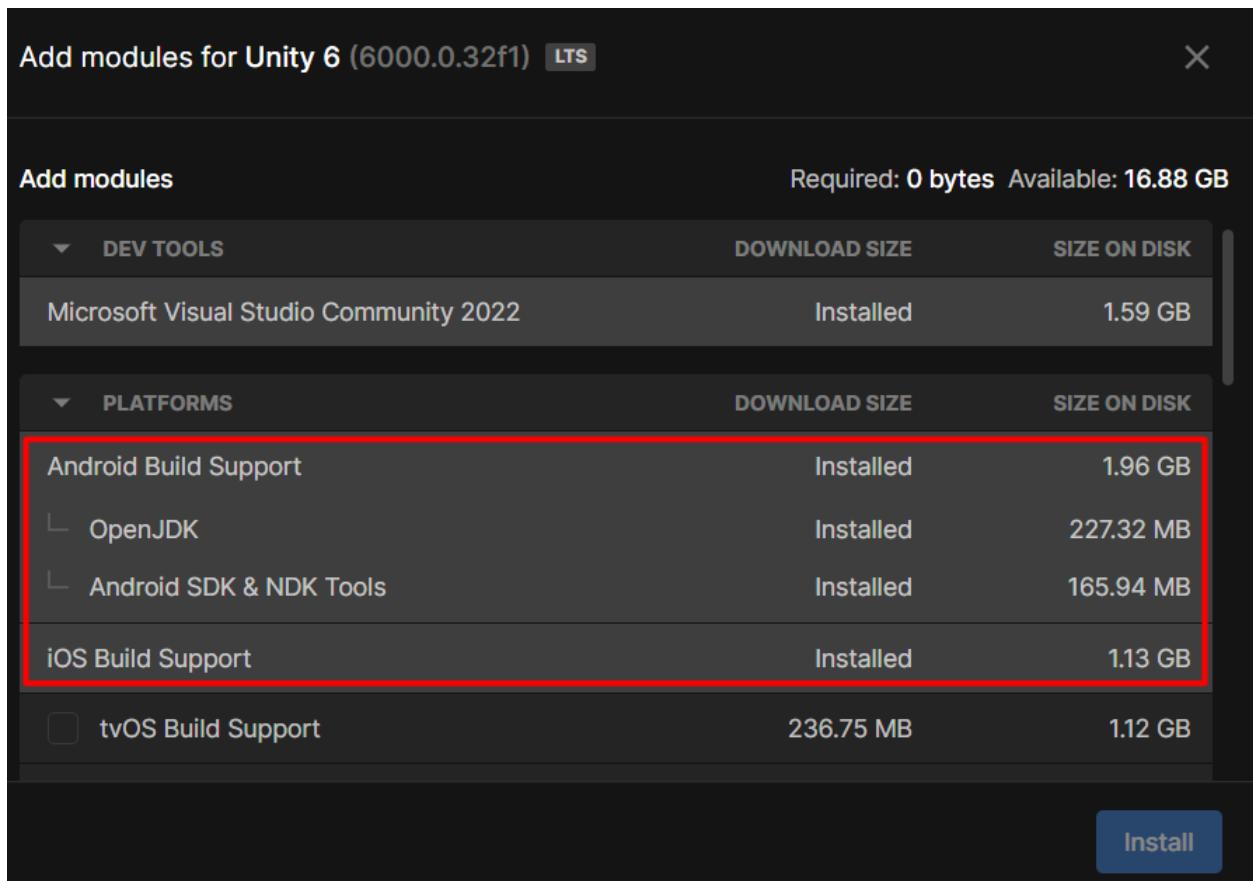


Рисунок 2.10 – Знімок екрану, щоб перевірити готовність середовища до створення мобільних застосунків

У разі відсутності необхідних компонентів додати їх через налаштування встановленої версії редактора(поставити галочку у квадрату зліва на потрібні компоненти та натиснути Install).

Ці налаштування дозволяють у подальшому створювати інсталяційні файли формату APK та запускати ігровий проєкт безпосередньо на смартфоні.

2.3.4 Підготовка до подальшої роботи.

Після успішного встановлення та налаштування Unity середовище вважається готовим до створення нового проєкту. На наступному етапі буде розглянуто процес створення нового ігрового проєкту, вибір типу сцени та ознайомлення з базовими елементами інтерфейсу Unity Editor.

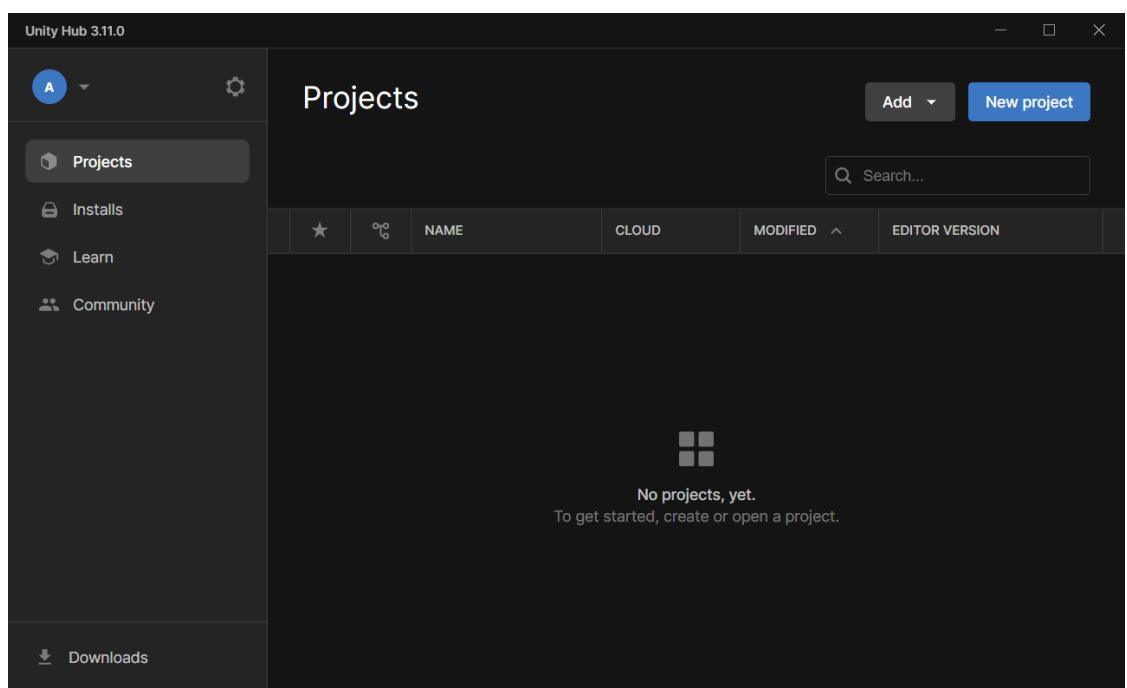
2.4 Створення нового 3D-проекту в Unity

Після встановлення та первинного налаштування середовища Unity можна переходити до створення нового ігрового проекту. Оскільки розроблювана гра передбачає тривимірний ігровий простір (рух героя в об'ємному тунелі), на даному етапі створюється 3D-проект, який забезпечує коректну роботу просторової фізики та камери.

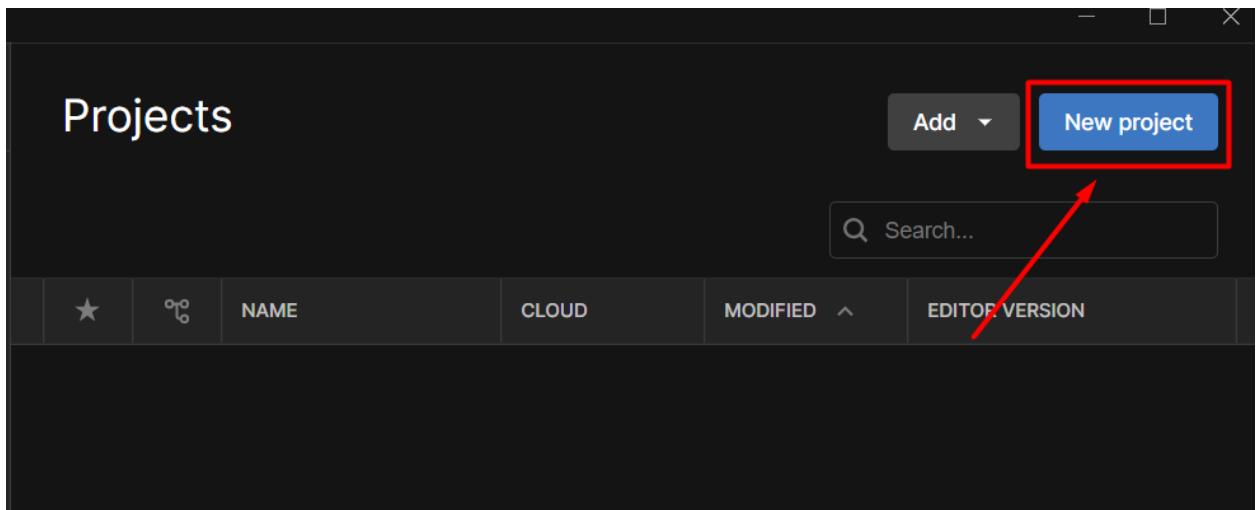
2.4.1 Створення нового проекту.

Для створення нового 3D-проекту необхідно виконати наступні кроки.

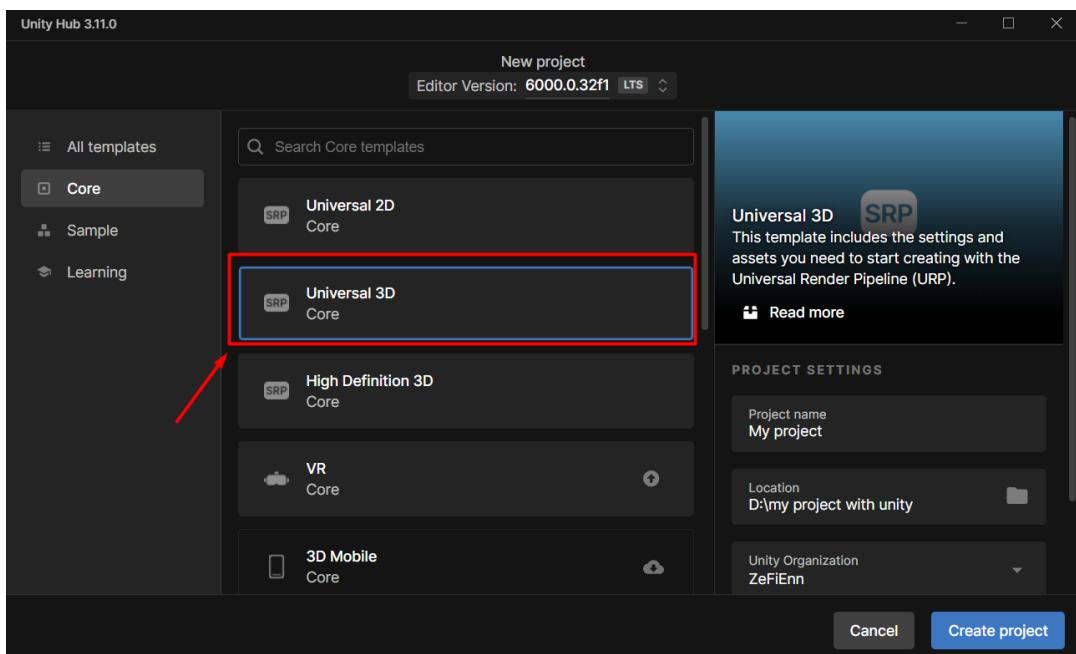
Крок 1. Запустити Unity Hub. Перейти на вкладку Projects.



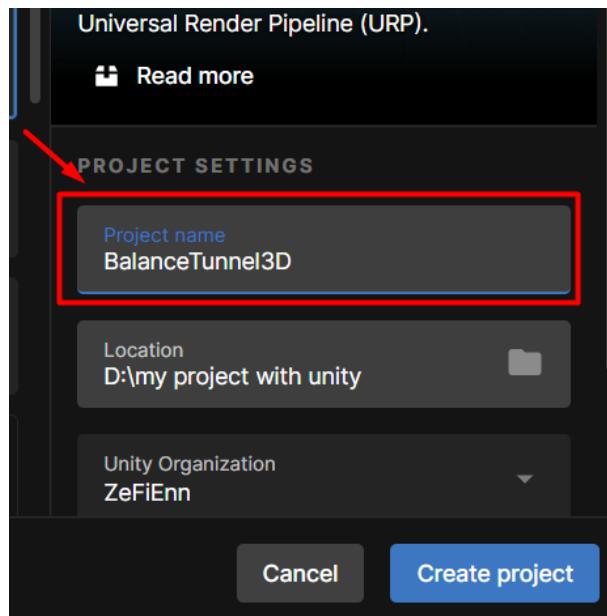
Крок 2. Натиснути кнопку New Project.



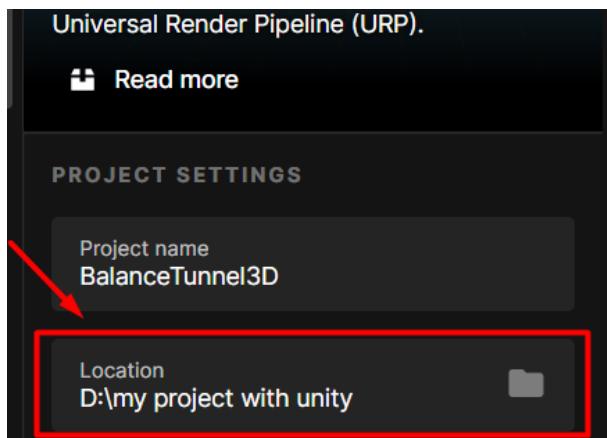
Крок 3. У списку шаблонів обрати Universal 3D.



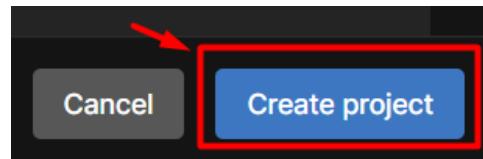
4. Вказати назву проекту (наприклад, BalanceTunnel3D).



5. Обрати директорію для збереження проекту.



6. Натиснути кнопку Create Project та дочекатися відкриття Unity Editor.

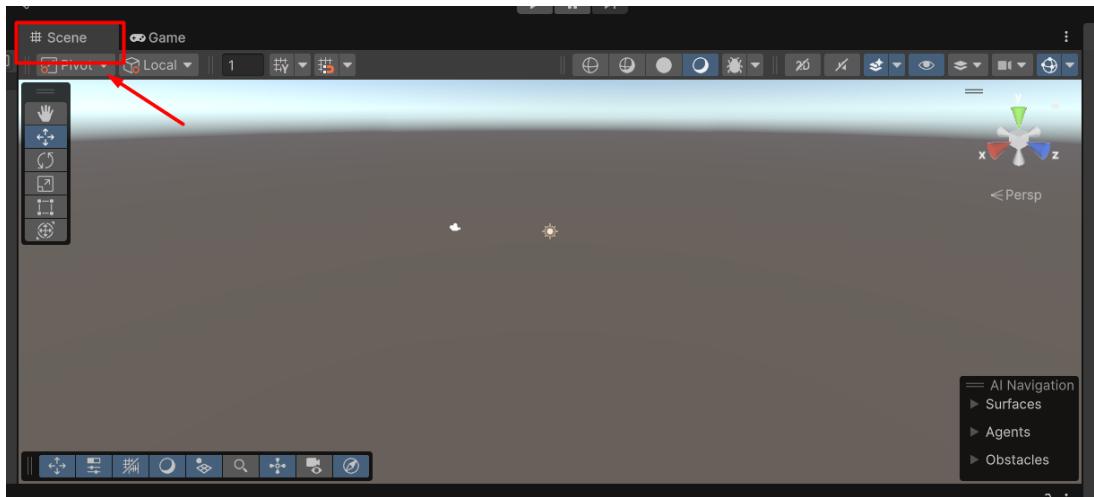


Після створення проекту Unity автоматично відкриває базову 3D-сцену з налаштованим освітленням та камерою.

2.4.2 Ознайомлення з інтерфейсом Unity Editor.

Після відкриття проекту користувач потрапляє в основне вікно Unity Editor, яке складається з таких основних панелей:

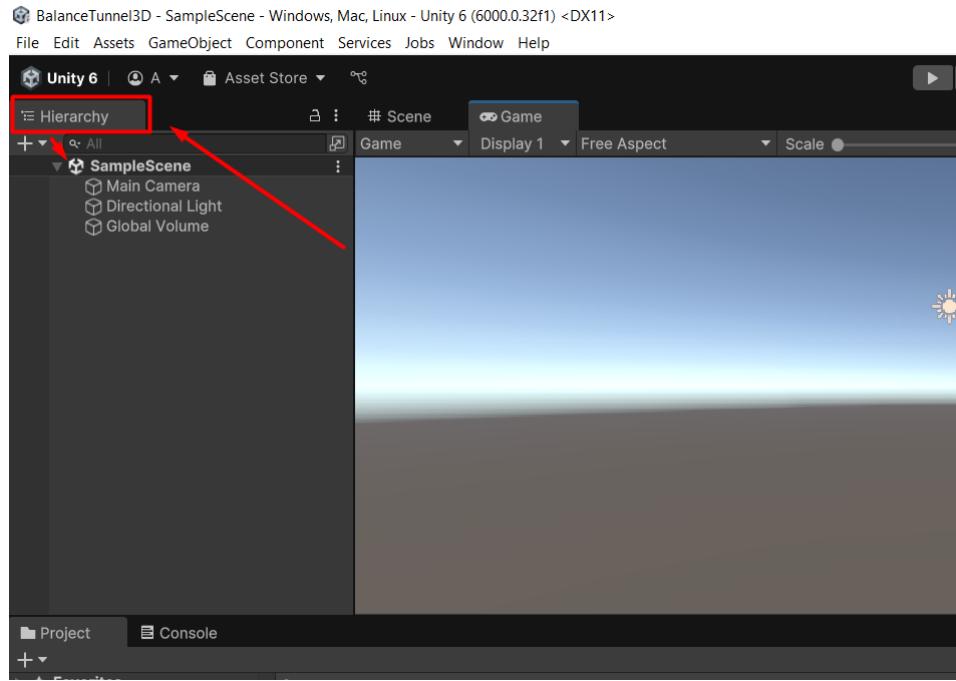
- Scene — робоча область для створення та редагування тривимірної сцени;



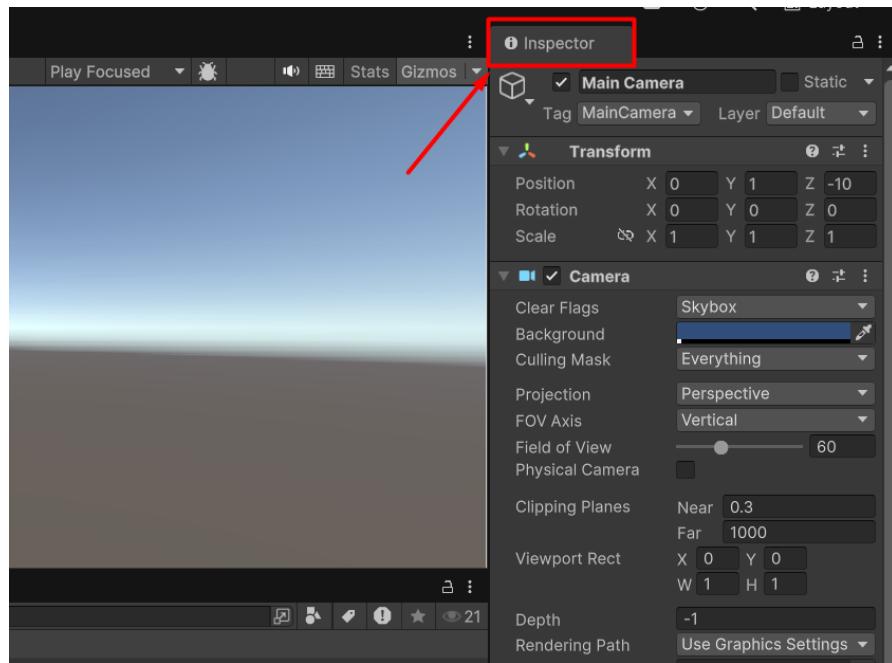
- Game — вікно попереднього перегляду гри з точки зору камери;



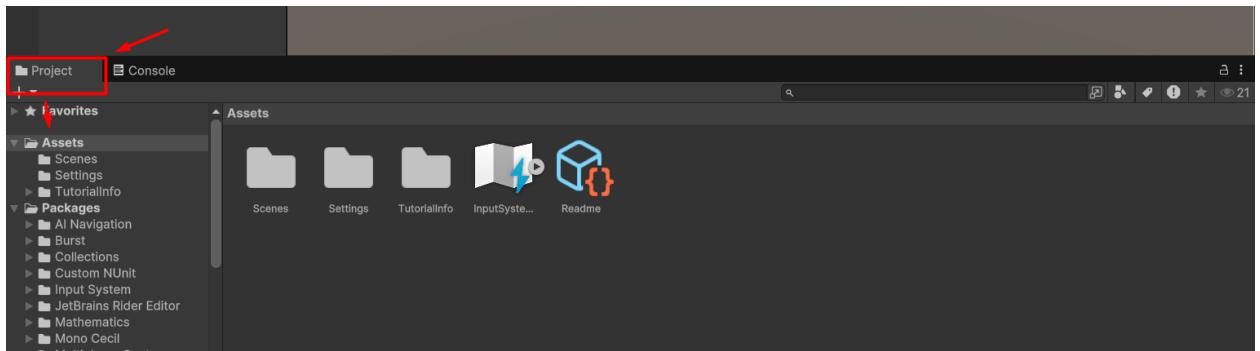
- Hierarchy — список усіх об'єктів сцени;



- Inspector — панель налаштування властивостей вибраного об'єкта;



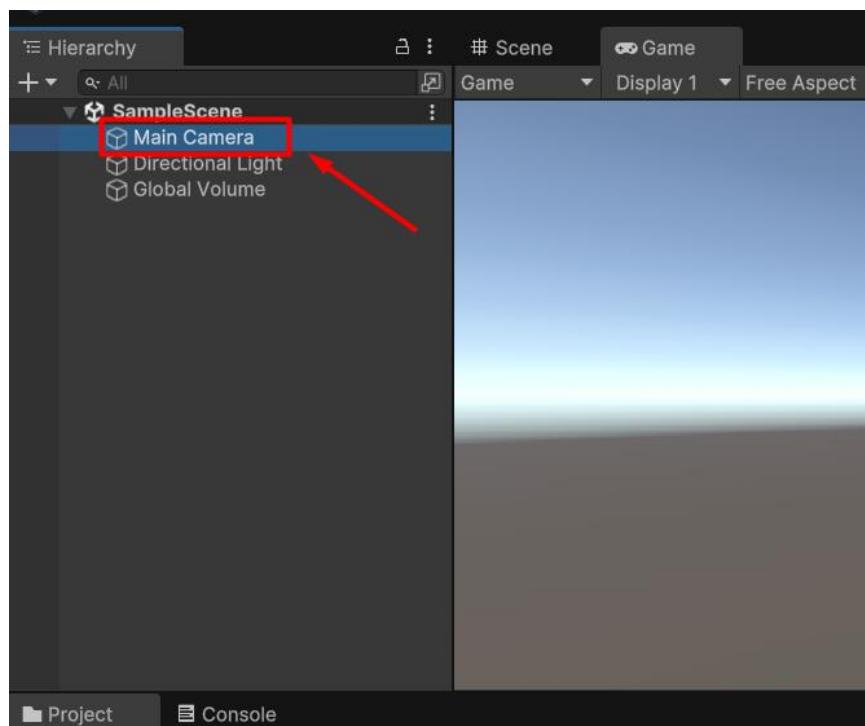
- Project — вікно для роботи з ресурсами проєкту.



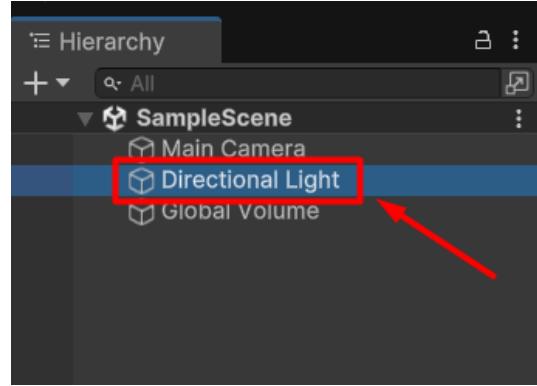
Для 3D-проектів особливу увагу слід приділяти навігації у сцені (обертання, масштабування та переміщення камери огляду).

2.4.3 Базові об'єкти та компоненти 3D-сцени.

У стандартній 3D-сцені за замовчуванням присутні наступні об'єкти. Main Camera — камера, що визначає видиму область сцени.

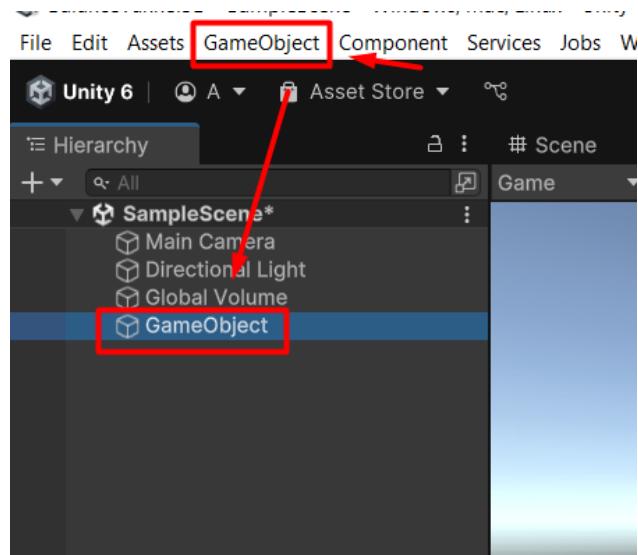


Directional Light — джерело освітлення, необхідне для коректного відображення об'ємних об'єктів.

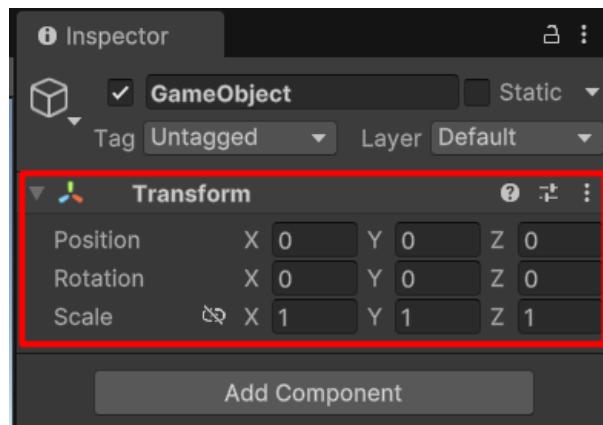


Основними компонентами, які використовуються у 3D-грі, є:

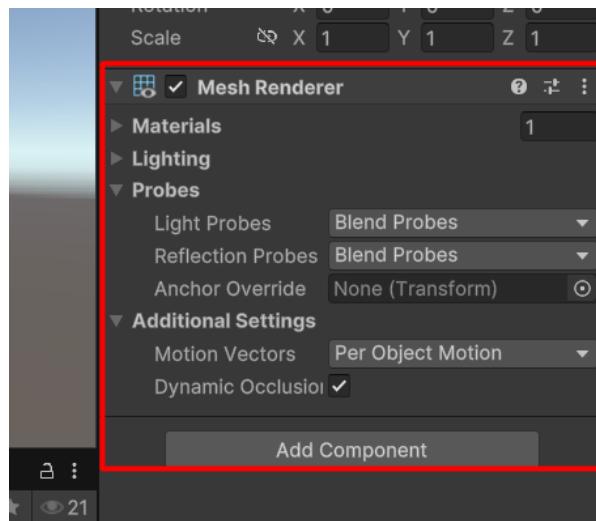
- GameObject — базовий об’єкт сцени;



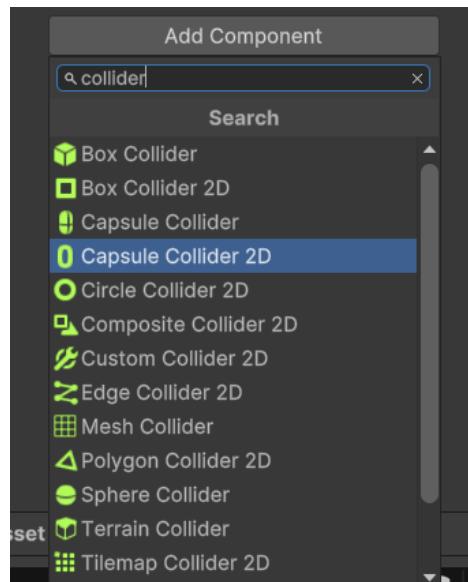
- Transform — компонент, що задає положення, обертання та масштаб об’єкта у просторі;



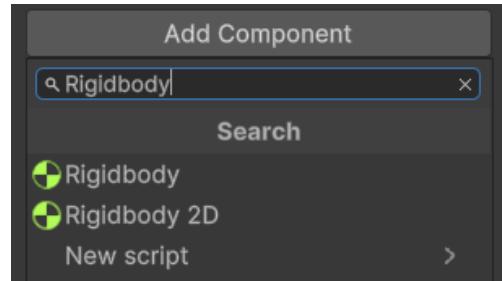
- Mesh Renderer — компонент для відображення тривимірної геометрії;



- Collider — компонент для обробки зіткнень між об’єктами;



- Rigidbody — компонент, що реалізує фізичну модель руху об’єкта.



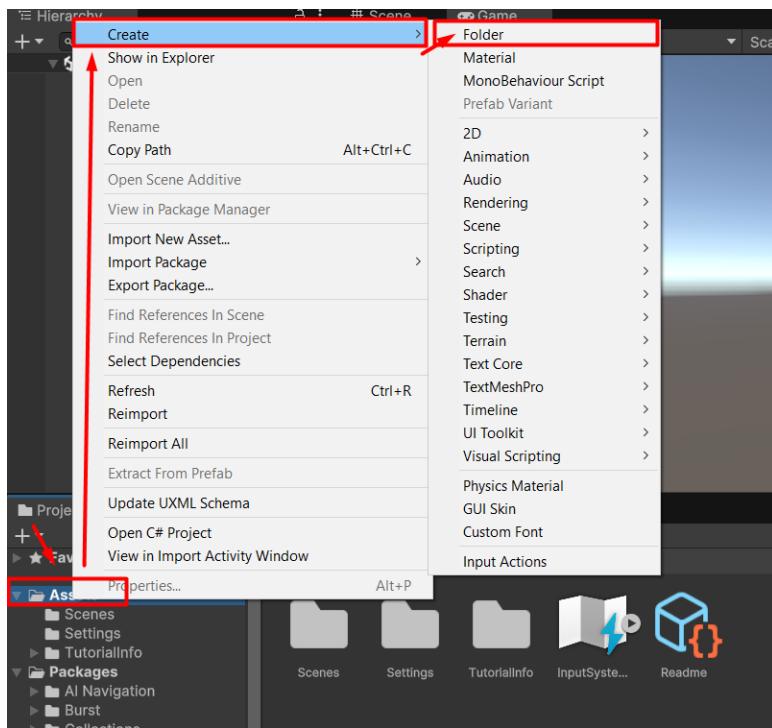
Саме ці компоненти будуть використані при створенні персонажу, тунелю та його стінок.

2.4.4 Підготовка структури проєкту.

Для зручності подальшої розробки рекомендується впорядкувати ресурси проєкту, створивши у вікні Project такі папки:

- Scenes — сцени гри;
- Scripts — програмні скрипти керування логікою гри;
- Materials — матеріали для 3D-об'єктів;
- Models — тривимірні моделі (тунель, додаткові об'єкти);
- Prefabs — збережені налаштовані об'єкти.

Для цього натискаємо правою кнопкою в вікні Project, створюємо 5 пустих папок, та називаємо відповідними іменами:

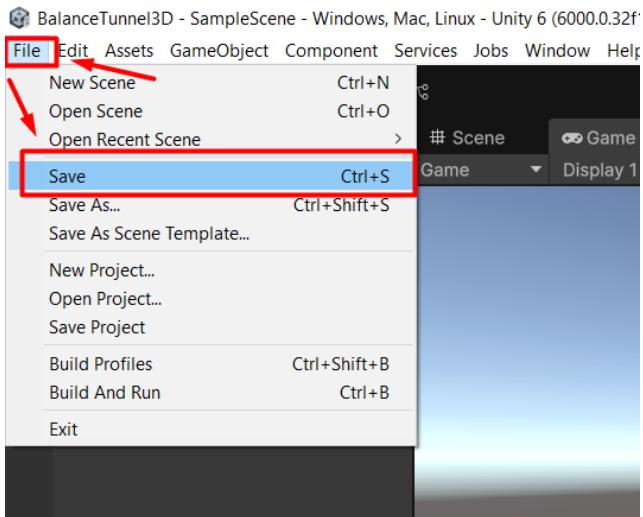


Чітка структура проєкту спрощує навігацію та подальше розширення функціональності гри.

2.4.5 Збереження сцени.

Після створення та первинного налаштування сцени необхідно зберегти її:

У меню File обрати пункт Save.



Збереження сцени є обов'язковим перед переходом до реалізації ігрового сценарію.

Таким чином, на даному етапі створено 3D-проект у Unity, виконано ознайомлення з інтерфейсом редактора та підготовлено базову структуру для подальшої розробки гри з використанням фізики та просторового керування.

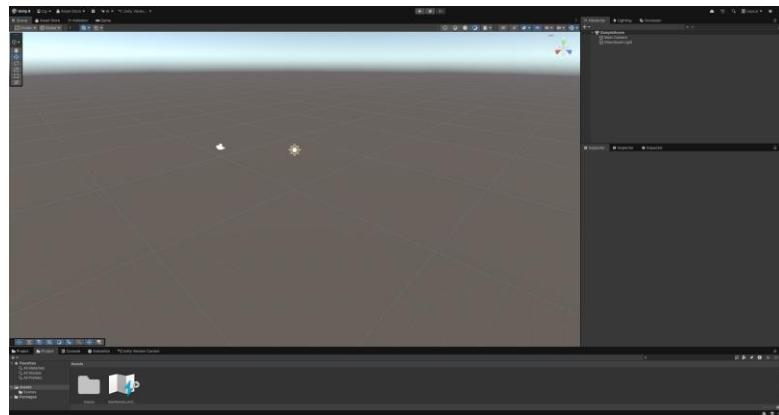
2.5 Підготовка середовища розробки та створення проєкту

На першому етапі виконується підготовка програмного середовища, необхідного для розробки тривимірної мобільної гри з використанням ігрового рушія Unity. Для цього на комп'ютер встановлюється Unity Hub, який використовується для керування версіями редактора Unity та створення нових проєктів. У Unity Hub обирається та встановлюється версія Unity Editor з довгостроковою підтримкою (LTS), що забезпечує стабільну роботу середовища розробки. Під час встановлення редактора обов'язково активується підтримка платформи Android разом із необхідними компонентами, такими як Android SDK, NDK та OpenJDK, які забезпечують можливість створення та збірки мобільних застосунків.

Після завершення встановлення середовища розробки створюється новий проєкт. Для цього в Unity Hub ініціюється створення проєкту з використанням шаблону тривимірної сцени. Проєкту надається назва, після чого обирається каталог для його збереження на комп'ютері. Після

підтвердження створення проекту Unity автоматично відкриває головне вікно редактора з початковою сценою.

Початкова сцена містить стандартний набір об'єктів, необхідних для базового відображення тривимірного простору, зокрема основну камеру та джерело спрямованого світла.



На даному етапі ці об'єкти не змінюються та не видаляються, оскільки вони використовуватимуться під час подальшого налаштування сцени. Таким чином створюється початковий каркас проекту, який є основою для реалізації ігрової логіки, візуальної частини та керування.

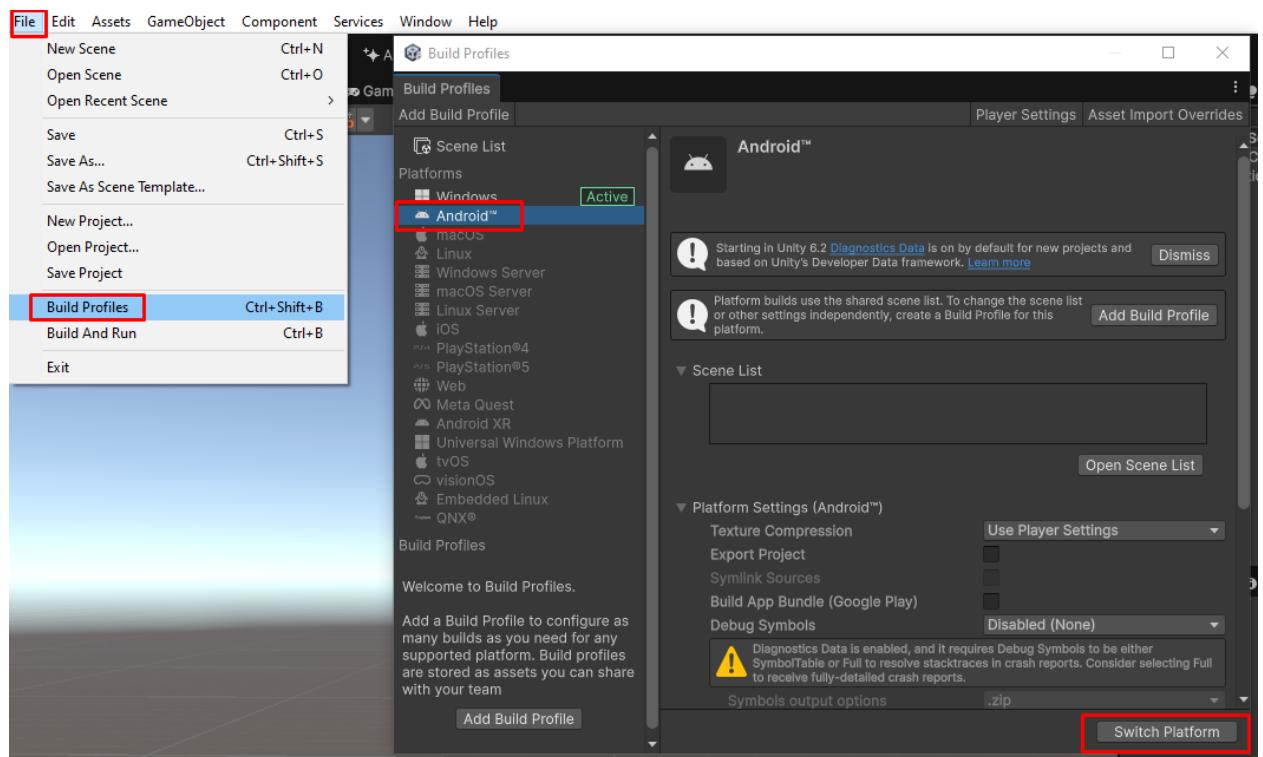
У результаті виконання першого етапу підготовлено середовище розробки, створено новий тривимірний проект у Unity та відкрито базову сцену, готову до подальшої поетапної реалізації гри.

2.6 Початкове налаштування сцени, камери та параметрів відображення для Android

Після створення нового проекту в Unity автоматично відкривається початкова сцена, яка містить стандартні об'єкти Main Camera та Directional Light. На цьому етапі виконується базове налаштування сцени з урахуванням того, що гра розробляється для мобільних пристройів під керуванням операційної системи Android.

Перед налаштуванням параметрів відображення, архітектури збірки та орієнтації екрана необхідно обов'язково встановити цільову платформу проекту. За замовчуванням після створення нового проекту Unity працює в режимі збірки для настільних операційних систем Windows, macOS або Linux. У цьому режимі налаштування Android у вікні Player недоступні, а всі параметри відображаються лише для настільної платформи.

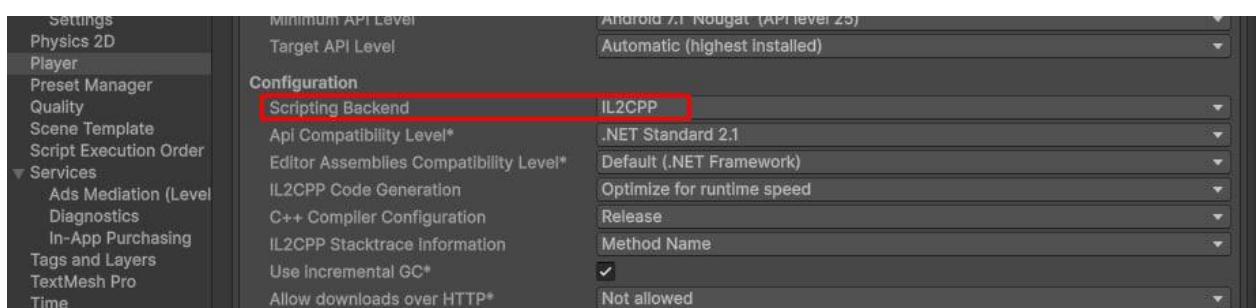
Для перемикання проекту на платформу Android у верхньому меню редактора Unity необхідно відкрити пункт File та перейти до вікна Build Profiles. У лівій частині цього вікна відображається список доступних платформ збірки. У списку потрібно вибрати платформу Android, після чого натиснути кнопку Switch Platform.



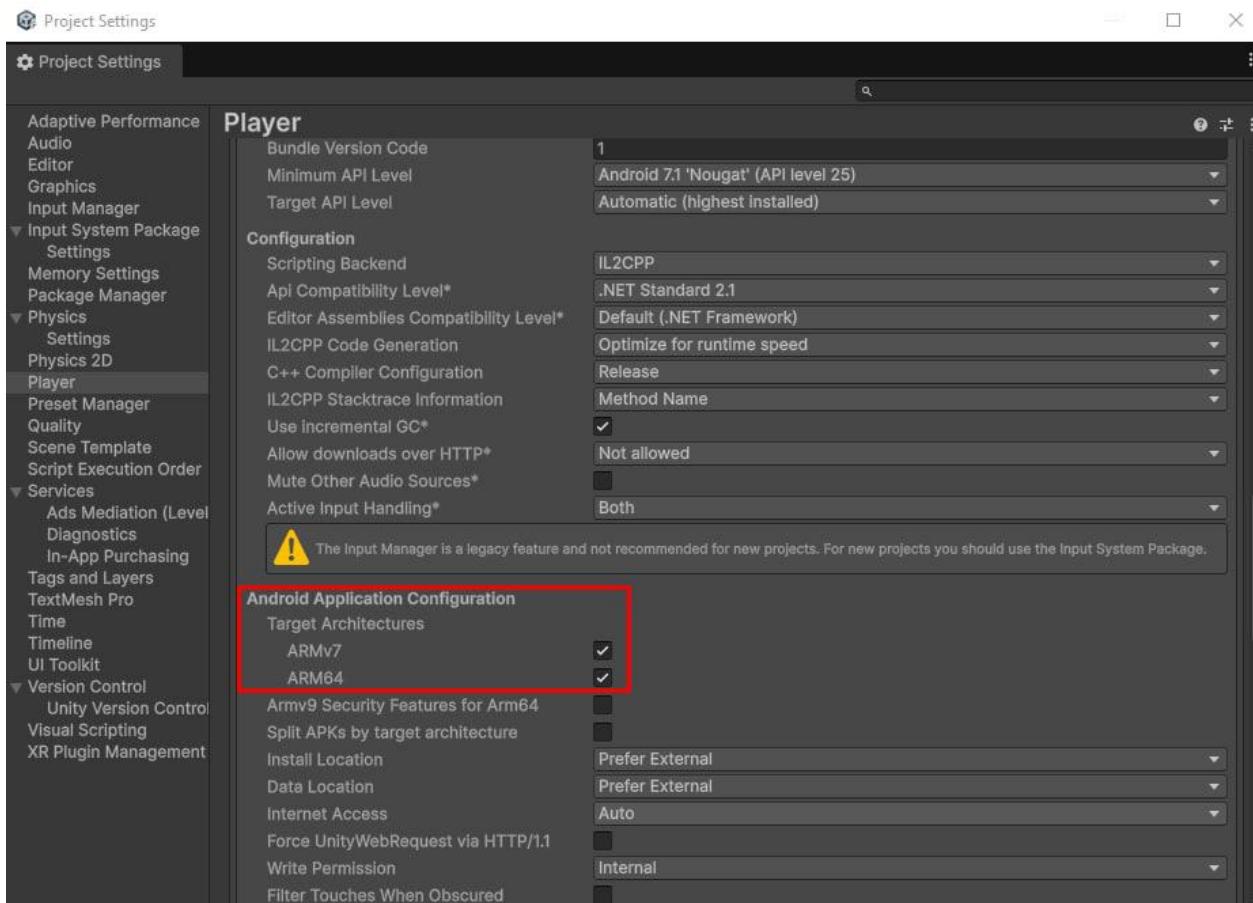
Процес перемикання може тривати деякий час, оскільки Unity виконує переконфігурацію проекту під мобільну платформу. Після завершення перемикання платформи налаштування проекту автоматично адаптуються для Android. Якщо платформа Android відсутня, це означає, що під час встановлення Unity Editor не були додані необхідні модулі для Android-

розробки. У такому випадку необхідно відкрити Unity Hub, перейти до встановленої версії Unity та додати модуль Android Build Support разом із Android SDK, NDK та OpenJDK.

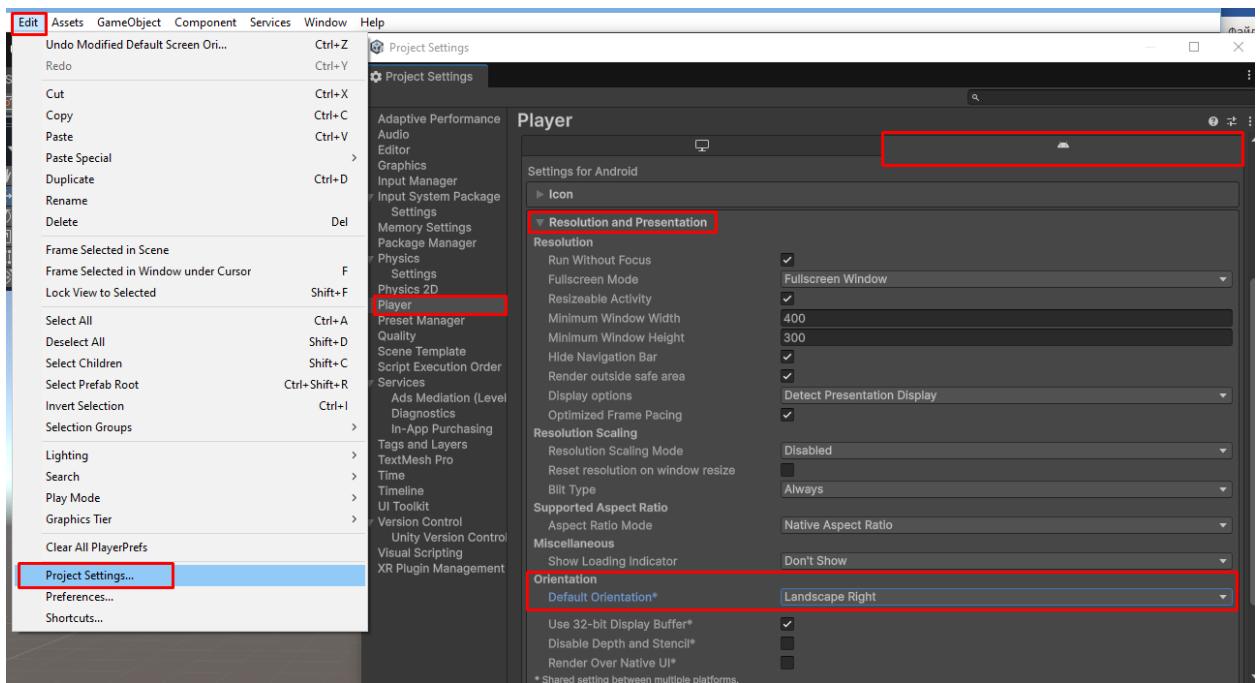
Наступним кроком виконується налаштування параметрів збірки та виконуваного середовища Android. Для цього у верхньому меню редактора Unity відкривається пункт Edit, після чого обирається вікно Project Settings. У лівій частині вікна налаштувань переходят до розділу Player, де у верхній частині активується вкладка Android. У секції Other Settings параметр Scripting Backend встановлюється у значення IL2CPP. Використання IL2CPP забезпечує попередню компіляцію коду у машинні інструкції цільової платформи, що підвищує продуктивність гри, стабільність виконання та відповідає вимогам сучасних версій Android.



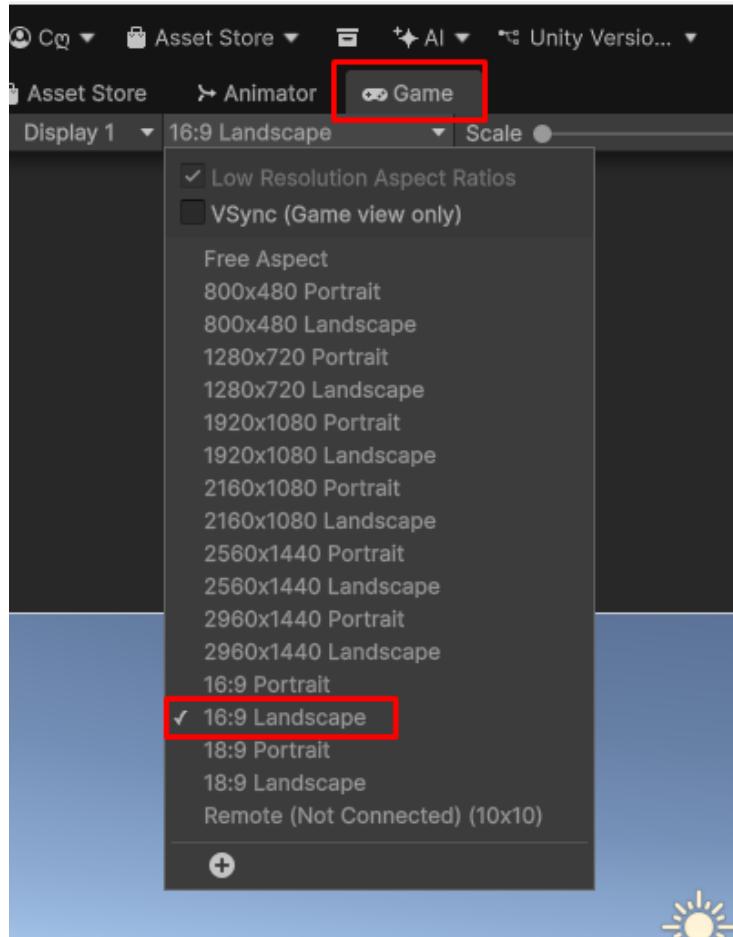
У тій самій секції налаштовуються підтримувані архітектури процесорів. У параметрі Target Architectures необхідно увімкнути ARMv7 та ARM64. Архітектура ARM64 є обов'язковою для публікації додатків у Google Play, тоді як підтримка ARMv7 дозволяє забезпечити сумісність із ширшим колом старіших мобільних пристройів. Інші параметри цієї секції залишаються у стандартному стані.



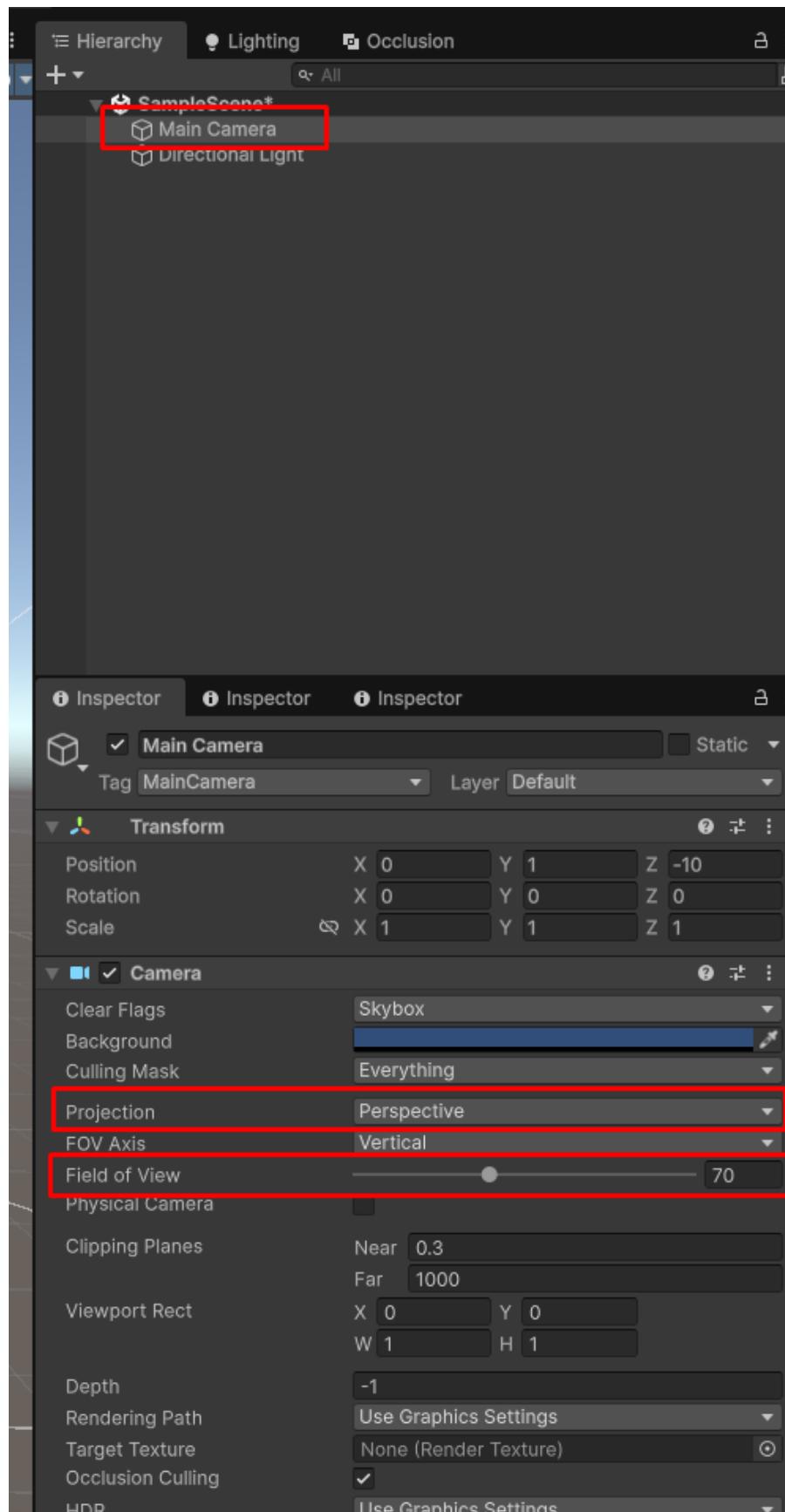
Після цього виконується налаштування параметрів відображення гри для цільової платформи. У розділі Resolution and Presentation параметр Default Orientation встановлюється у значення Landscape Left або Landscape Right, що забезпечує горизонтальну орієнтацію гри та оптимальний кут огляду для руху персонажа вздовж тунелю. Інші параметри цього розділу на початковому етапі залишаються без змін.



Далі необхідно перевірити коректність відображення сцени у вікні попереднього перегляду. Для цього відкривається вкладка Game, розташована поруч із вкладкою Scene. У верхній частині вікна Game обирається режим відображення з горизонтальною орієнтацією екрана (Landscape) та співвідношенням сторін 16:9. Це дозволяє заздалегідь оцінити, як ігрова сцена буде виглядати на екрані смартфона під час запуску гри.



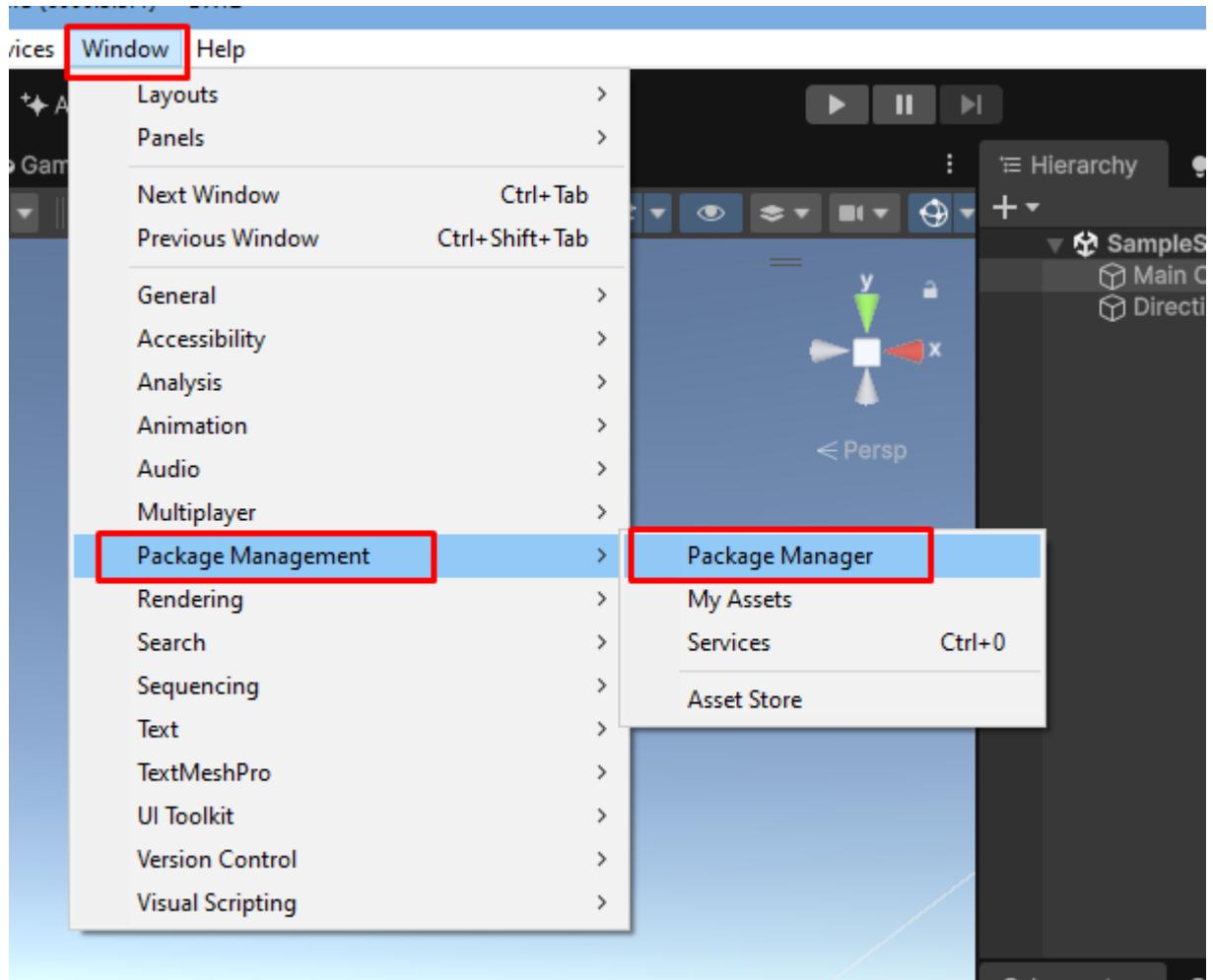
На завершення етапу виконується початкове налаштування основної камери сцени. У вікні Hierarchy обирається об'єкт Main Camera, після чого у вікні Inspector змінюються її параметри. Тип проєкції камери встановлюється у режим Perspective, що забезпечує коректне відображення тривимірного простору. Значення параметра Field of View задається в межах від 60 до 70 градусів, що дозволяє отримати достатньо широкий огляд тунелю без надмірних спотворень перспективи. Позиція та обертання камери на цьому етапі можуть залишатися стандартними, оскільки остаточне розташування буде визначене після додавання ігрового персонажа та реалізації механізму слідування камери.



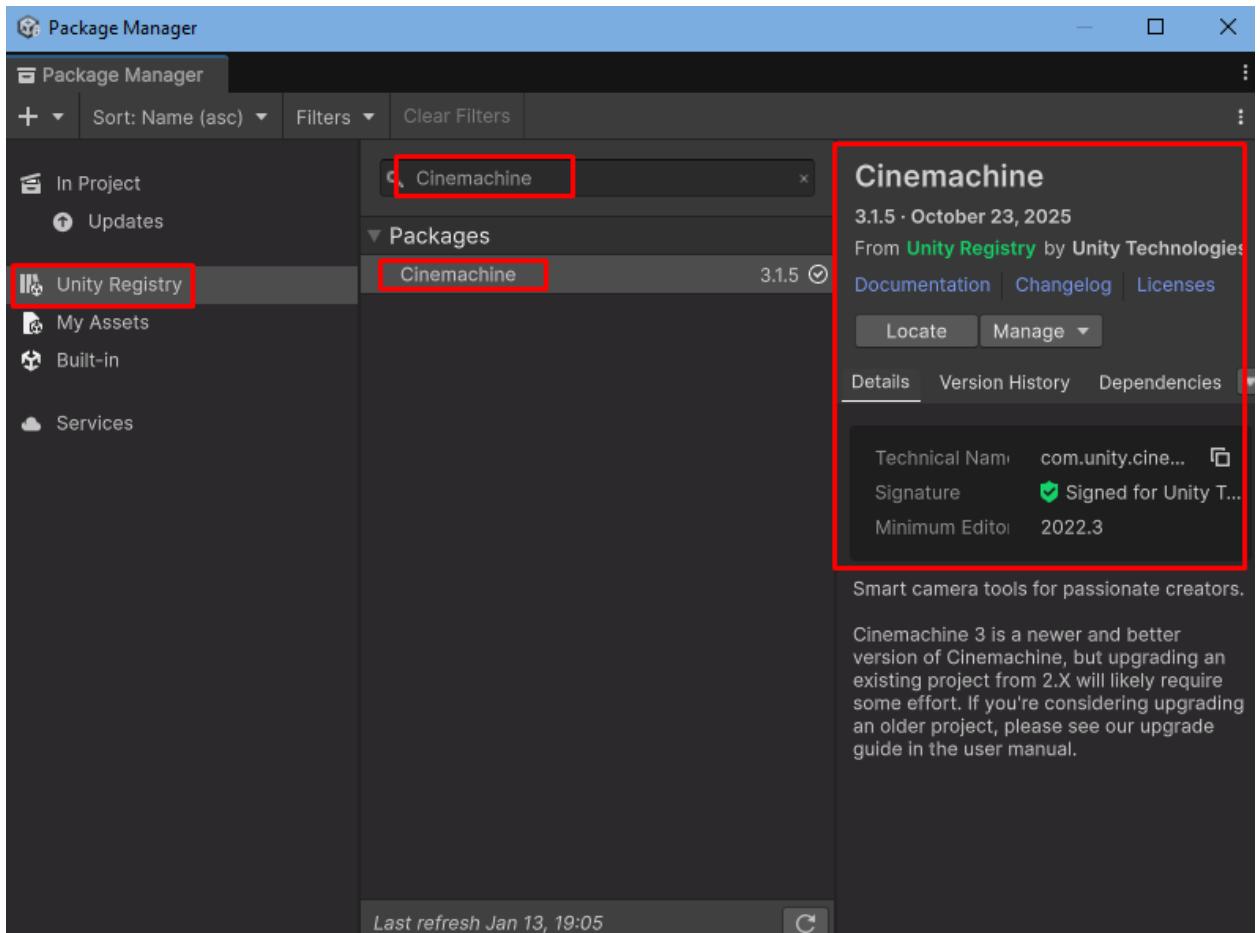
Після базового налаштування основної камери доцільно використати систему Cinemachine, яка є офіційним інструментом Unity для керування поведінкою камери. Cinemachine дозволяє автоматично реалізувати

слідування камери за ігровим об'єктом, згладжувати рух та підтримувати стабільний кут огляду без необхідності створення власних скриптів.

Для підключення Cinemachine у редакторі Unity у верхньому меню відкривається пункт Window, після чого обирається Package Manager.



У списку пакетів необхідно знайти пакет Cinemachine та встановити його, якщо він ще не доданий до проєкту.

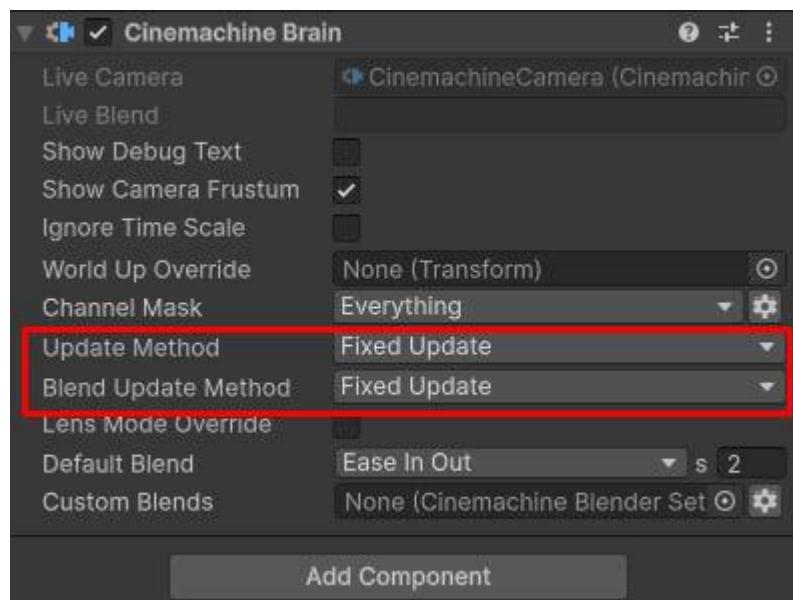


Після завершення встановлення в ієрархії сцени обирається об'єкт Main Camera, до якого автоматично додається компонент Cinemachine Brain. Цей компонент відповідає за керування всіма віртуальними камерами сцени.

Далі створюється віртуальна камера Cinemachine. Для цього у верхньому меню обирається пункт GameObject, після чого відкривається розділ Cinemachine та створюється об'єкт Cinemachine Camera. У вікні Inspector для створеної віртуальної камери в полі в подальшому буде вказано ігровий об'єкт м'яча, а параметри позиції та обертання дозволять розмістити камеру позаду та трохи вище персонажа. На цьому етапі точне налаштування слідування не виконується, оскільки ігровий об'єкт ще не додано до сцени.

При налаштуванні Main Camera рекомендується виставити параметр Update Method у значення Fixed Update, що дозволяє камері оновлюватись у синхронізації з фізичним оновленням об'єктів на сцені, забезпечуючи стабільність і плавність рухів. Аналогічно, для параметра Blend Update Method

також встановлюється Fixed Update, що забезпечує коректне згладжування переходів між різними віртуальними камерами без ривків та смикань під час зміни кадру.



Використання Cinemachine на даному етапі забезпечує готову основу для коректної роботи камери під час руху м'яча тунелем та значно спрощує подальшу реалізацію ігрового процесу.

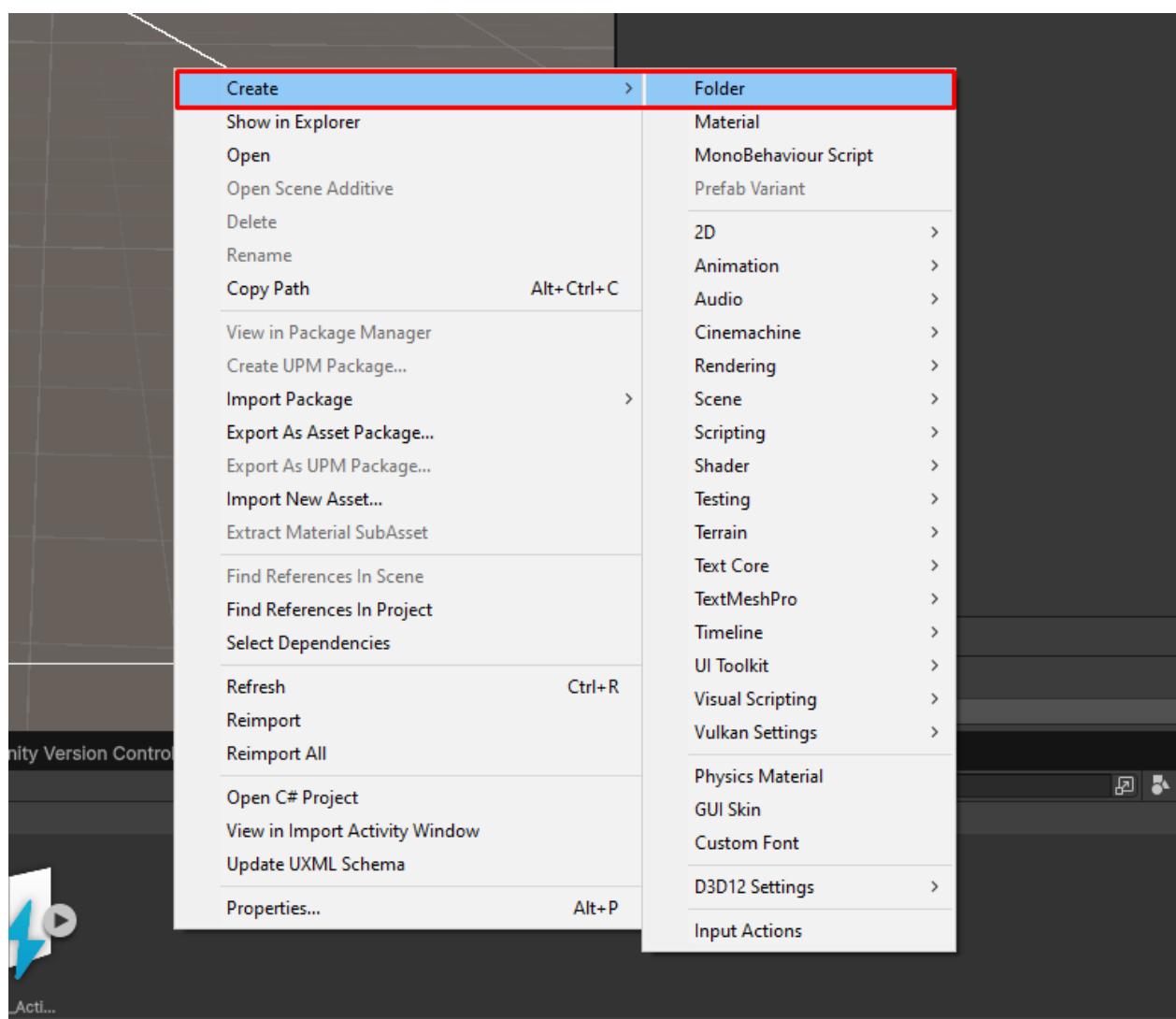
Окрему увагу слід приділити освітленню сцени. Об'єкт Directional Light забезпечує базове освітлення та дозволяє чітко бачити геометрію об'єктів під час подальшої розробки. На цьому етапі його інтенсивність та напрямок можуть залишатися без змін, оскільки художнє налаштування освітлення виконується після завершення основної ігрової логіки.

У результаті виконання другого етапу сцена підготовлена для подальшого наповнення ігровими об'єктами, параметри відображення налаштовані відповідно до вимог платформи Android, а камера готова до використання у мобільній грі з перспективним оглядом.

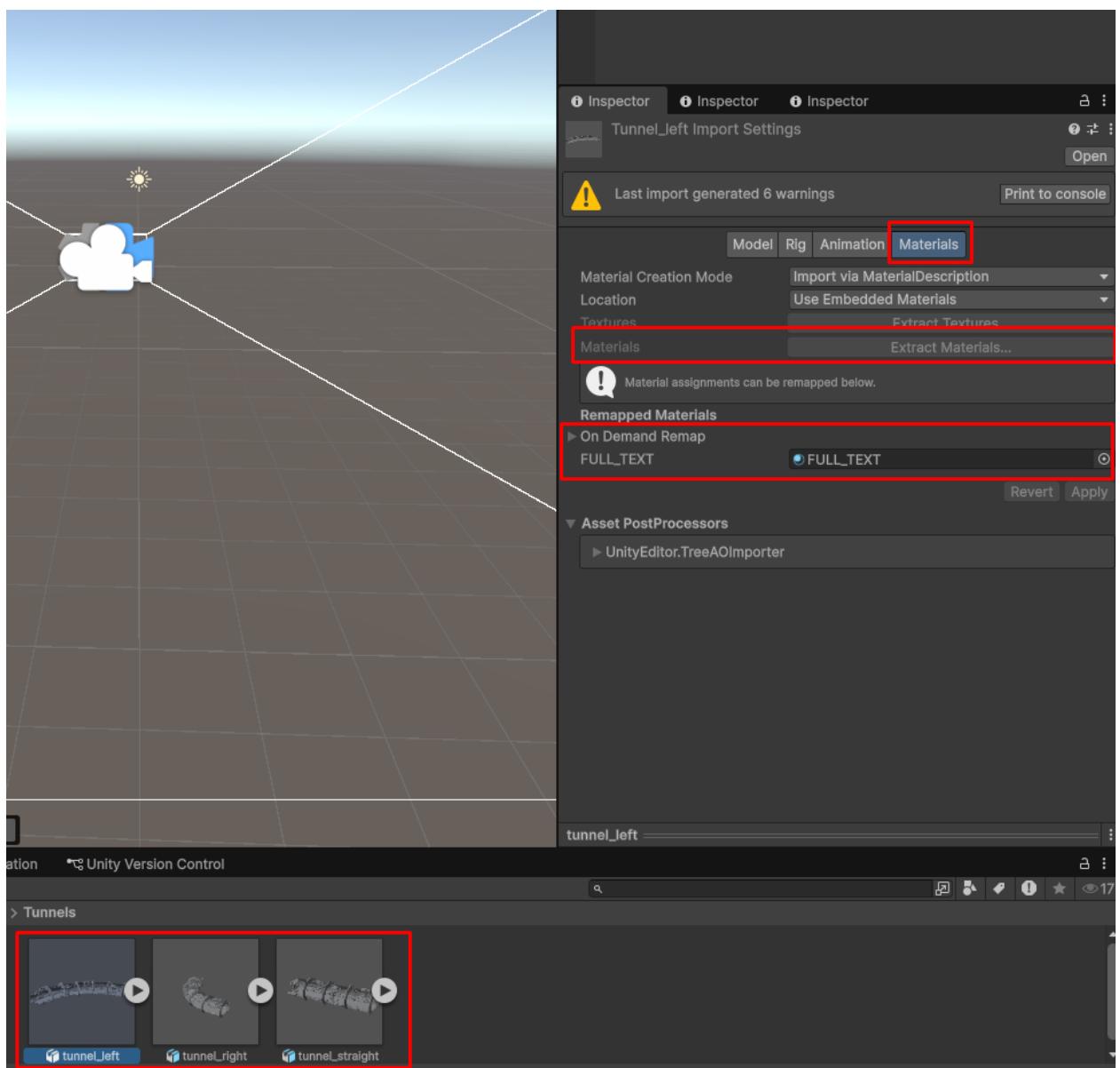
2.7 Організація структури проєкту, імпорт моделей та налаштування матеріалів

На цьому етапі виконується підготовка ресурсів гри, впорядкування структури проєкту та налаштування візуального вигляду ігрових об'єктів. Коректна організація файлів і правильне підключення матеріалів забезпечують стабільну роботу проєкту та спрощують подальшу розробку.

У вікні Project у кореневій папці Assets створюється структура папок для збереження основних типів ресурсів. Окремі папки використовуються для тривимірних моделей, матеріалів, текстур і скриптів. Після цього у папку моделей імпортуються файли тунелів і модель ігрового персонажа шляхом перетягування з файлової системи або за допомогою стандартної команди імпорту Unity.



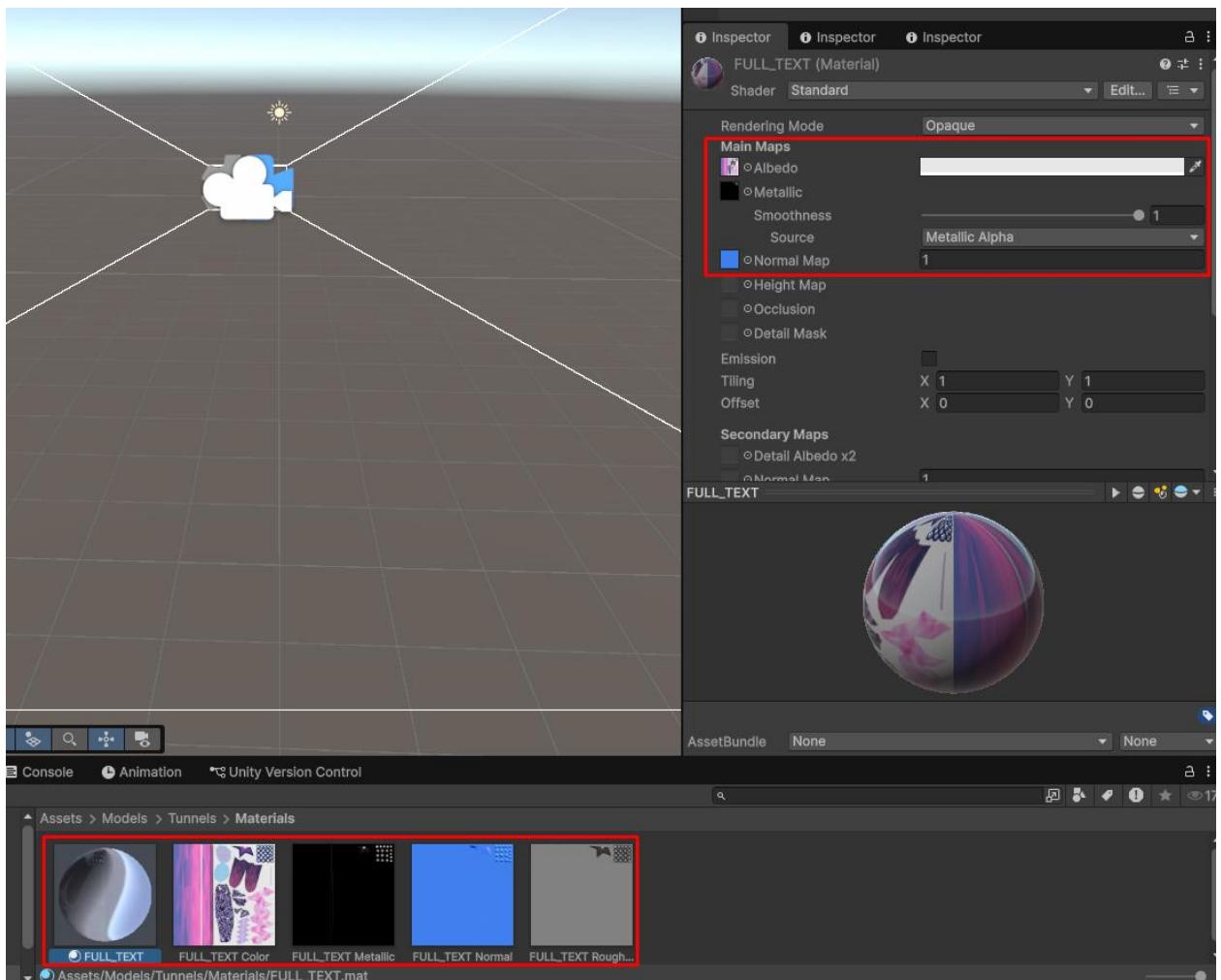
Після додавання моделей необхідно виконати витяг матеріалів, якщо вони були вбудовані в файл моделі. Для цього у вікні Project обирається файл моделі, після чого у вікні Inspector відкривається вкладка Materials. У цій вкладці використовується команда Extract Materials, після чого вказується папка, у яку Unity збереже створені матеріали. У результаті для кожної моделі створюються окремі файли матеріалів, доступні для редагування.



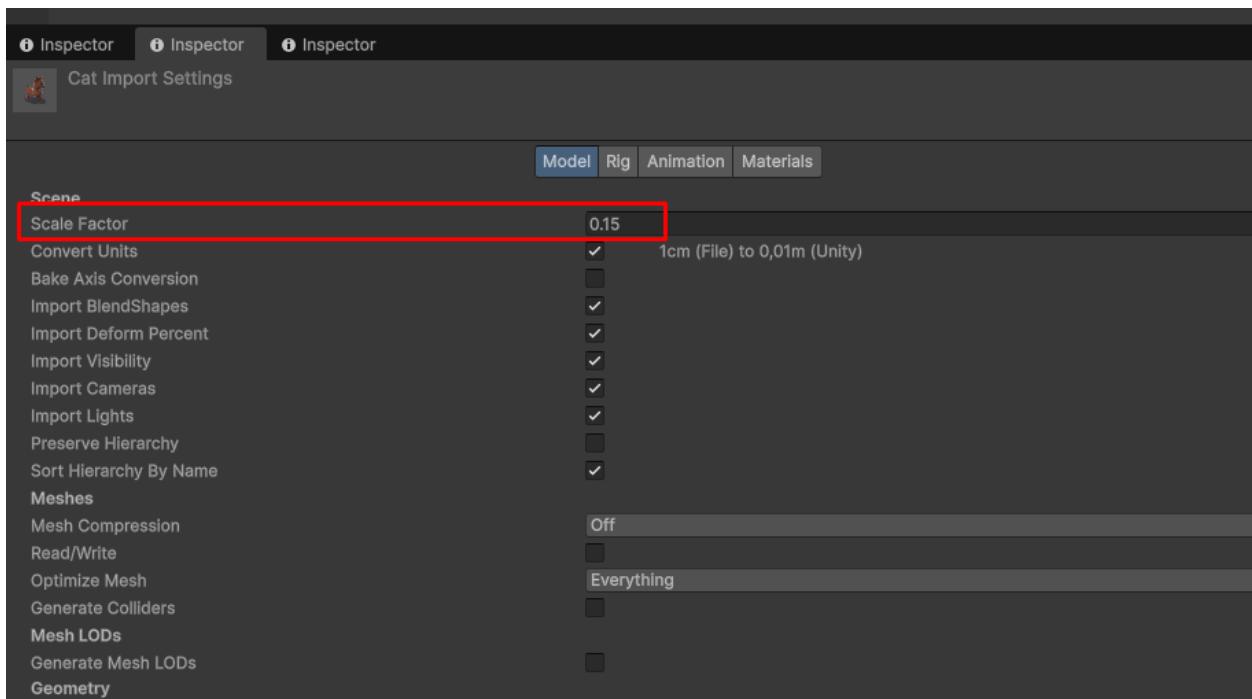
Далі виконується налаштування матеріалів та підключення текстур. Для цього у вікні Project обирається відповідний матеріал, після чого у вікні Inspector змінюються його параметри.

У полі Base Color встановлюється текстура кольору поверхні, яка визначає основний візуальний вигляд матеріалу. Якщо для моделі передбачено карту металевості, вона додається у поле Metallic, що дозволяє налаштовувати відбивання світла та характер поверхні. За наявності нормал-мапи вона підключається у поле Normal Map, після чого Unity автоматично виконує її перетворення у відповідний формат. Використання нормал-мапи дозволяє створити ілюзію дрібних деталей поверхні без збільшення кількості полігонів.

Після налаштування параметрів матеріалу перевіряється його коректне відображення на моделі. За необхідності матеріал перетягується на відповідний об'єкт у сцені або призначається через компонент Mesh Renderer у вікні Inspector. Аналогічні дії виконуються для матеріалів тунелю та ігрового персонажа.



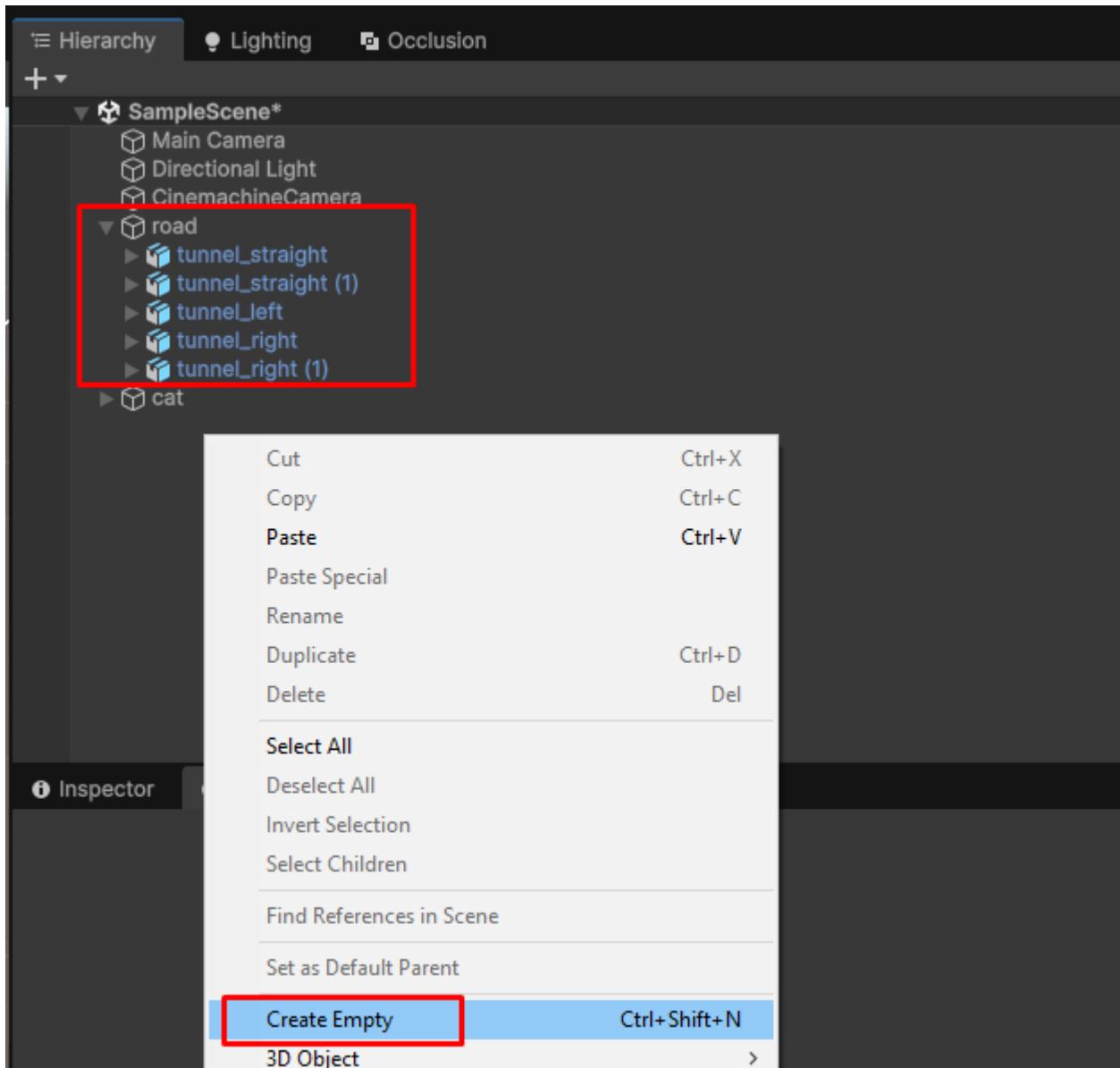
Після імпорту моделей і налаштування матеріалів необхідно скоригувати масштаб об'єктів відповідно до розмірів ігрової сцени. Часто тривимірні моделі імпортуються з розмірами, що не відповідають масштабам Unity. Для виправлення цього у вікні Project обирається файл моделі, після чого у вікні Inspector відкривається вкладка Model. У параметрі Scale Factor встановлюється відповідне значення, яке зменшує або збільшує модель до необхідних розмірів.



Після зміни масштабу застосовуються внесені зміни, і модель оновлюється у сцені. Такий підхід дозволяє зберегти правильні пропорції об'єкта без ручного масштабування кожного екземпляра.

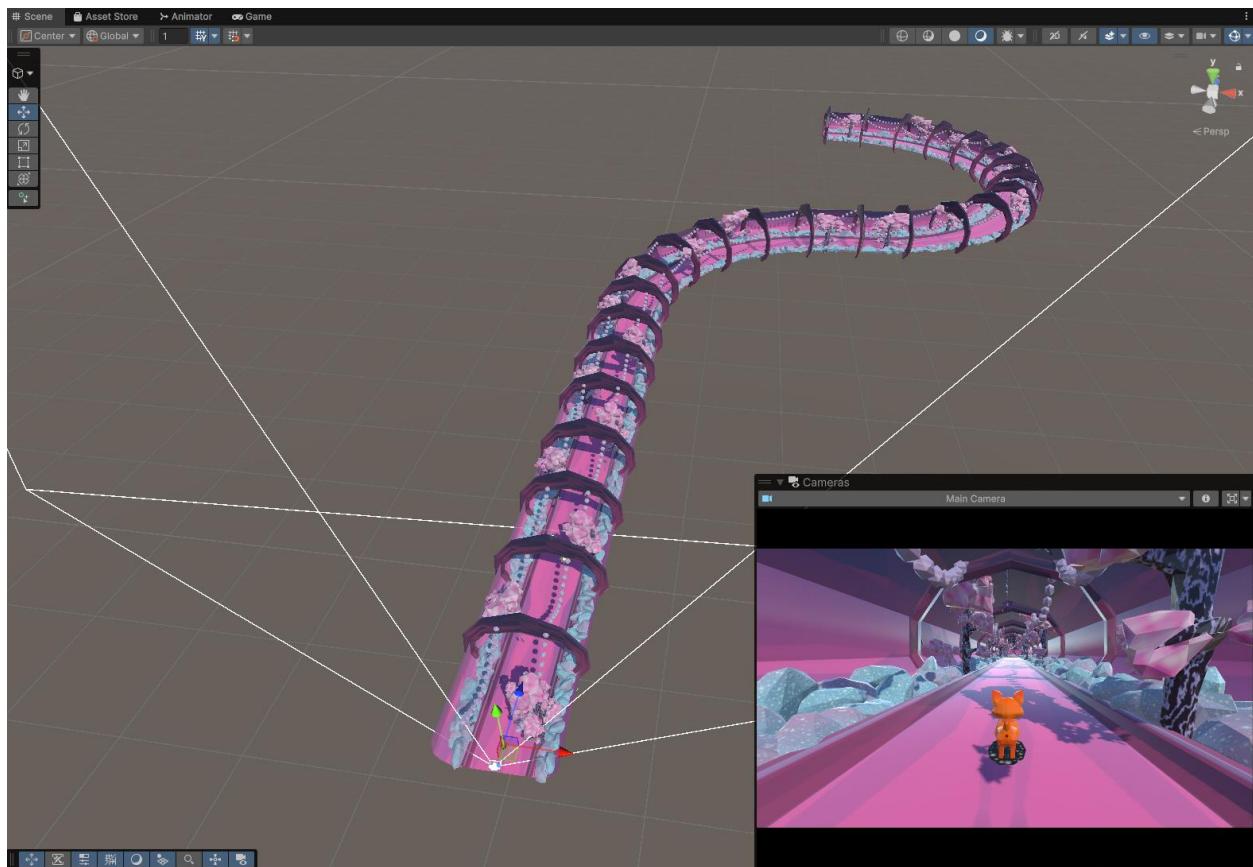
Для зручності організації сцени доцільно використовувати порожні об'єкти як контейнери для префабів. Для цього у вікні Hierarchy створюється порожній об'єкт за допомогою команди Create Empty. Такий об'єкт використовується як батьківський контейнер, у який поміщаються екземпляри префабів. Наприклад, для сегментів тунелю може бути створений окремий порожній об'єкт, який міститиме всі частини маршруту. Це дозволяє легко переміщати, масштабувати або вимикати групи об'єктів, а також підтримувати впорядковану ієархію сцени.

Після створення порожніх об'єктів відповідні префаби перетягуються у них у вікні Hierarchy. Позиція кожного сегмента тунелю налаштовується таким чином, щоб вони з'єднувалися між собою без зазорів і утворювали єдиний безперервний маршрут. Analogічний підхід може бути застосований і для ігрового персонажа, якщо в подальшому планується його заміна або використання декількох варіантів.



Додавання префабів і використання порожніх об'єктів як контейнерів значно спрощує подальшу роботу зі сценою та дозволяє масштабувати проект без втрати керованості.

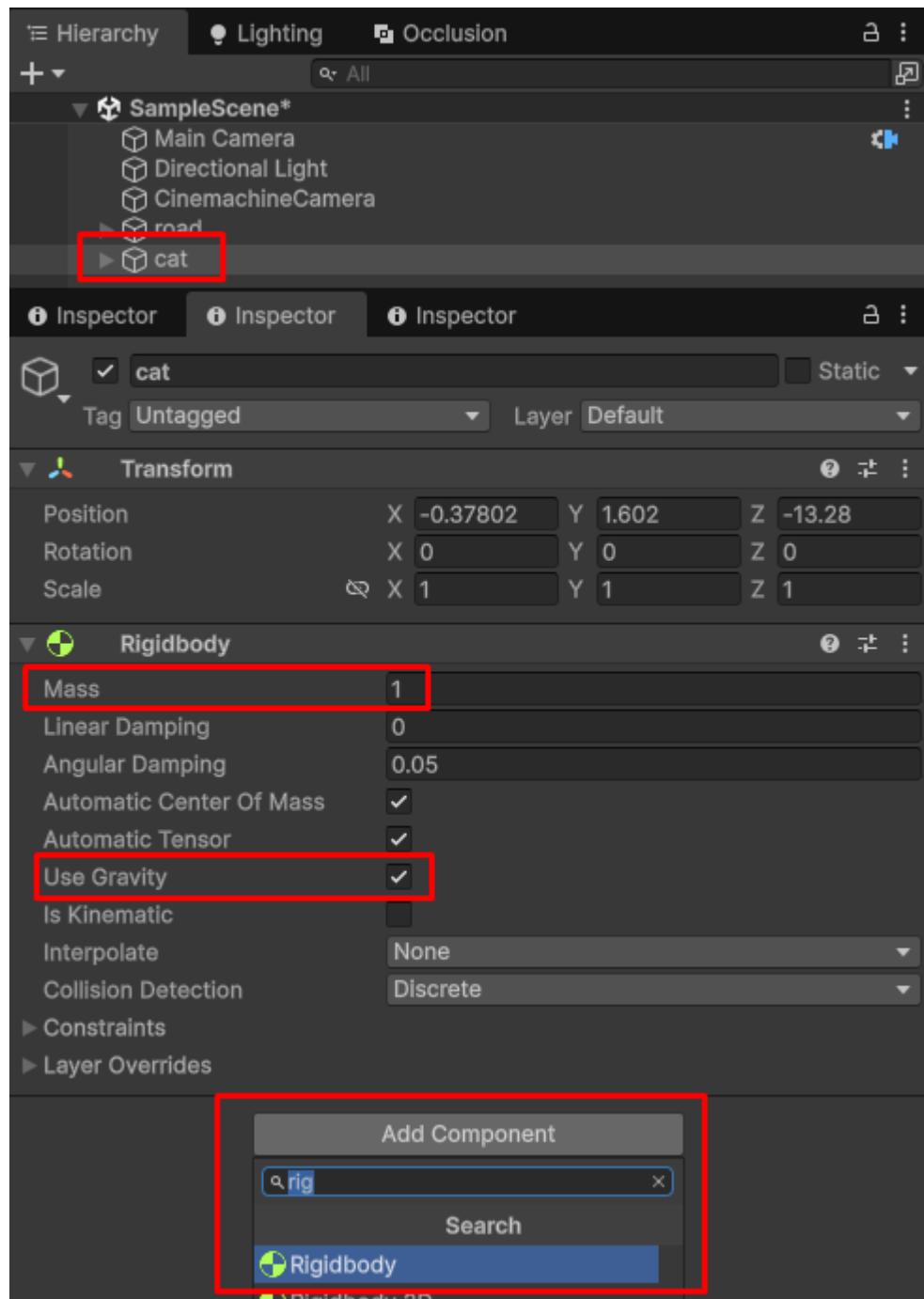
У результаті виконання третього етапу проект містить впорядковану структуру ресурсів, імпортовані моделі, коректно налаштовані матеріали з підключеними текстурами та базову сцену з побудованим маршрутом. Це створює основу для подальшого налаштування фізики, керування та ігрової логіки.



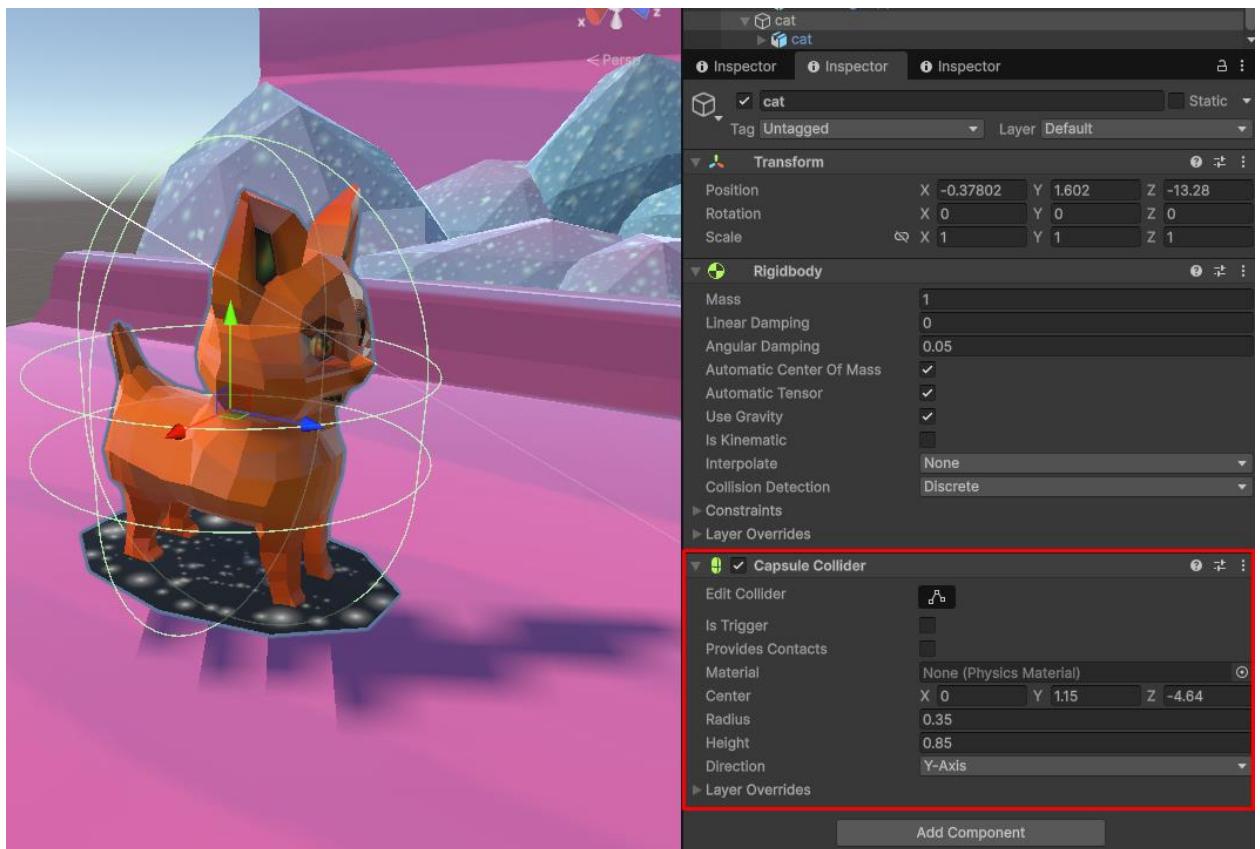
2.8 Налаштування фізики ігрового персонажа

На цьому етапі виконується додавання фізичної поведінки ігрового персонажа та реалізація його постійного руху вперед уздовж тунелю. Ігровий персонаж представлений у вигляді моделі котика, який переміщується всередині тунелю під дією керуючих сил та фізичних обмежень.

Насамперед на сцені обирається об'єкт ігрового персонажа у вікні Hierarchy. У вікні Inspector до нього додається компонент Rigidbody, який забезпечує участь об'єкта у фізичних обчисленнях Unity. Для коректної поведінки персонажа параметр Use Gravity залишається увімкненим, а значення Mass встановлюється у межах від 1 до 3, що забезпечує стабільний рух без надмірної інерції. Параметри Drag і Angular Drag залишаються стандартними на цьому етапі.

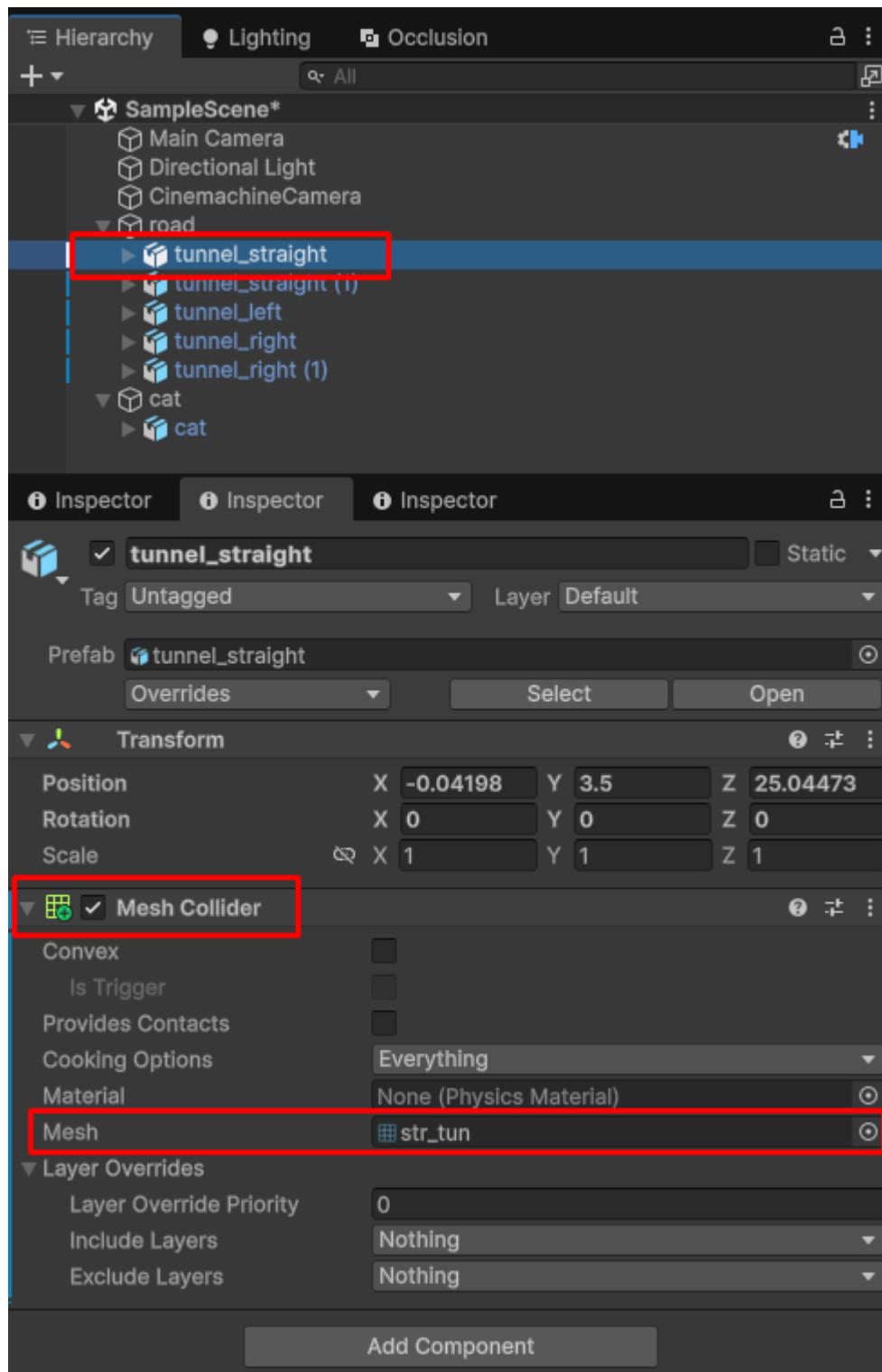


Далі до об'єкта додається компонент Collider, який відповідає за зіткнення з геометрією тунелю. Тип колайдера підбирається залежно від форми моделі. Для спрощення розрахунків рекомендується використовувати Capsule Collider або Sphere Collider, навіть якщо модель має складну форму. Колайдер масштабується таким чином, щоб він охоплював основну частину моделі котика, не виходячи за межі тунелю та не торкаючись його стінок у початковому положенні.



Після налаштування фізичних компонентів ігрового персонажа необхідно забезпечити коректну фізичну взаємодію з геометрією тунелю. Для цього на всі сегменти тунелю додаються колайдери, які визначають його стінки як фізичні перешкоди.

Кожен сегмент тунелю обирається у вікні Hierarchy, після чого у вікні Inspector перевіряється наявність компонента Collider. Якщо модель тунелю була імпортована з тривимірного редактора, зазвичай Unity автоматично додає до неї Mesh Collider. У випадку його відсутності компонент Mesh Collider додається вручну. Mesh Collider використовується для точної відповідності складній формі тунелю та забезпечує коректне зіткнення персонажа зі стінками.



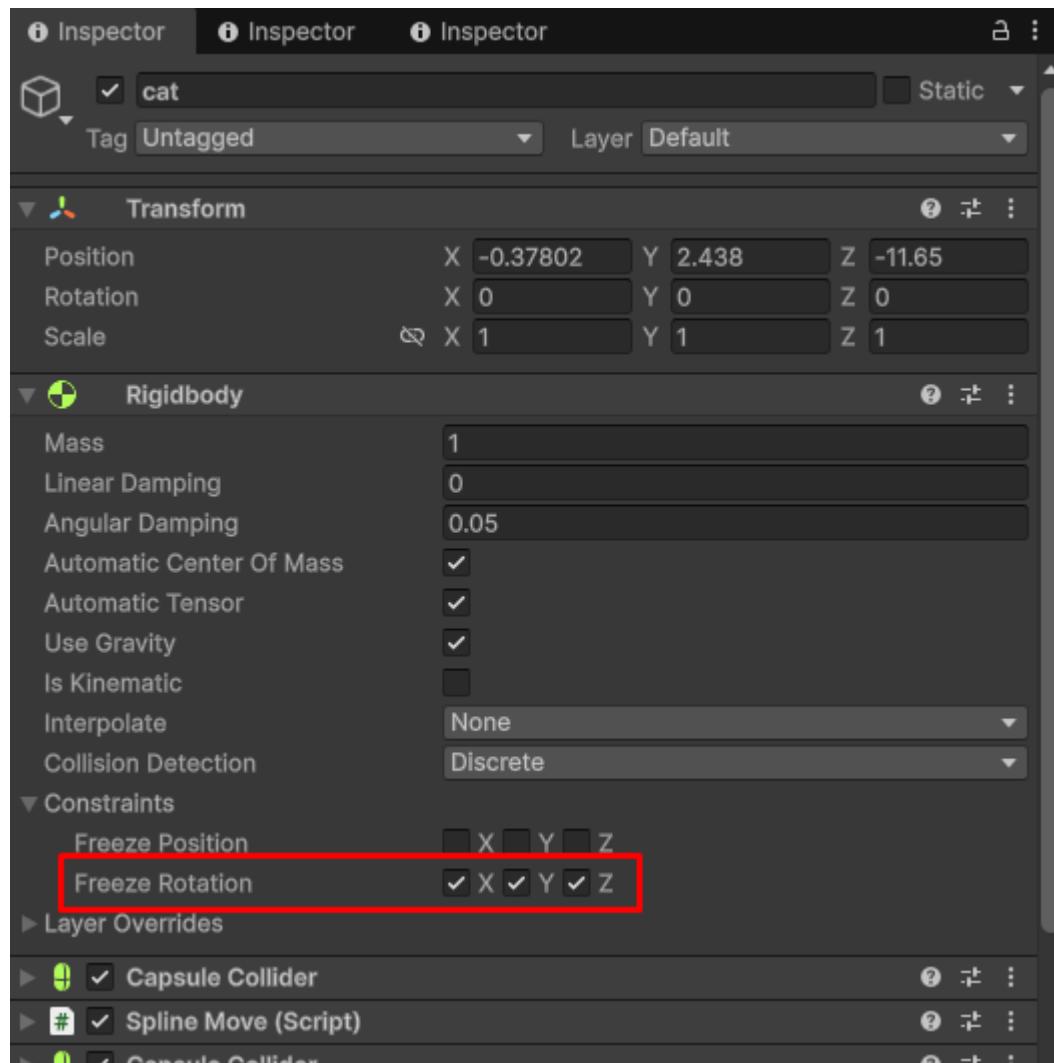
Після додавання Mesh Collider необхідно перевірити параметр Convex. Для статичних об'єктів сцени, якими є сегменти тунелю, цей параметр повинен бути вимкнений. Це дозволяє використовувати складну геометрію колайдера без обмежень та забезпечує точну фізичну взаємодію. Також

перевіряється, що параметр Is Trigger вимкнений, оскільки стінки тунелю мають фізично зупиняти персонажа, а не лише фіксувати факт проходження.

Для оптимізації продуктивності всі сегменти тунелю повинні залишатися нерухомими. Якщо на них автоматично було додано компонент Rigidbody, його необхідно видалити. Тунель у грі є статичним об'єктом, і його фізична поведінка реалізується виключно через колайдери.

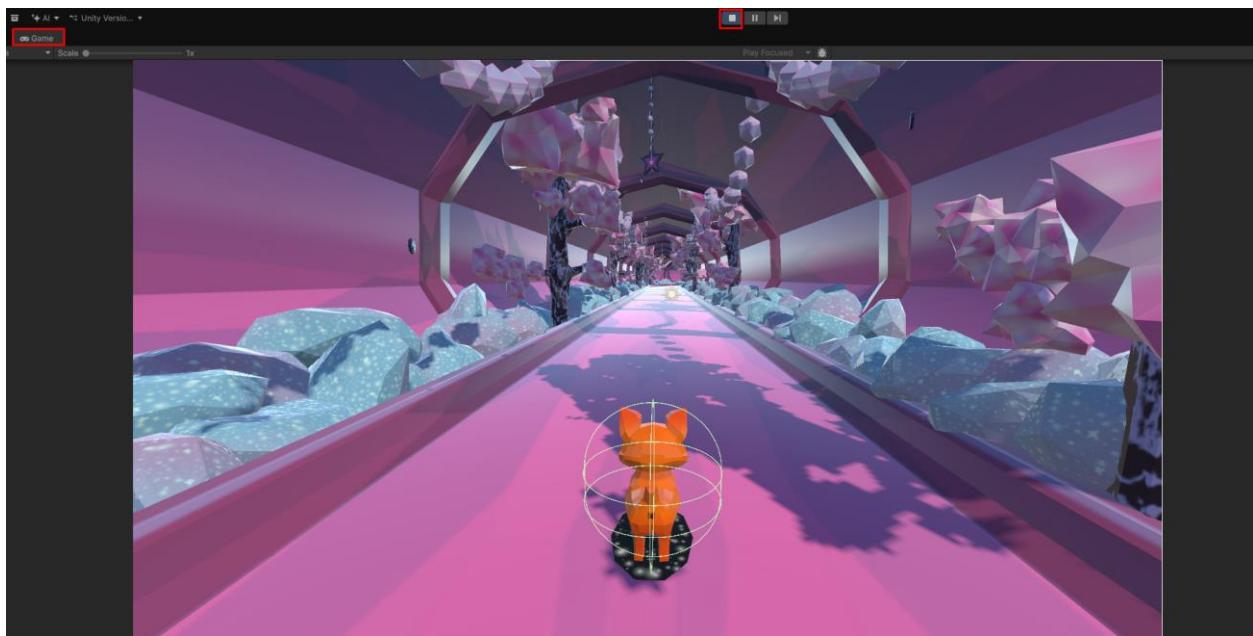
Після налаштування колайдерів сцена запускається в режимі відтворення для перевірки коректності взаємодії. Ігровий персонаж повинен залишатися всередині тунелю, не проходити крізь його стінки та стабільно реагувати на гравітацію. У разі виявлення зазорів або некоректних зіткнень розміри колайдерів або масштаб моделей коригуються. При запуску сцени ігровий персонаж може одразу перекидатись на бік або обертатися навколо своєї осі. Така поведінка пов'язана з фізичними обчислennями Unity і є нормальнюю для об'єктів зі складною формою.

Для запобігання обертанню персонажа необхідно обмежити його фізичне обертання. Для цього у вікні Hierarchy обирається об'єкт ігрового персонажа, після чого у вікні Inspector відкривається компонент Rigidbody. У секції Constraints активуються параметри Freeze Rotation по осіах X та Z. Це дозволяє персонажу залишатися у вертикальному положенні та запобігає його падінню на бік, зберігаючи при цьому можливість руху вздовж тунелю.



У деяких випадках також може бути доцільним заморозити обертання по осі Y, якщо механіка гри не передбачає поворот персонажа навколо вертикальної осі. Остаточний вибір обмежень залежить від бажаної поведінки персонажа під час руху.

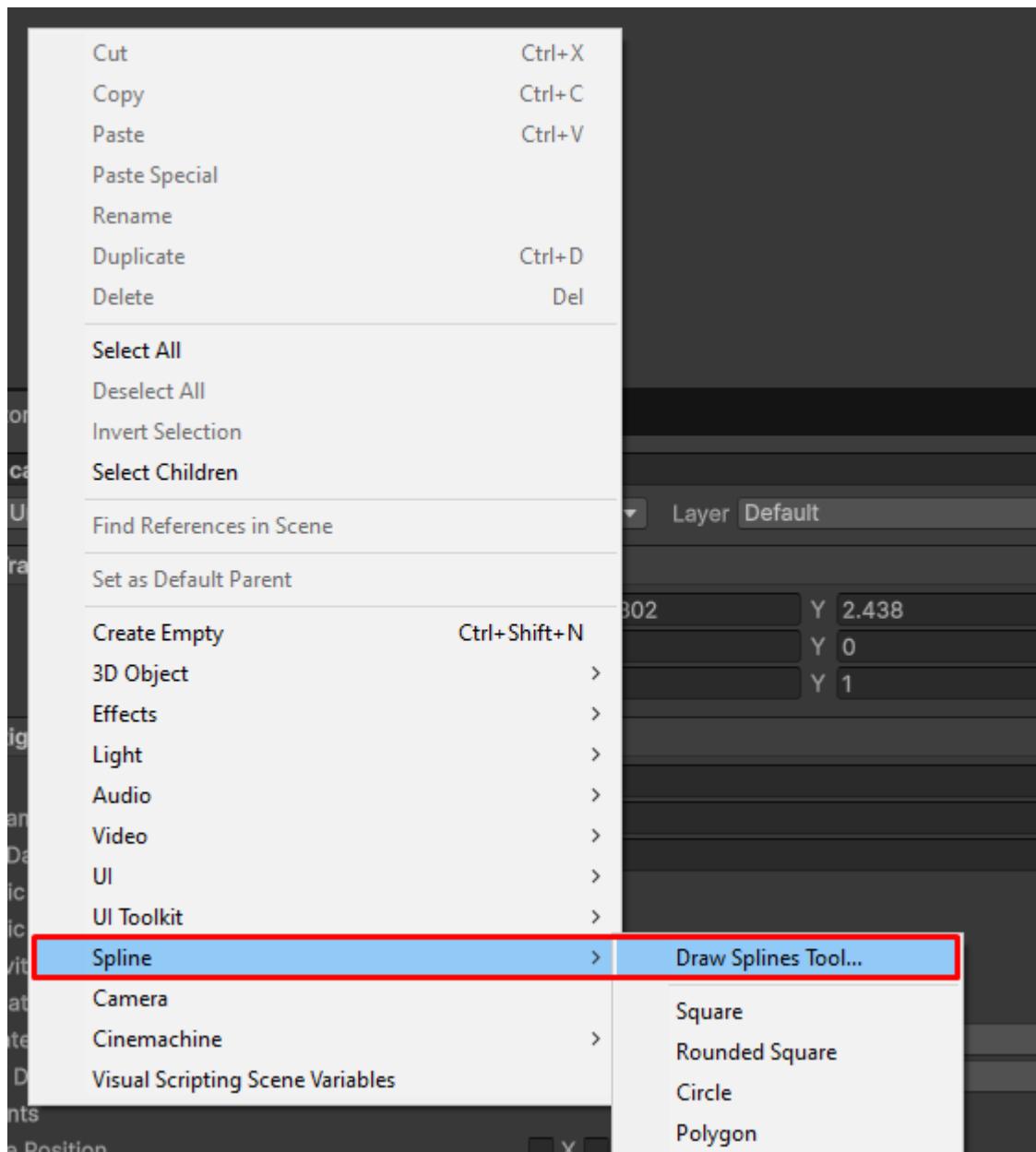
Після застосування обмежень сцена знову запускається в режимі відтворення для перевірки стабільності положення персонажа. Ігровий персонаж повинен зберігати правильну орієнтацію та не перекидатись під дією гравітації або під час контакту зі стінками тунелю.



2.9 Створення маршруту тунелю за допомогою сплайна

Для задання траєкторії руху персонажа у звивистому тунелі використовується система сплайнів Unity. Сплайн визначає центральну лінію маршруту, вздовж якої буде переміщуватися персонаж та орієнтуватися камера. Побудова маршруту виконується без програмування, безпосередньо у вікні сцени за допомогою вбудованих інструментів редактування.

Для створення сплайна у вікні Hierarchy створюється порожній об'єкт сцени Draw spline tools. Після цього у вікні Inspector до об'єкта додається компонент Spline Container.



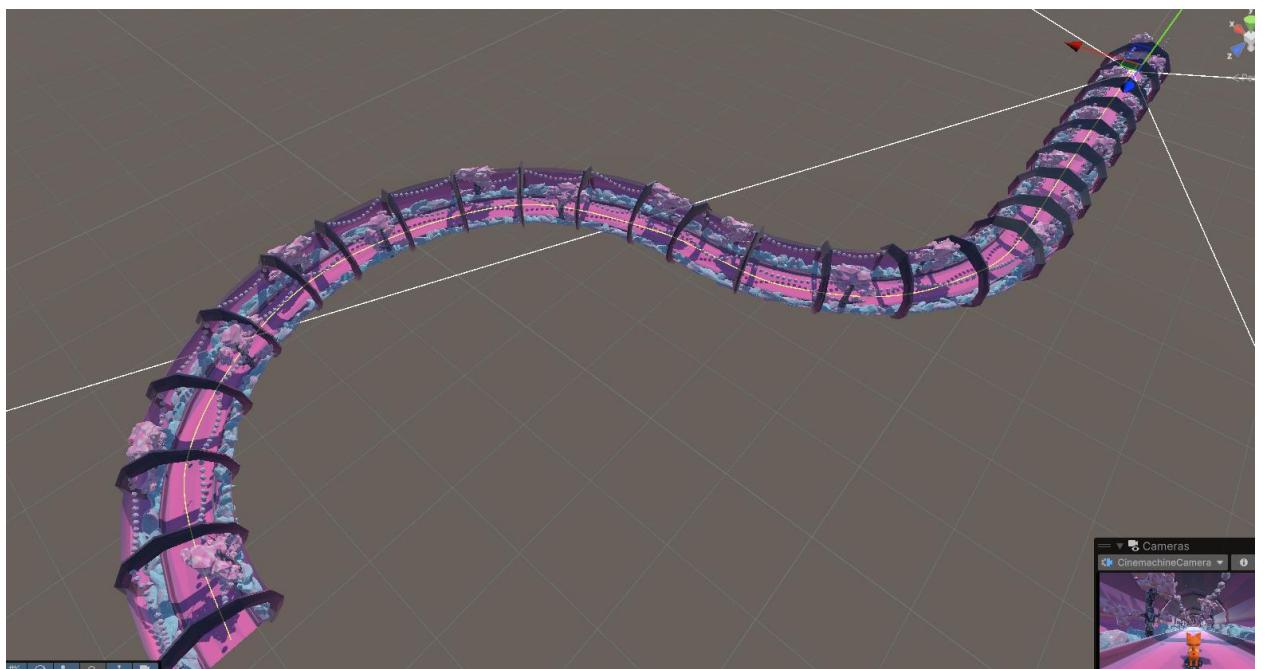
У верхній частині вікна Scene активуються інструменти редагування сплайнів. Для побудови маршруту використовується інструмент Draw Spline Tool, який дозволяє безпосередньо малювати траєкторію руху у просторі сцени. Після активації цього інструмента клацанням миші у вікні Scene послідовно створюються контрольні точки сплайна, які формують загальний напрямок тунелю.

Під час малювання сплайна інструмент автоматично створює плавні сегменти між точками, що дозволяє отримати природні вигини траєкторії без різких зламів. За необхідності положення контрольних точок можна

змінювати шляхом їх переміщення у вікні Scene, коригуючи форму маршруту та кут поворотів.

Для подовження маршруту до сплайна додаються нові точки, а для створення складніших ділянок тунелю використовується більша кількість контрольних точок із меншим кроком між ними. Важливо забезпечити плавність кривизни сплайна, оскільки різкі зміни напрямку можуть негативно впливати на керованість персонажа та стабільність руху камери.

Після завершення побудови маршруту об'єкт зі сплайном використовується як опорна лінія для розміщення сегментів тунелю. Моделі прямого та поворотного тунелю вирівнюються вздовж сплайна таким чином, щоб їхня центральна вісь збігалася з траекторією руху. Це дозволяє легко створювати індивідуальні маршрути шляхом комбінування готових елементів.



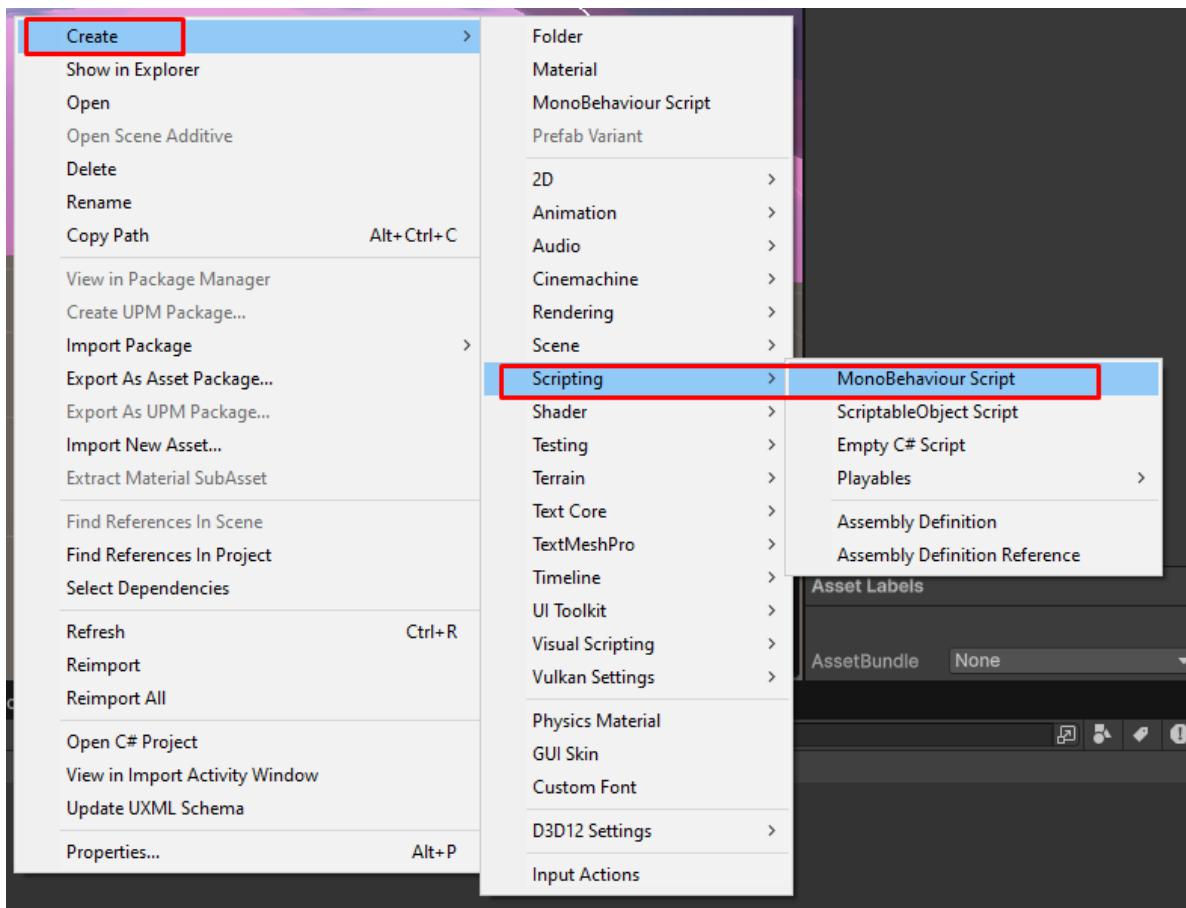
У результаті виконання цього етапу створюється повноцінний маршрут рівня, який може бути швидко змінений або розширеній без внесення змін у програмний код гри, що суттєво спрощує процес проєктування рівнів.

2.10 Реалізація руху персонажа вздовж сплайна

На цьому етапі реалізується основна механіка руху ігрового персонажа вздовж тунелю, побудованого за допомогою сплайна. Персонаж рухається автоматично вперед уздовж заданої траєкторії, тоді як керування гравця використовується для бічного зміщення персонажа з метою уникнення зіткнень зі стінками тунелю. Такий підхід дозволяє забезпечити стабільний рух навіть на плавних або різких поворотах маршруту.

Для керування рухом створюється спеціальний програмний скрипт, який відповідає за автоматичне переміщення персонажа вперед уздовж сплайна, а також за бічне зміщення персонажа під дією введення з боку гравця. Усі обчислення виконуються на основі поточного положення на сплайні та його дотичної, що дозволяє визначати напрямок руху навіть на складних ділянках маршруту.

Спочатку у вікні Project відкривається папка Assets. У ній створюється папка Scripts, якщо вона ще не існує. Далі у папці Scripts створюється новий C#-скрипт за допомогою команди Create → MonoBehaviour Script, якому надається назва SplineMove. Після цього файл відкривається для редагування.



У створений скрипт вставляється наведений нижче програмний код, який реалізує рух персонажа вздовж сплайна, бічне керування та підтримку керування як з клавіатури, так і за допомогою нахилу мобільного пристрою.

Переміщення вперед відбувається автоматично з постійною швидкістю та не залежить від дій гравця. Поточна позиція персонажа на сплайні визначається параметром прогресу, який поступово змінюється з часом. На основі цього параметра обчислюється просторове положення персонажа, а також вектор напрямку руху, відповідно до якого виконується обертання ігрового об'єкта.

Керування з боку гравця впливає лише на бічне зміщення персонажа відносно центральної осі тунелю. Для цього використовується напрямок, перпендикулярний до поточного напряму руху вздовж сплайна. Такий підхід дозволяє зберегти рух уздовж маршруту незалежно від його вигинів і забезпечує інтуїтивне керування в умовах звивистого тунелю.

Чутливість керування визначається параметром sensitivity, який задає інтенсивність реакції персонажа на введення. Під час тестування у середовищі Unity керування здійснюється за допомогою клавіатури, а під час запуску гри на мобільному пристрой — за допомогою нахилу смартфона, який читається з акселерометра. Зміна значення цього параметра дозволяє адаптувати керування під різні типи пристройів і вподобання гравця.

Для обмеження допустимого бічного руху використовується параметр maxOffset, який задає максимальну відстань від центральної лінії сплайна, на яку персонаж може зміщуватися вліво або вправо. Це запобігає виходу персонажа за межі тунелю та забезпечує коректну взаємодію з колайдерами стін.

Фізичне переміщення персонажа реалізується з використанням компонента Rigidbody. Усі операції руху виконуються у методі FixedUpdate, що забезпечує стабільну роботу фізики та усуває ривки руху. Орієнтація персонажа автоматично оновлюється відповідно до напряму сплайна, що створює відчуття природного руху вздовж тунелю.

Нижче наведено повний програмний код скрипта руху персонажа вздовж сплайна:

```
using UnityEngine;
using UnityEngine.Splines;

[RequireComponent(typeof(Rigidbody))]
public class SplineMove : MonoBehaviour
{
    public SplineContainer spline; // Сама траса (сплайн), по якій
                                   // рухатиметься об'ект
    public float forwardSpeed = 6f; // Швидкість руху вперед по сплайну
    public float sensetivity = 5f; // Чутливість руху вбік
                                   // (ліворуч/праворуч)
    public float maxOffset = 2f; // Максимальне бокове відхилення від
                               // центру сплайна
```

```

private float t; // Параметр руху по сплайну (0..1)
private float lateralOffset; // Поточне бокове зміщення від центру
сплайна
private Rigidbody rb; // Компонент Rigidbody для фізичного руху

void Awake()
{
    rb = GetComponent<Rigidbody>(); // Отримуємо Rigidbody

    // Налаштування Rigidbody (за потреби)
    // rb.useGravity = false;
    // rb.interpolation = RigidbodyInterpolation.Interpolate;
    // rb.collisionDetectionMode = CollisionDetectionMode.Continuous;

    // Якщо пристрій підтримує гіроскоп, активуємо його
    if (SystemInfo.supportsGyroscope)
        Input.gyro.enabled = true;
}

void FixedUpdate()
{
    float dt = Time.fixedDeltaTime; // Час між кадрами фізики
    float length = spline.Spline.GetLength(); // Загальна довжина
сплайна

    // === Отримання вводу гравця ===
    float input = GetHorizontalInput(); // Горизонтальний ввід
(клавіші або нахил пристрою)

    // Розрахунок бокового зміщення об'єкта
    lateralOffset += input * sensitivity * dt;
    lateralOffset = Mathf.Clamp(lateralOffset, -maxOffset,
maxOffset); // Обмеження бокового зміщення

    // === Рух вперед по сплайну ===
    t += (forwardSpeed / length) * dt; // Пропорційне збільшення t
по сплайну
    t = Mathf.Clamp01(t); // Обмеження t значенням від 0 до 1
}

```

```

    Vector3 splinePos = spline.EvaluatePosition(t); // Поточна
позиція на сплайні

    Vector3 forward = ((Vector3)spline.EvaluateTangent(t)).normalized; // Вектор напрямку
вперед по сплайну

    Vector3 right = Vector3.Cross(Vector3.up, forward).normalized;
// Вектор вправо від напрямку сплайна

    // === Цільова позиція об'єкта з урахуванням бокового зміщення
====

    Vector3 targetPos = splinePos + right * lateralOffset;

    // === Фізичний рух до цільової позиції ===
    Vector3 move = (targetPos - rb.position) * 12f; // Розрахунок
вектору руху

    rb.linearVelocity = new Vector3(move.x, 0f, move.z); // //
Застосування лінійної швидкості без зміни висоти

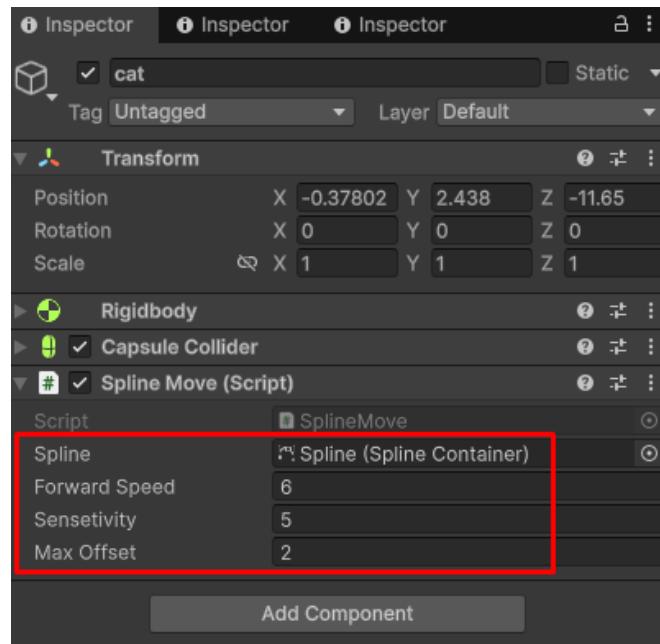
    // === Поворот об'єкта у напрямку руху ===
    rb.MoveRotation(Quaternion.LookRotation(forward, Vector3.up));
}

float GetHorizontalInput()
{
#if UNITY_ANDROID && !UNITY_EDITOR
    // На Android використовуємо нахил пристрою через гіроскоп
    if (SystemInfo.supportsGyroscope)
        return Mathf.Clamp(Input.gyro.gravity.x, -1f, 1f);
    else
        return 0f; // Якщо гіроскоп не підтримується, руху немає
#else
    // На ПК використовуємо клавіші або джойстик
    return Input.GetAxis("Horizontal");
#endif
}

}

```

Після збереження скрипта необхідно призначити його ігровому персонажу. Для цього об'єкт персонажа обирається у вікні Hierarchy, після чого скрипт SplineMove перетягується на нього у вікні Inspector. У цьому ж вікні в полі Spline вказується об'єкт маршруту, який містить компонент SplineContainer.



Після завершення налаштувань сцена запускається у режимі Play для перевірки роботи механіки руху. У результаті виконання цього етапу реалізовано базовий рух персонажа вздовж звивистого тунелю, готовий до додавання системою зіткнень та підрахунку балів.

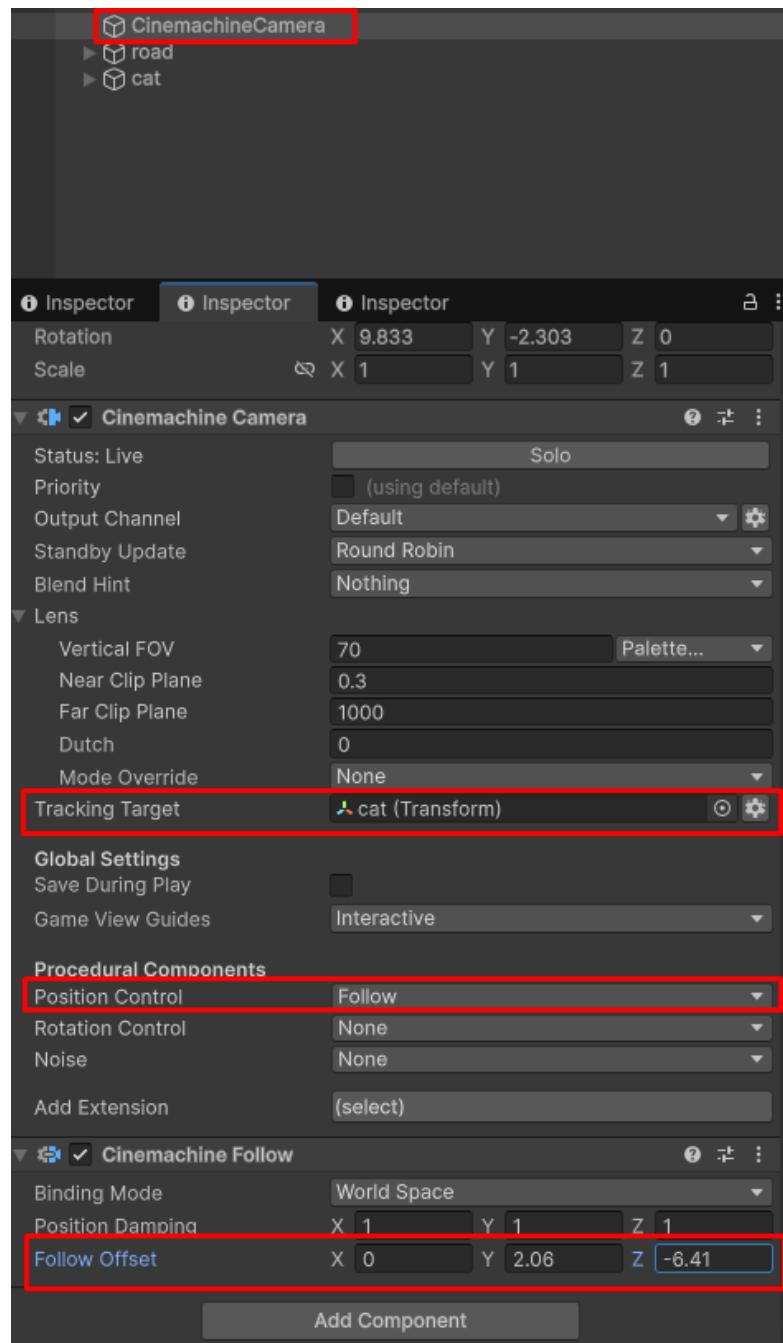
2.11 Налаштування камери для слідування за персонажем

Для забезпечення коректного відображення ігрового процесу необхідно налаштувати камеру так, щоб вона автоматично слідувала за персонажем під час його руху тунелем.

У вікні Inspector для об'єкта CinemachineCamera у полі Tracking Target необхідно вказати трансформ персонажа (котика), перетягнувши його зі списку Hierarchy. Це визначає об'єкт, за яким камера буде слідувати.

У секції Procedural Components у параметрі Position Control обирається компонент Cinemachine Follow. Це активує механізм слідування камери за персонажем. Параметр Rotation Control на цьому етапі залишається значенням None, оскільки обертання камери не використовується.

У компоненті Cinemachine Follow встановлюється режим Binding Mode = World Space. Параметр Follow Offset задає зміщення камери відносно персонажа. Ці значення визначають положення камери позаду та трохи вище персонажа і можуть коригуватись під час тестування сцени.



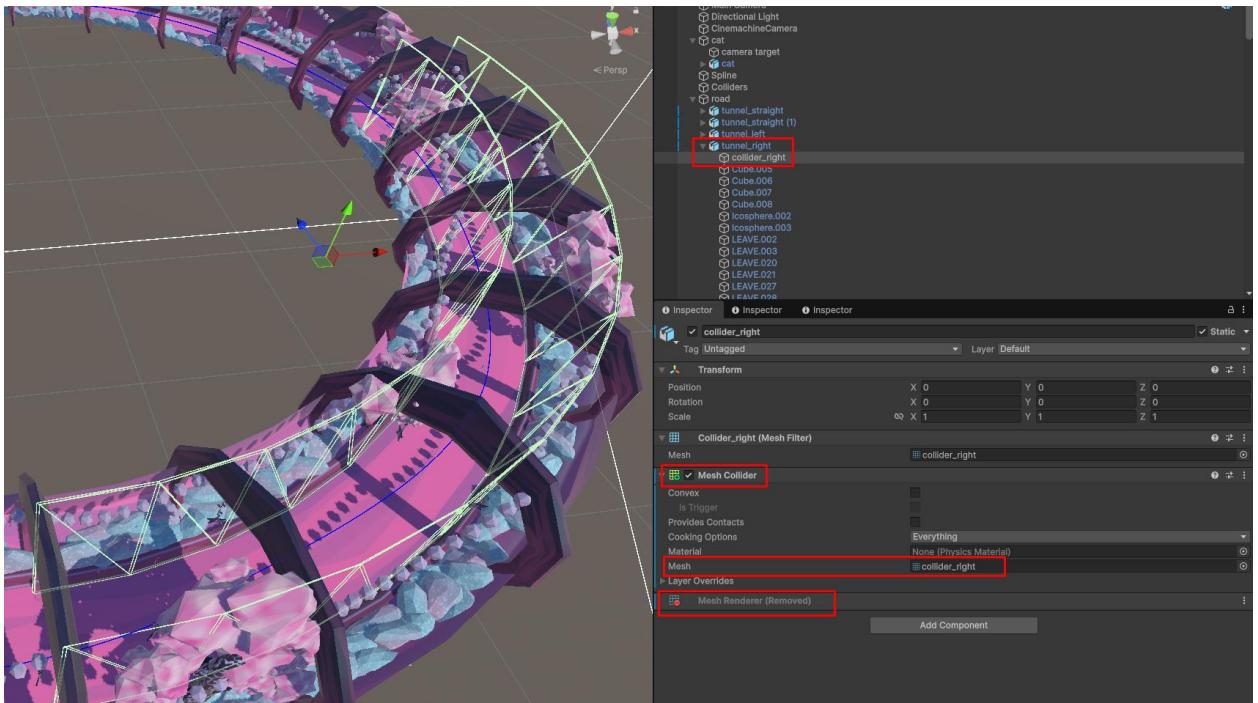
Після завершення налаштувань необхідно запустити сцену в режимі Play та перевірити, що камера плавно слідує за персонажем під час його руху і зберігає стабільний огляд ігрового простору.

2.12 Обробка зіткнень і визначення помилок гравця

На цьому етапі реалізується механізм виявлення помилок гравця під час руху персонажа звивистим тунелем. Помилкою вважається будь-який дотик ігрового персонажа до стінок тунелю. Інформація про такі зіткнення в подальшому використовується для нарахування штрафних балів та формування загального ігрового рахунку.

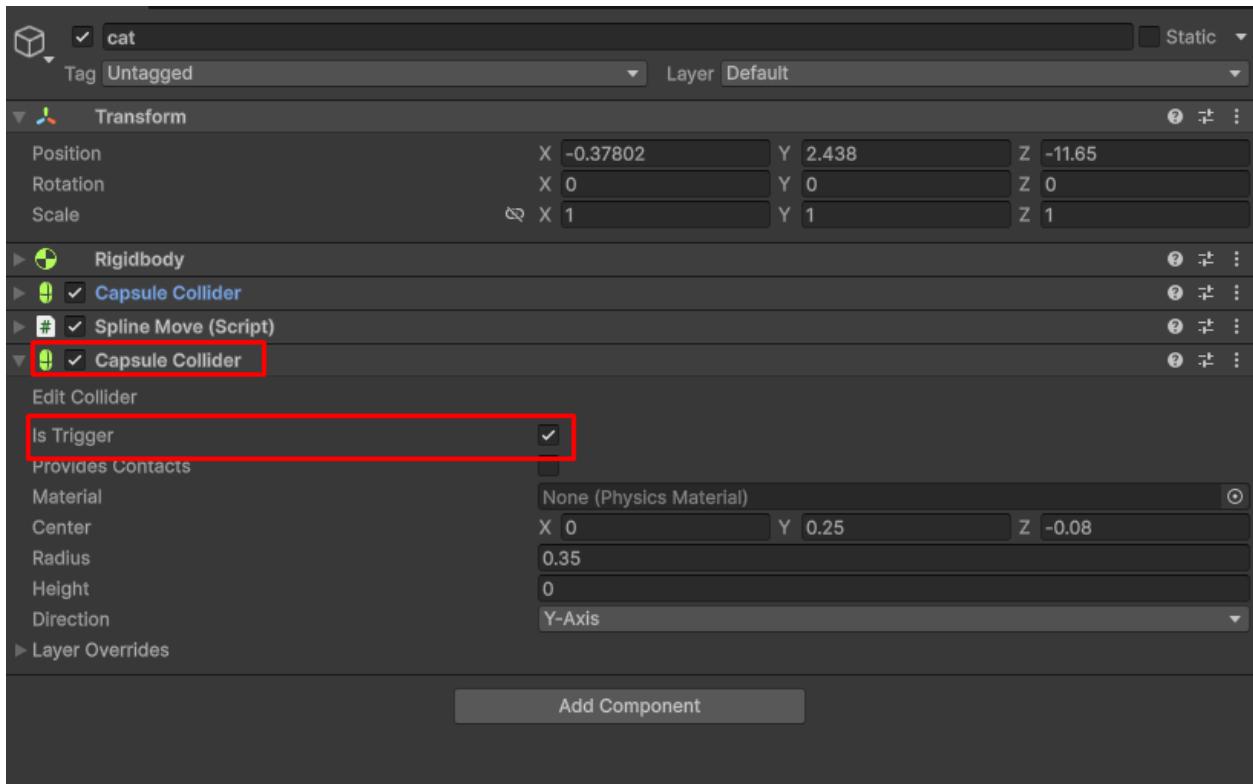
З огляду на складну вигнуту форму тунелю та обмеження фізичного рушія Unity, обробка зіткнень реалізується без використання тригерних Mesh Collider на стінках. Натомість застосовується більш стабільний і рекомендований підхід, при якому тригерний колайдер розміщується безпосередньо на ігровому персонажі, а стінки тунелю залишаються звичайними фізичними колайдерами.

Для коректної фізичної взаємодії зігнутого тунелю використовуються спеціальні допоміжні моделі, що повторюють допустимі межі руху персонажа. Ці моделі призначені виключно для фізичних розрахунків і не повинні відображатися у грі. У вікні Hierarchy обираються відповідні об'єкти допоміжних стінок тунелю. У вікні Inspector з них видаляється або вимикається компонент Mesh Renderer, що робить ці об'єкти невидимими під час гри. Після цього до кожного з таких об'єктів додається компонент Mesh Collider, який забезпечує точну відповідність вигнутій формі тунелю.



Параметр Is Trigger для цих колайдерів залишається вимкненим, а параметр Convex також не використовується. Таким чином зберігається правильна геометрія тунелю та забезпечується стабільна робота фізики навіть на складних поворотах маршруту.

Для виявлення моменту дотику до стінок тунелю тригерний колайдер розміщується безпосередньо на ігровому персонажі. У вікні Hierarchy обирається об'єкт персонажа, після чого у вікні Inspector до нього додається додатковий компонент Collider.



Рекомендується використовувати Capsule Collider, оскільки він добре підходить для об'єктів з округлою формою та забезпечує плавну взаємодію зі стінками тунелю. Для цього колайдера активується параметр Is Trigger. Розміри тригерного колайдера налаштовуються таким чином, щоб він трохи перевищував основний фізичний колайдер персонажа. Це дозволяє фіксувати зіткнення ще до того, як персонаж візуально почне проникати у стінку.

Основний колайдер персонажа, який використовується для фізичного руху, залишається без змін і працює разом з компонентом Rigidbody. Таким чином тригерний колайдер відповідає лише за логіку помилок, а не за фізичне переміщення.

Після налаштування колайдерів необхідно реалізувати програмну логіку, яка реагує на вхід тригерного колайдера персонажа у межі стінок тунелю. Для цього у папці Scripts створюється новий C#-скрипт з назвою CollisionHandler.

Створений скрипт відкривається для редагування. У ньому використовується метод OnTriggerEnter, який автоматично викликається Unity

у момент, коли тригерний колайдер персонажа входить у контакт з будь-яким іншим колайдером сцени.

```
using UnityEngine;

public class CollisionHandler : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        Debug.Log("Зіткнення зі стінкою тунелю");
    }
}
```

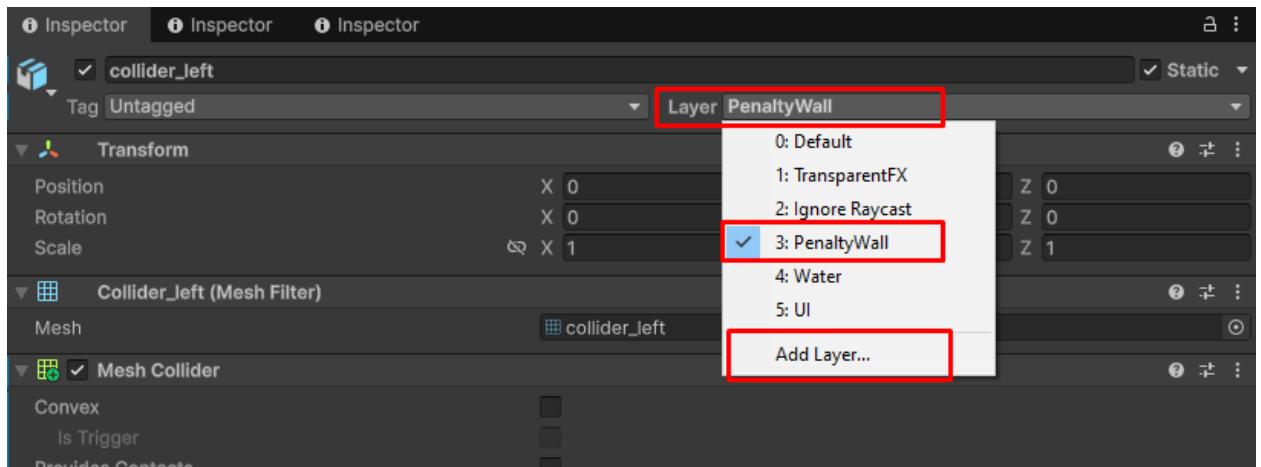
Даний код виконує мінімальну перевірку і виводить повідомлення у консоль у момент зіткнення. На цьому етапі цього достатньо, щоб переконатися в коректній роботі системи. Логіка підрахунку штрафів буде реалізована на наступному етапі. Після збереження скрипта CollisionHandler він перетягується на об'єкт ігрового персонажа у вікні Hierarchy. Таким чином обробка тригерних подій прив'язується безпосередньо до котика. Далі сцена запускається у режимі Play. Під час руху персонажа тунелем у момент дотику до стінок у вікні Console повинно з'являтися повідомлення про зіткнення. Це підтверджує, що тригерний колайдер персонажа коректно взаємодіє з Mesh Collider допоміжних стінок тунелю.

Тригерний колайдер, розміщений на ігровому персонажі, спрацьовує при вході у контакт з будь-яким колайдером, незалежно від його призначення. Оскільки на тунелі присутній загальний Mesh Collider, який відповідає за фізичне обмеження руху, тригер буде постійно перебувати у стані контакту з ним. У такій ситуації Unity викликає події тригера при кожному оновленні фізики, що призводить до багаторазової фіксації зіткнень і некоректного нарахування штрафів. Для виправлення цього, у вікні Hierarchy обирається

будь-який об'єкт допоміжної стінки. У верхній частині вікна Inspector у полі Layer створюється новий шар PenaltyWall.



Після створення шару він призначається всім допоміжним стінкам, які відповідають за фіксацію помилок. Загальний тунель при цьому залишається на стандартному шарі Default або іншому шарі, який не використовується для штрафів.



Після налаштування шарів необхідно оновити логіку скрипта CollisionHandler таким чином, щоб він реагував лише на колайдери з відповідного шару.

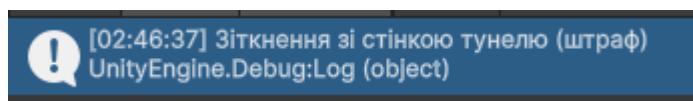
Оновлений код виглядає наступним чином:

```
using UnityEngine;

public class CollisionHandler : MonoBehaviour
{
    [SerializeField] private string wallLayerName = "PenaltyWall";

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer(wallLayerName))
        {
            Debug.Log("Зіткнення зі стінкою тунелю (штраф)");
        }
    }
}
```

Для уникнення багаторазової фіксації зіткнень тригерний колайдер персонажа реагує лише на допоміжні стінки, які мають спеціально визначений шар. Загальна геометрія тунелю, що використовується для фізичного обмеження руху, ігнорується системою штрафів.



У результаті виконання цього етапу реалізовано стабільну систему виявлення помилок гравця, яка коректно працює з вигнутим тунелем, не потребує спрошення геометрії.

2.13 Реалізація системи підрахунку балів

На цьому етапі реалізується система підрахунку балів, яка визначає успішність проходження маршруту. За кожну умовну дистанцію руху персонажа без зіткнень нараховується один бал. У випадку дотику до стінки

тунелю рахунок зменшується на два бали, а нарахування позитивних балів тимчасово припиняється. Такий підхід стимулює гравця рухатися акуратно та уникати помилок.

Для гнучкого керування складністю гри значення дистанції, за яку нараховується один бал, виноситься у параметри компонента і налаштовується через вікно Inspector без необхідності редагування коду.

У папці Scripts створюється новий C#-скрипт з назвою ScoreManager. Даний скрипт відповідає за зберігання поточного рахунку, відстеження пройденої дистанції та обробку штрафів. Після створення файл відкривається для редагування.

```
using UnityEngine;

public class ScoreManager : MonoBehaviour
{
    [SerializeField] private float distanceForOnePoint = 1f;

    [SerializeField] private Transform player;

    public int score = 0;
    public bool canScore = true;

    private float lastPosition;
    private float accumulatedDistance = 0f;

    void Start()
    {
        lastPosition = player.position.z;
    }

    void Update()
    {
        if (!canScore)
        {
            lastPosition = player.position.z;
            return;
        }
    }
}
```

```

    }

    float currentPosition = player.position.z;
    float delta = currentPosition - lastPosition;

    if (delta > 0f)
    {
        accumulatedDistance += delta;

        while (accumulatedDistance >= distanceForOnePoint)
        {
            score += 1;
            accumulatedDistance -= distanceForOnePoint;
        }
    }

    lastPosition = currentPosition;
}

public void ApplyPenalty()
{
    score -= 2;
    canScore = false;
    Invoke(nameof(ResumeScoring), 0.5f);
}

private void ResumeScoring()
{
    canScore = true;
}

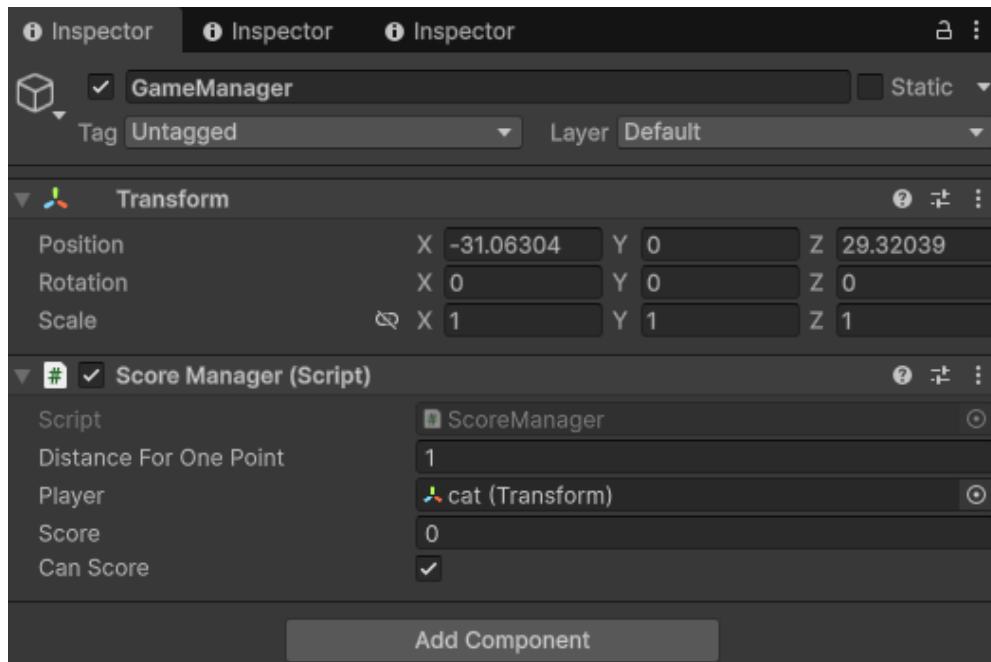
```

У цьому коді змінна `distanceForOnePoint` визначає, яку дистанцію необхідно пройти для отримання одного бала. Під час руху персонажа відстань поступово накопичується у змінній `accumulatedDistance`. Коли накопичене значення досягає заданого порогу, рахунок збільшується на один бал. Таким

чином система коректно працює при будь-якій швидкості руху та не втрачає пройдену дистанцію.

Метод `ApplyPenalty` використовується для застосування штрафу. Він зменшує рахунок на два бали та тимчасово блокує нарахування позитивних балів. Через короткий проміжок часу нарахування автоматично відновлюється.

Після збереження скрипту `ScoreManager` додається до порожнього об'єкта сцени, наприклад `GameManager`. У полі `Player` у вікні `Inspector` необхідно вказати об'єкт ігрового персонажа.



Для застосування штрафів використовується скрипт `CollisionHandler`, який призначається ігровому персонажу. Він реагує на спрацювання тригерного колайдера та перевіряє, чи належить об'єкт, з яким відбулося зіткнення, до шару штрафних стінок.

```
using UnityEngine;
public class CollisionHandler : MonoBehaviour
{
    [SerializeField] private string wallLayerName = "PenaltyWall";
```

```

[SerializeField] private ScoreManager scoreManager;
private bool collisionBlocked = false;
private void OnTriggerEnter(Collider other)
{
    if (collisionBlocked)
        return;

    if (other.gameObject.layer == 
LayerMask.NameToLayer(wallLayerName) )
    {
        collisionBlocked = true;
        scoreManager.ApplyPenalty();
        Invoke(nameof(ResetCollision), 0.5f);
    }
}

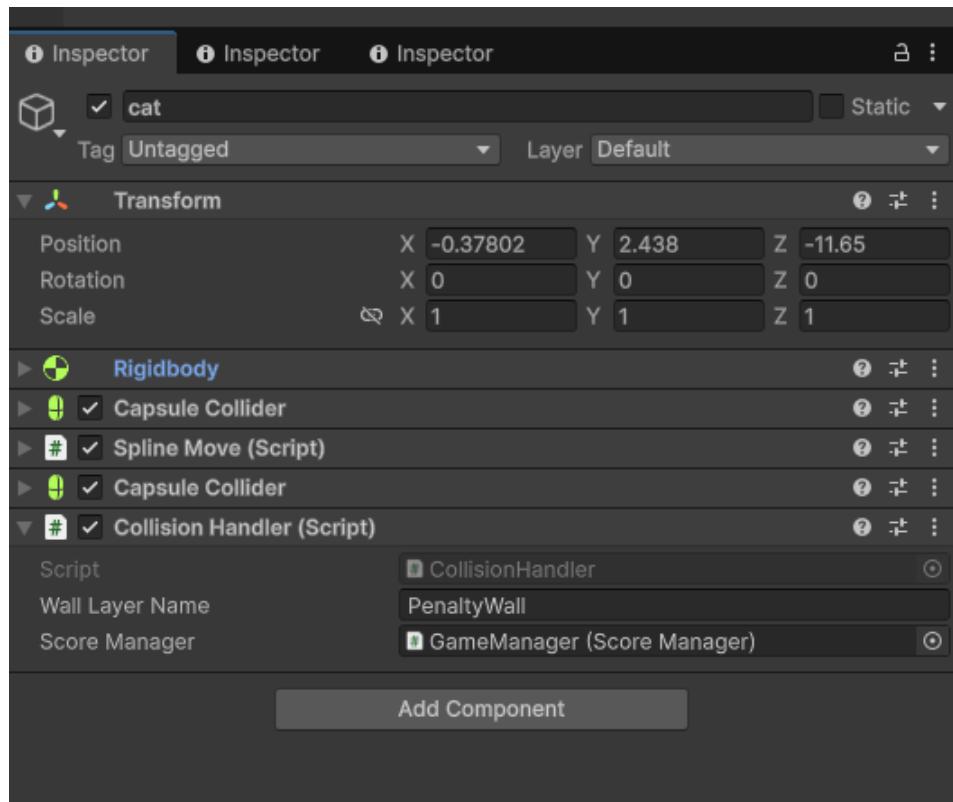
private void ResetCollision()
{
    collisionBlocked = false;
}
}

```

Скрипт перевіряє шар об'єкта, з яким відбувся контакт. Якщо це допоміжна стінка тунелю, рахунок зменшується, а повторні штрафи за одне зіткнення тимчасово блокуються. Поле ScoreManager заповнюється через Inspector шляхом перетягування відповідного об'єкта сцени.

Після підключення скриптів у вікні Inspector з'являється параметр Distance For One Point. Зміна цього значення дозволяє керувати складністю гри. Менше значення призводить до швидшого нарахування балів, більше — ускладнює ігровий процес. Усі зміни застосовуються без перекомпіляції коду.

Під час запуску сцени персонаж починає рухатися тунелем, а рахунок збільшується відповідно до пройденої дистанції без зіткнень. У момент дотику до стінок тунелю рахунок зменшується на два бали, а нарахування позитивних балів тимчасово припиняється. Після короткої паузи система автоматично відновлює роботу.



Таким чином реалізована система підрахунку балів відповідає правилам гри, легко налаштовується та готова до підключення користувальського інтерфейсу і подальшої збірки проекту для Android.

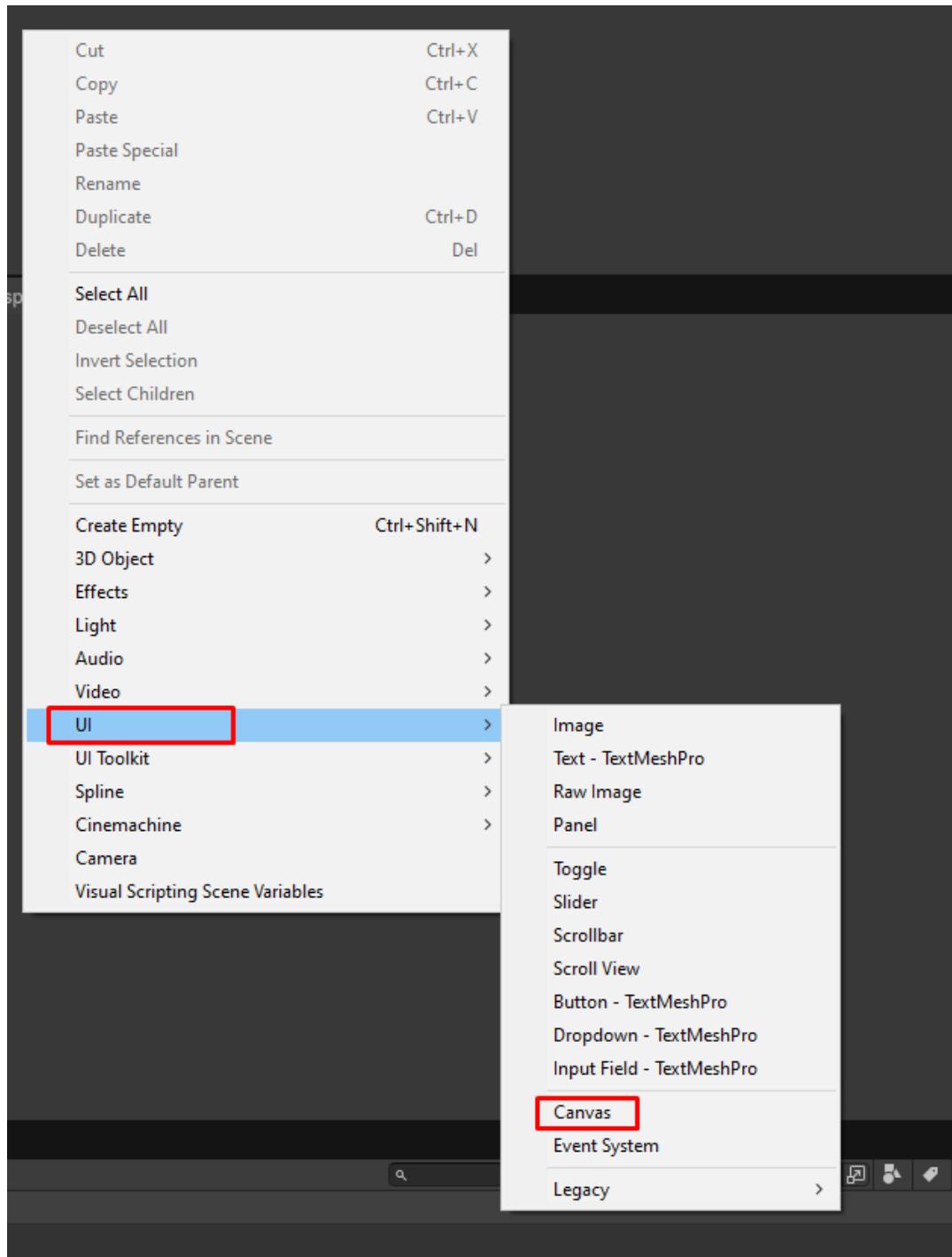
2.14 Створення стартового екрану гри

Стартовий екран є першим елементом взаємодії гравця з грою. Він дозволяє ознайомитися з персонажем, задати початкові параметри гри та усвідомлено розпочати проходження тунелю. На цьому етапі створюється стартовий інтерфейс, який містить візуальне зображення персонажа, кнопку початку гри та елемент керування швидкістю руху.

Для спрощення структури проекту стартовий екран реалізується у вигляді UI-панелі, яка відображається поверх сцени та вимикається після натискання кнопки Start. Всі екрани гри розміщуються в одній сцені, що

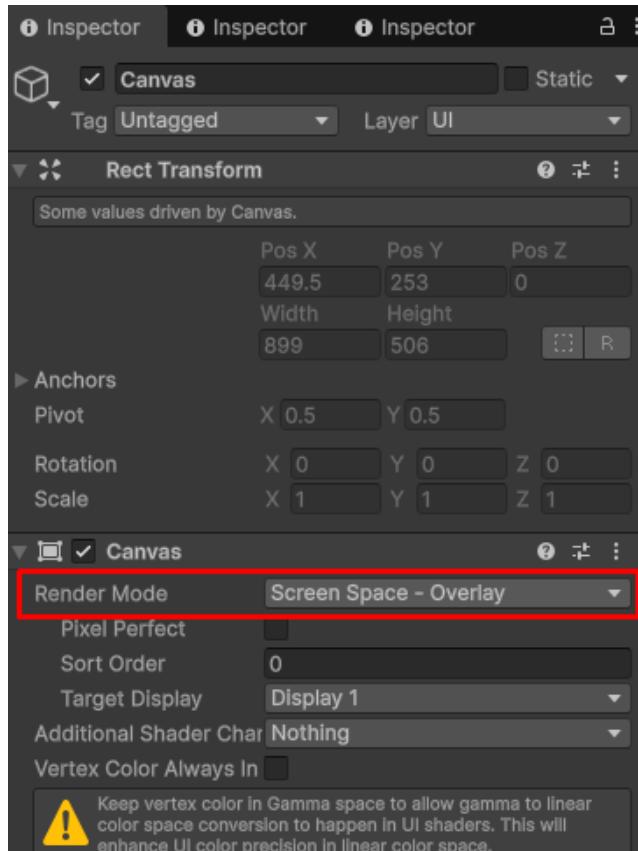
забезпечує стабільну роботу на мобільних пристроях та спрощує логіку керування.

У вікні Hierarchy створюється об'єкт Canvas за допомогою команди Create → UI → Canvas.



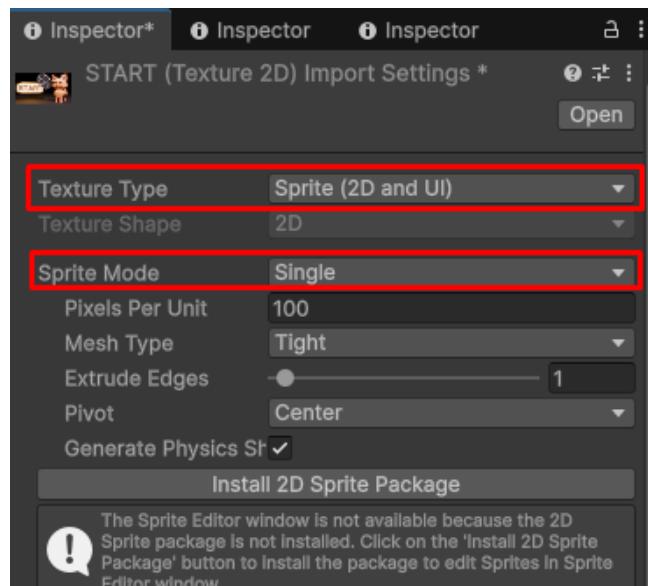
Для Canvas автоматично додається об'єкт EventSystem, який відповідає за обробку натискань. У налаштуваннях Canvas у полі Render Mode

встановлюється значення Screen Space – Overlay, що дозволяє відображати інтерфейс поверх усієї сцени.

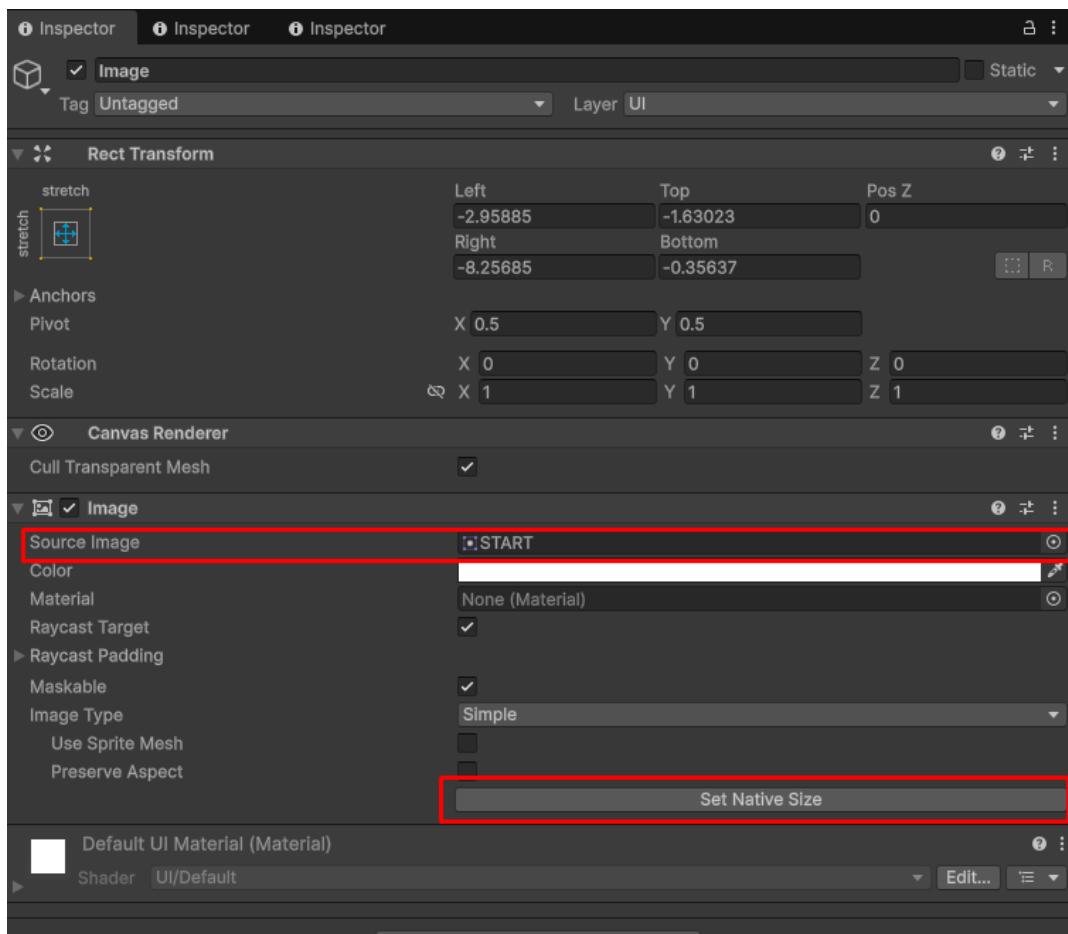


Всередині Canvas створюється порожній UI-об'єкт Panel, який отримує назву StartPanel. Цей об'єкт буде відповідати за стартовий екран гри. Його розміри встановлюються таким чином, щоб він займав увесь екран. За потреби фону можна задати напівпрозорий колір або залишити його повністю прозорим.

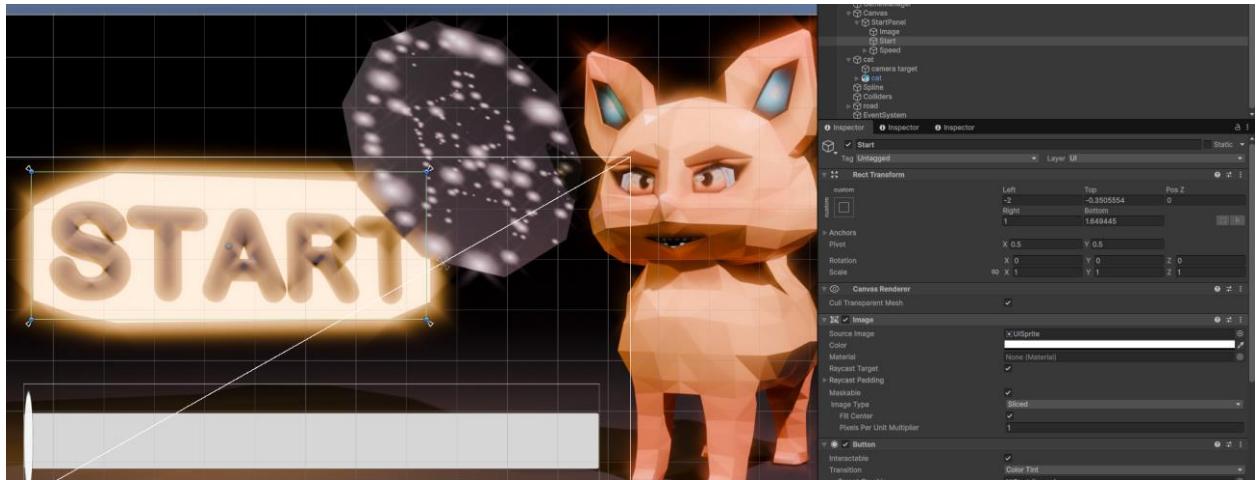
Для відображення персонажа на стартовому екрані використовується заздалегідь підготовлений рендер або зображення. Усередині об'єкта StartPanel створюється UI-елемент Image. У властивостях Image у полі Source Image обирається текстура з рендером персонажа.



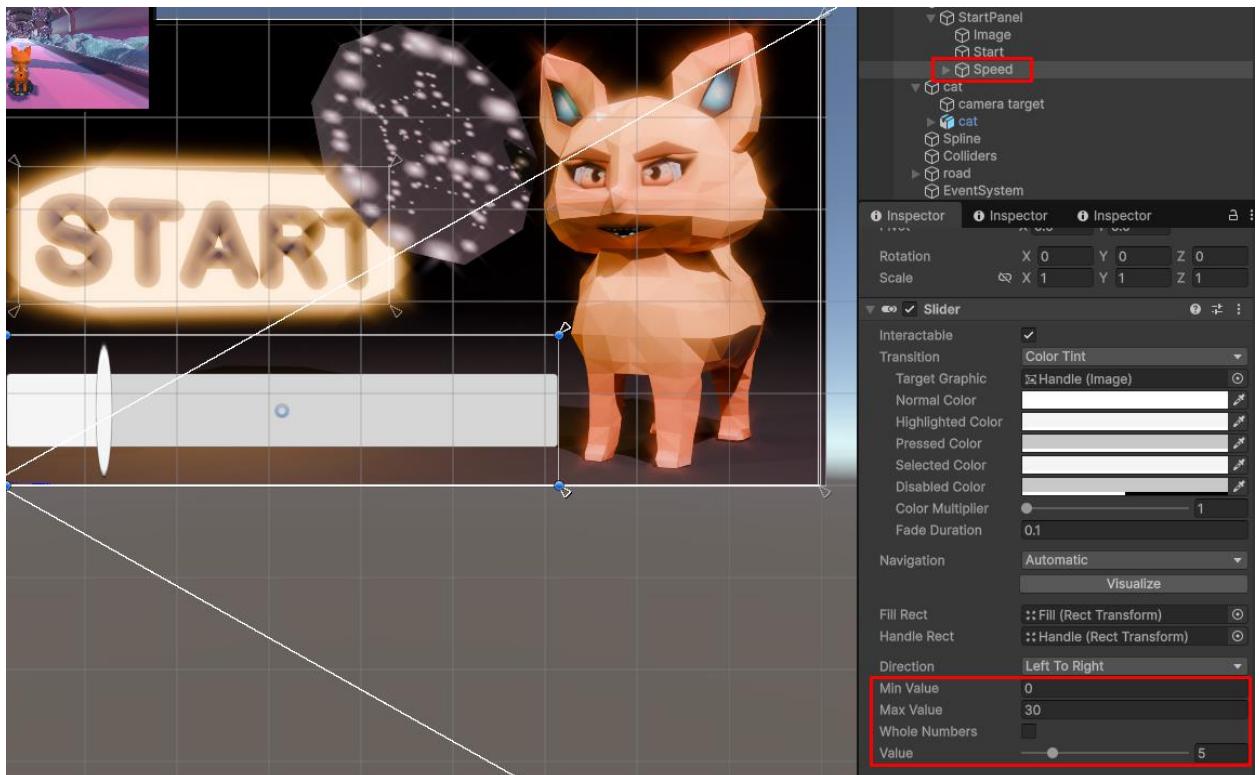
Зображення масштабується та розташовується таким чином, щоб персонаж був добре помітний і не перекривав елементи керування. Цей елемент не має ігрової логіки та виконує виключно візуальну роль, створюючи перше враження від гри.



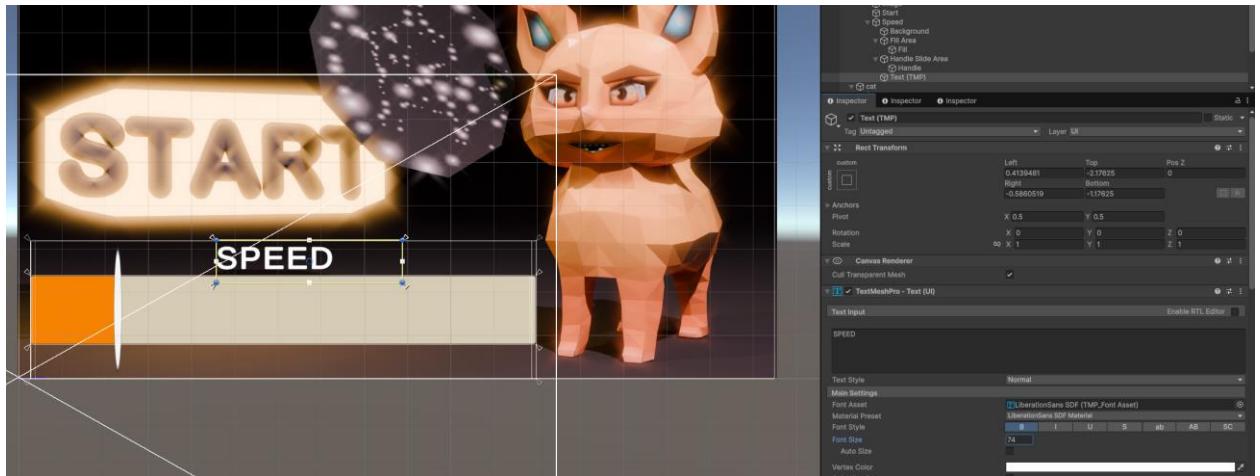
Для розміщення кнопки та налаштування швидкості всередині StartPanel створюється кнопка Start за допомогою команди Create → UI → Button. Текст кнопки змінюється на «Start».



Також додається елемент Slider, який буде використовуватися для вибору швидкості руху персонажа.



Для зручності над Slider можна додати текстове пояснення, наприклад «Швидкість».



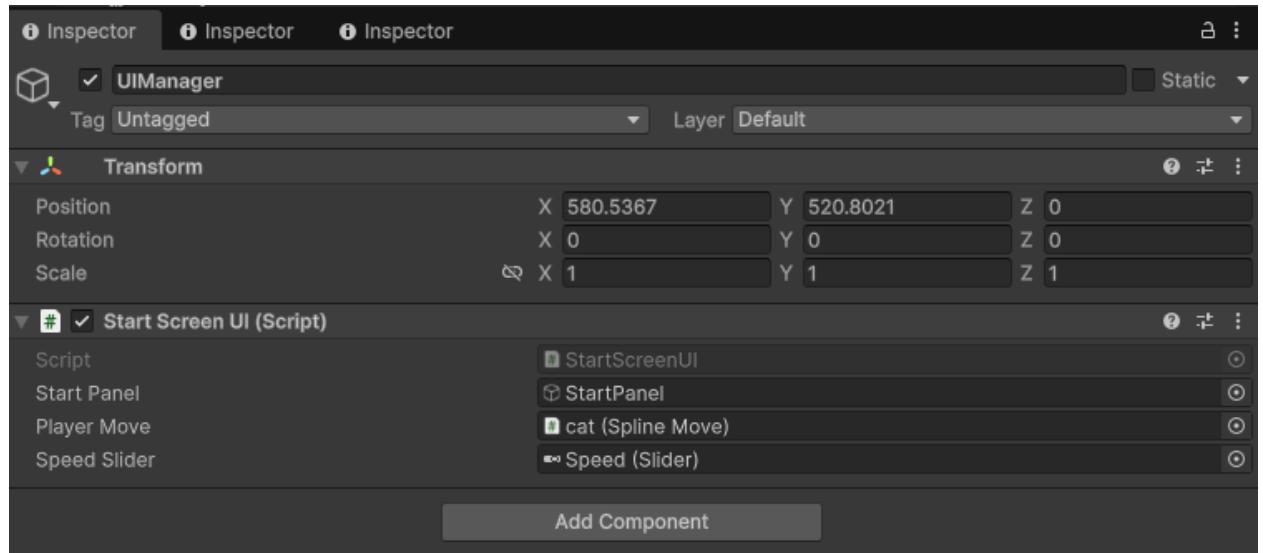
Для керування логікою стартового екрану у папці Scripts створюється новий C#-скрипт з назвою StartScreenUI. Цей скрипт відповідатиме за запуск гри та передачу вибраної швидкості у систему руху персонажа.

```
using UnityEngine;
public class StartScreenUI : MonoBehaviour
{
    [SerializeField] private GameObject startPanel;
    [SerializeField] private SplineMove playerMove;
    [SerializeField] private UnityEngine.UI.Slider speedSlider;

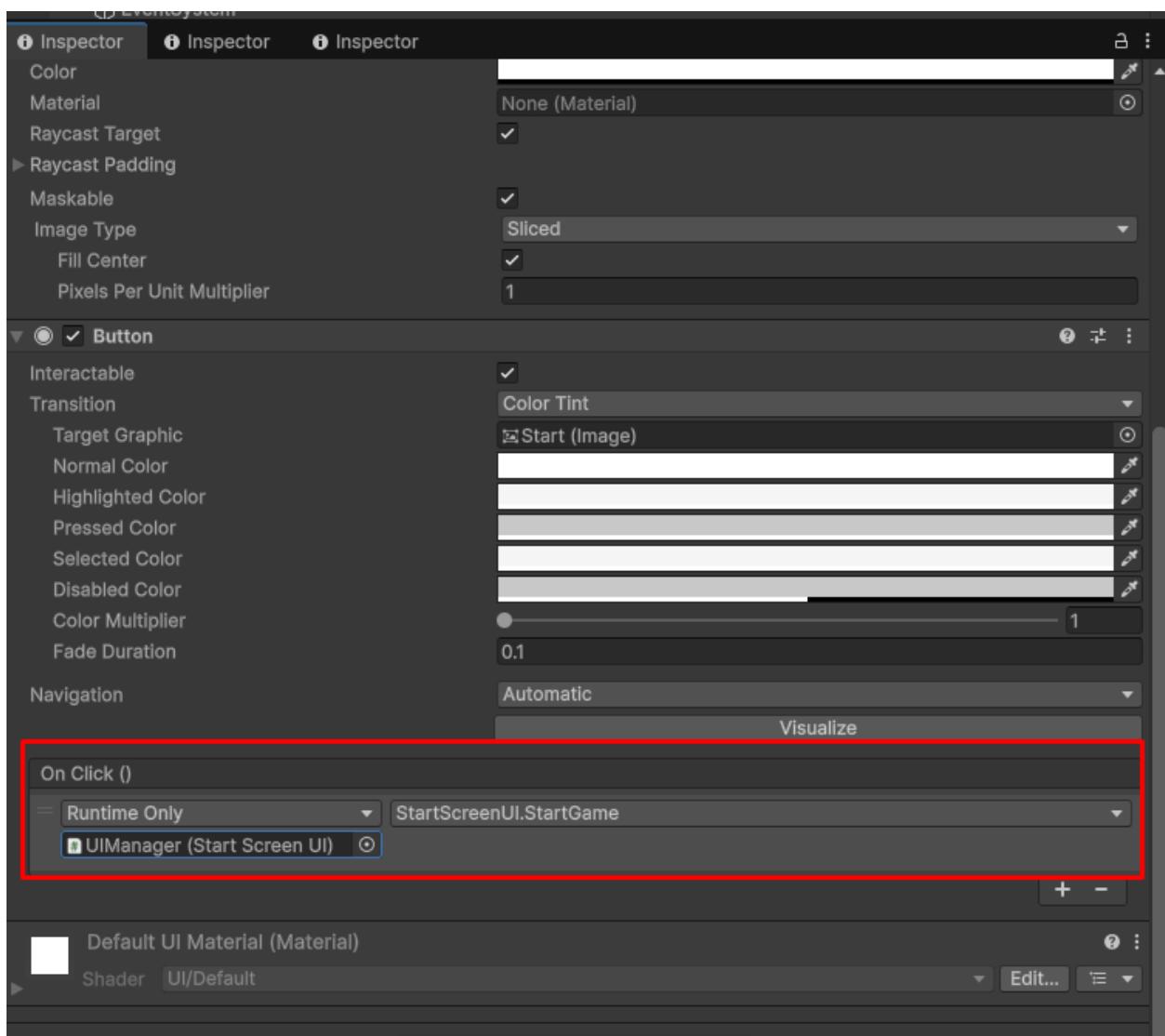
    void Start()
    {
        startPanel.SetActive(true);
        Time.timeScale = 0f;
    }
    public void StartGame()
    {
        playerMove.SetSpeed(speedSlider.value);
        startPanel.SetActive(false);
        Time.timeScale = 1f;
    }
}
```

У цьому коді об'єкт startPanel відповідає за стартовий екран. Під час запуску гри час призупиняється, що не дозволяє персонажу почати рух до натискання кнопки. Після натискання кнопки Start вибране значення швидкості передається у скрипт руху персонажа, стартовий екран вимикається, а ігровий процес починається.

Скрипт StartScreenUI додається до будь-якого зручного об'єкта сцени, наприклад до порожнього об'єкта UIManager. У вікні Inspector у відповідні поля перетягуються об'єкти StartPanel, Slider швидкості та ігровий персонаж зі скриптом руху.



Для кнопки Start у компоненті Button у розділі On Click додається подія, яка викликає метод StartGame зі скрипта StartScreenUI.



Після завершення налаштувань при запуску гри на екрані відображається стартовий екран з рендером персонажа, панеллю керування та кнопкою Start. Гравець може обрати бажану швидкість руху, після чого натиснути кнопку і розпочати проходження тунелю. Стартовий екран зникає, а ігровий процес починається з обраними параметрами.

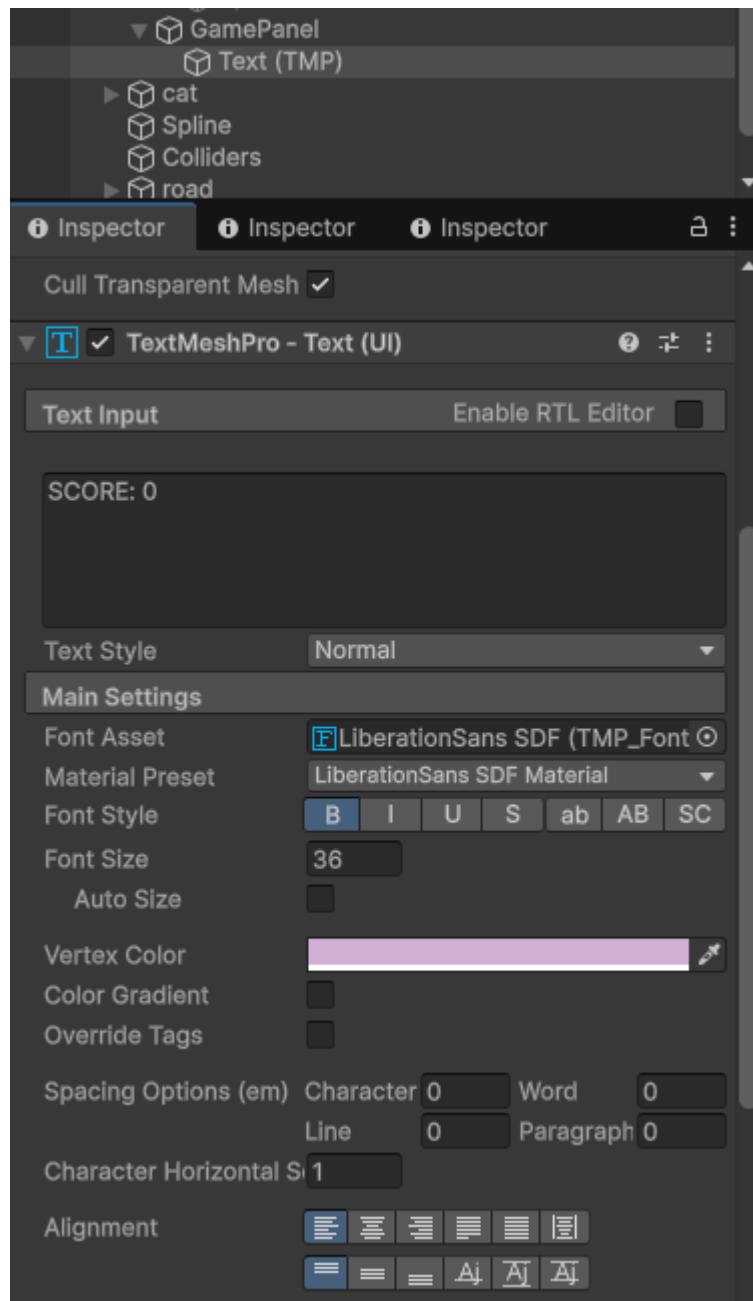
2.15 Відображення рахунку під час гри

Під час ігрового процесу гравець повинен бачити поточний рахунок, який змінюється залежно від пройденої дистанції та зіткнень зі стінками тунелю. Для цього в грі реалізується окремий елемент інтерфейсу, який постійно оновлює значення балів та відображається поверх ігрової сцени.

Відображення рахунку здійснюється за допомогою UI-елемента Text, який розміщується в Canvas і активний лише під час гри. Сам підрахунок балів виконується в окремому скрипті ScoreManager, який не залежить від інтерфейсу. Таким чином забезпечується чітке розділення логіки гри та її візуального представлення.

У Canvas створюється новий UI-об'єкт Panel з назвою GamePanel. Ця панель буде активною лише під час гри. Усередині GamePanel створюється текстовий елемент (Text або TextMeshPro), який отримує назву ScoreText. Він розміщується у зручному місці екрана, наприклад у верхньому лівому або правому куті, та відображає поточну кількість балів.

Текст за замовчуванням можна встановити як «Score: 0». Розмір шрифту та колір підбираються так, щоб інформація була добре помітною на фоні сцени.

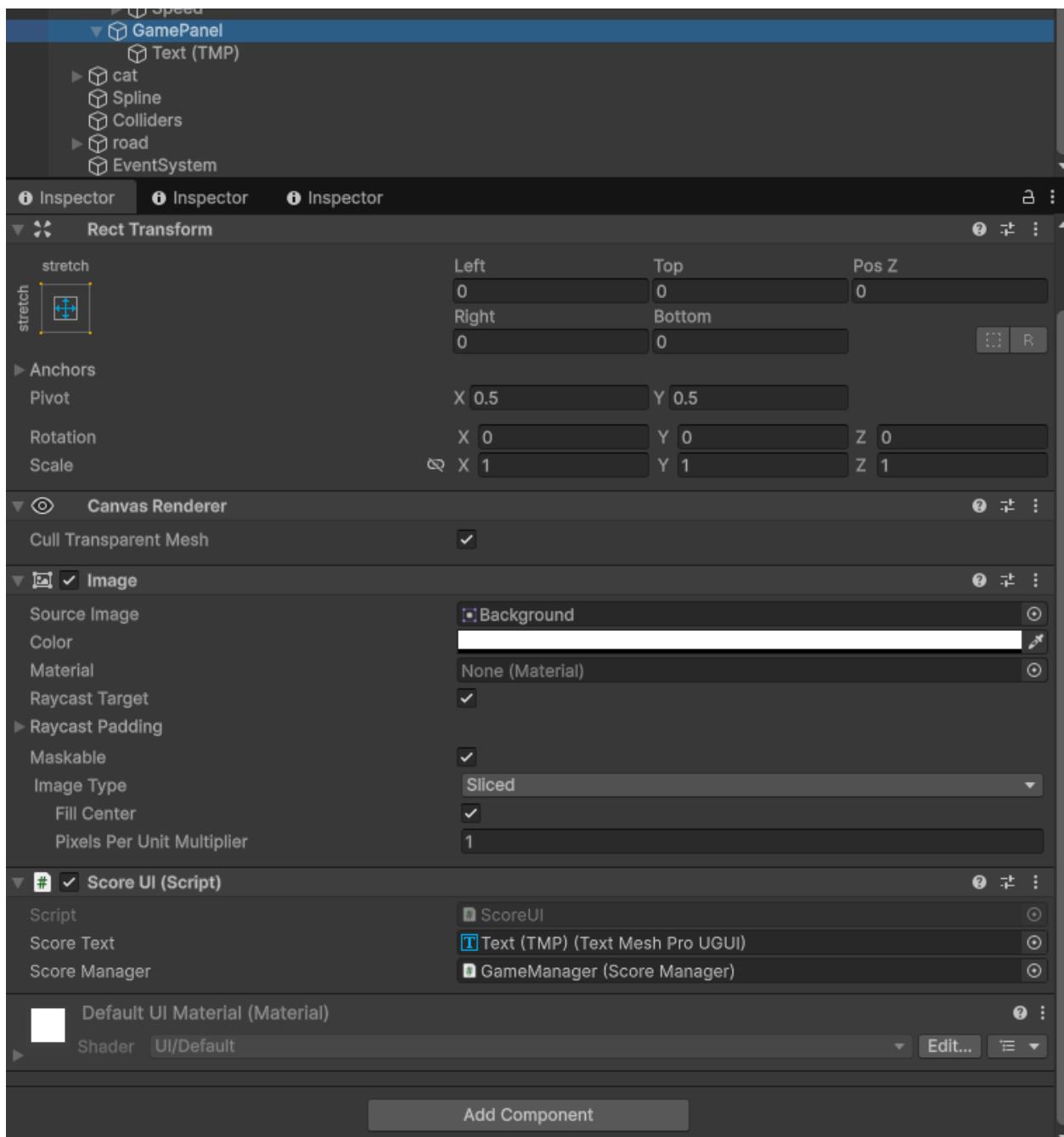


Для зв’язку між системою підрахунку балів та інтерфейсом створюється новий C#-скрипт з назвою ScoreUI. Цей скрипт відповідає лише за відображення значення рахунку на екрані.

```
using UnityEngine;
using UnityEngine.UI;
public class ScoreUI : MonoBehaviour
{
    [SerializeField] private TMP_Text scoreText;
    [SerializeField] private ScoreManager scoreManager;
```

```
void Update()
{
    scoreText.text = "Score: " + scoreManager.score.ToString();
}
}
```

У цьому коді щокадрово зчитується поточне значення рахунку зі скрипта ScoreManager та оновлюється текстовий елемент. Скрипт ScoreUI додається до об'єкта GamePanel. У вікні Inspector в поле Score Text перетягується відповідний текстовий елемент, а в поле Score Manager — об'єкт сцени, на якому знаходиться компонент ScoreManager.



Після цього при запуску гри текстовий елемент починає автоматично оновлюватися та відображає актуальний рахунок гравця.

Підрахунок балів здійснюється автоматично залежно від пройденої дистанції вздовж тунелю. За кожну умовну дистанцію, значення якої задається в Inspector, гравцю нараховується один бал. У випадку зіткнення зі стінками тунелю з рахунку віднімається два бали. Оновлення інтерфейсу відбувається незалежно від того, яким чином змінюється рахунок, що робить систему універсальною та легкою для розширення.

Ігрова панель з відображенням рахунку приховується на момент запуску сцени. Після натискання кнопки Start стартовий екран вимикається, а ігровий інтерфейс активується програмно. Такий підхід дозволяє чітко контролювати етапи гри та уникнути некоректного відображення елементів інтерфейсу. Оновлений код скрипта StartScreenUI:

```
using UnityEngine;
using UnityEngine.UI;

public class StartScreenUI : MonoBehaviour
{
    [SerializeField] private GameObject startPanel;
    [SerializeField] private GameObject gamePanel;
    [SerializeField] private SplineMove playerMove;
    [SerializeField] private Slider speedSlider;

    void Start()
    {
        startPanel.SetActive(true);
        gamePanel.SetActive(false);
        Time.timeScale = 0f;
    }

    public void StartGame()
    {
        playerMove.SetSpeed(speedSlider.value);

        startPanel.SetActive(false);
        gamePanel.SetActive(true);

        Time.timeScale = 1f;
    }
}
```

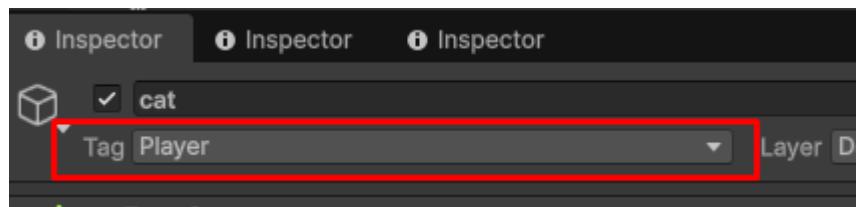
Після завершення даного етапу під час гри на екрані постійно відображається поточний рахунок. Гравець бачить, як значення змінюються в

реальному часі залежно від його дій, що підвищує залученість та зрозумільність ігрового процесу.

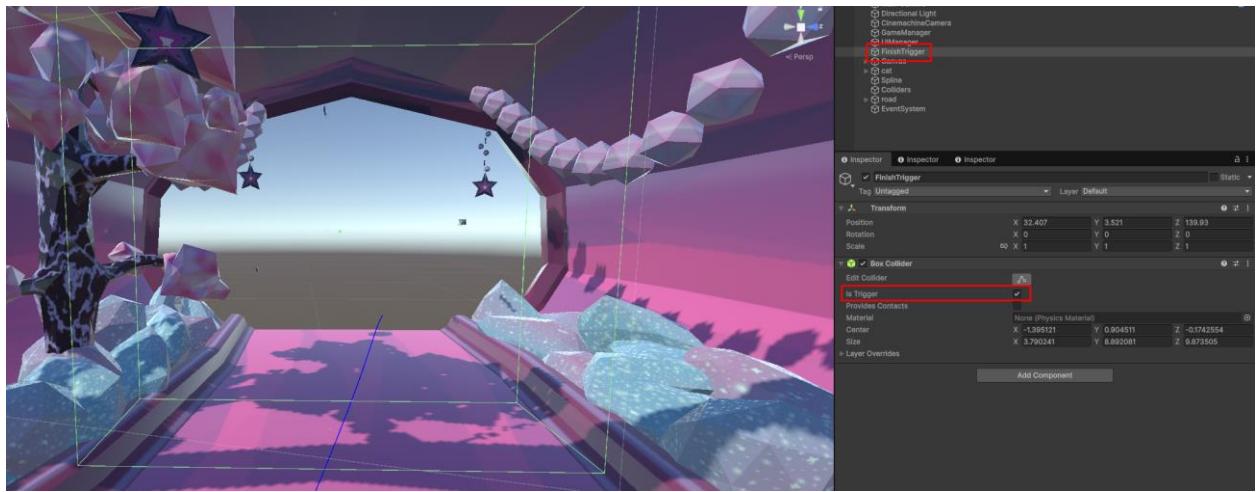
2.16 Завершення рівня та створення фінального екрану гри

Після проходження всього тунелю гра повинна коректно завершуватися та інформувати гравця про результат. Для цього реалізується фінальний екран, який з'являється після досягнення кінця маршруту, зупиняє рух персонажа та відображає підсумковий рахунок. Такий підхід дозволяє чітко відокремити ігровий процес від етапу підбиття підсумків та створює завершене враження від гри.

Завершення рівня визначається за допомогою спеціального тригерного колайдера, розміщеного в кінці тунелю. При вході персонажа в цю зону ігровий процес припиняється, а інтерфейс перемикається з ігрового екрану на фінальний.

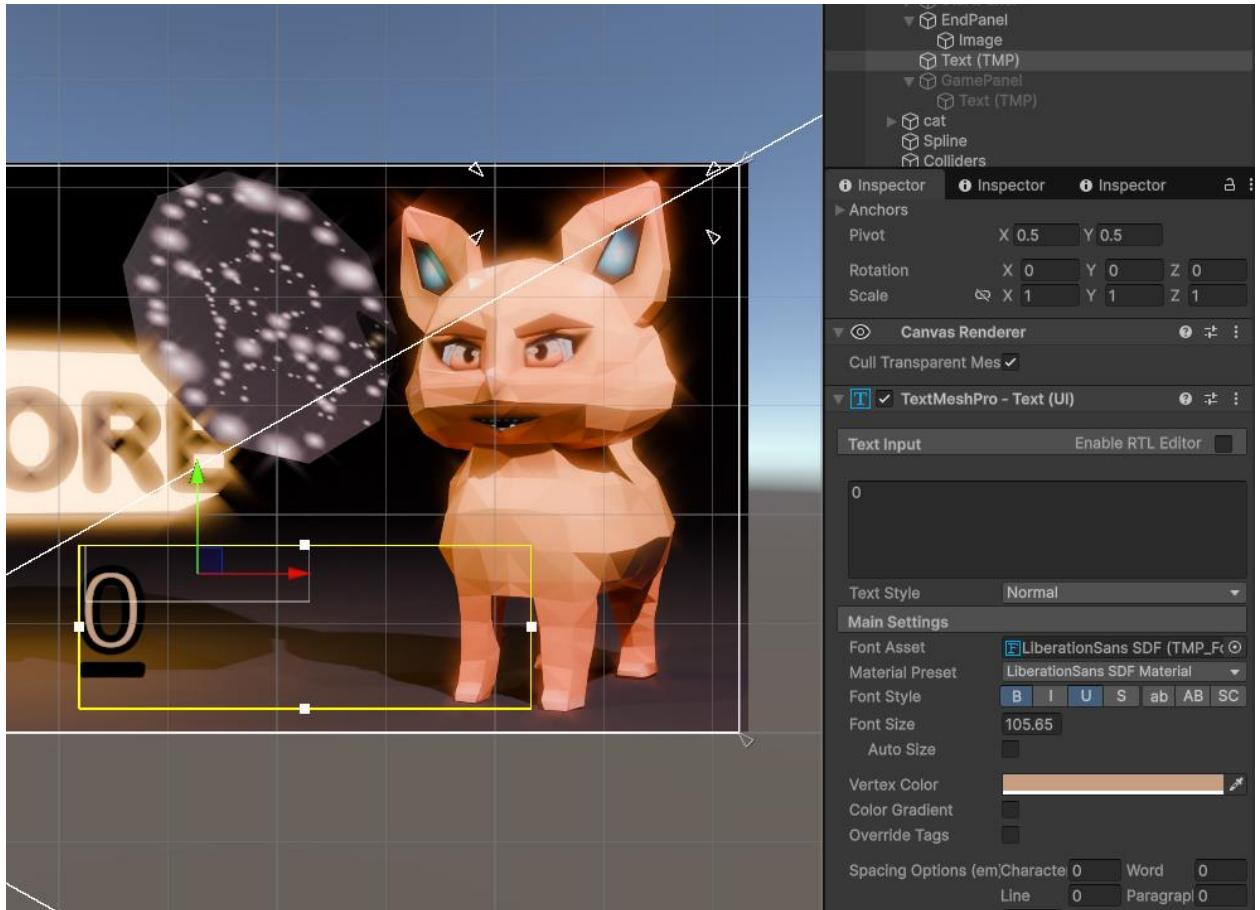


У сцені створюється порожній об'єкт, наприклад FinishTrigger, який розміщується в кінці сплайна або безпосередньо за останнім сегментом тунелю. До цього об'єкта додається Collider з увімкненим параметром Is Trigger. Для зручності та стабільності рекомендується використовувати Box Collider, розміри якого підбираються так, щоб персонаж гарантовано проходив через нього.



Фінішний об'єкт можна розмістити на окремому шарі, наприклад Finish, щоб уникнути конфліктів з іншими тригерами сцени.

У Canvas створюється нова UI-панель з назвою EndPanel. Вона за замовчуванням повинна бути вимкненою. Усередині EndPanel розміщується текстовий елемент, який відображає фінальний рахунок, наприклад «0».



EndPanel не бере участі в ігровому процесі та активується лише один раз — після завершення рівня.

Для обробки завершення рівня створюється новий C#-скрипт з назвою FinishHandler. Він відповідає за перемикання інтерфейсу, зупинку руху персонажа та передачу фінального рахунку у фінальний екран.

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class FinishHandler : MonoBehaviour
{
    [SerializeField] private GameObject gamePanel;
    [SerializeField] private GameObject endPanel;
    [SerializeField] private TMP_Text finalScoreText;
    [SerializeField] private ScoreManager scoreManager;
    [SerializeField] private SplineMove playerMove;

    private bool finished = false;

    private void Start()
    {
        endPanel.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (finished)
            return;

        if (other.CompareTag("Player"))
        {
            finished = true;
            EndGame();
        }
    }

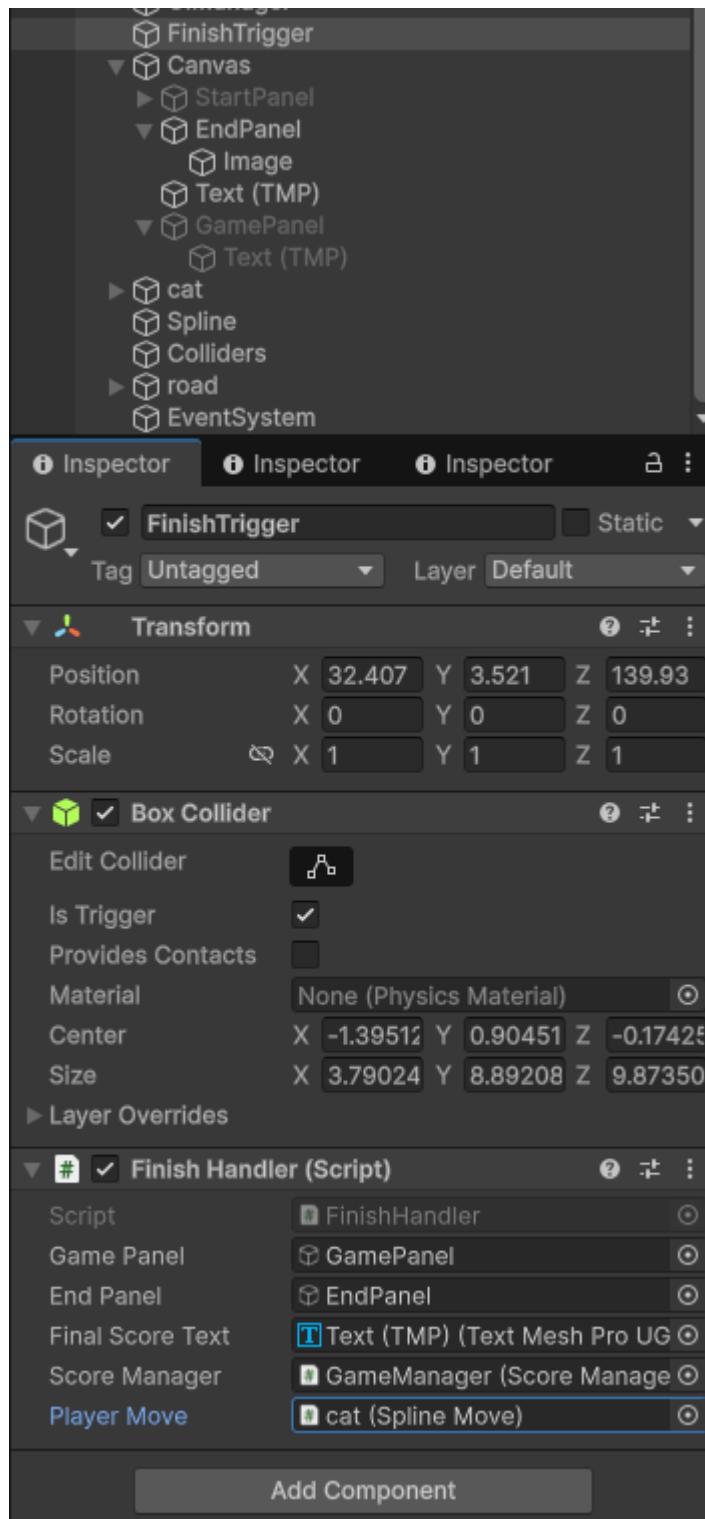
    private void EndGame()
    {
        playerMove.SetSpeed(0f);

        gamePanel.SetActive(false);
        endPanel.SetActive(true);

        finalScoreText.text = "" + scoreManager.score.ToString();
        Time.timeScale = 0f;
    }
}
```

У цьому коді перевіряється факт входу персонажа у фінішну зону. Після первого спрацювання рух персонажа зупиняється, ігровий інтерфейс приховується, а фінальний екран активується. Поточне значення рахунку передається з компонента ScoreManager та виводиться у текстовому полі фінального екрану.

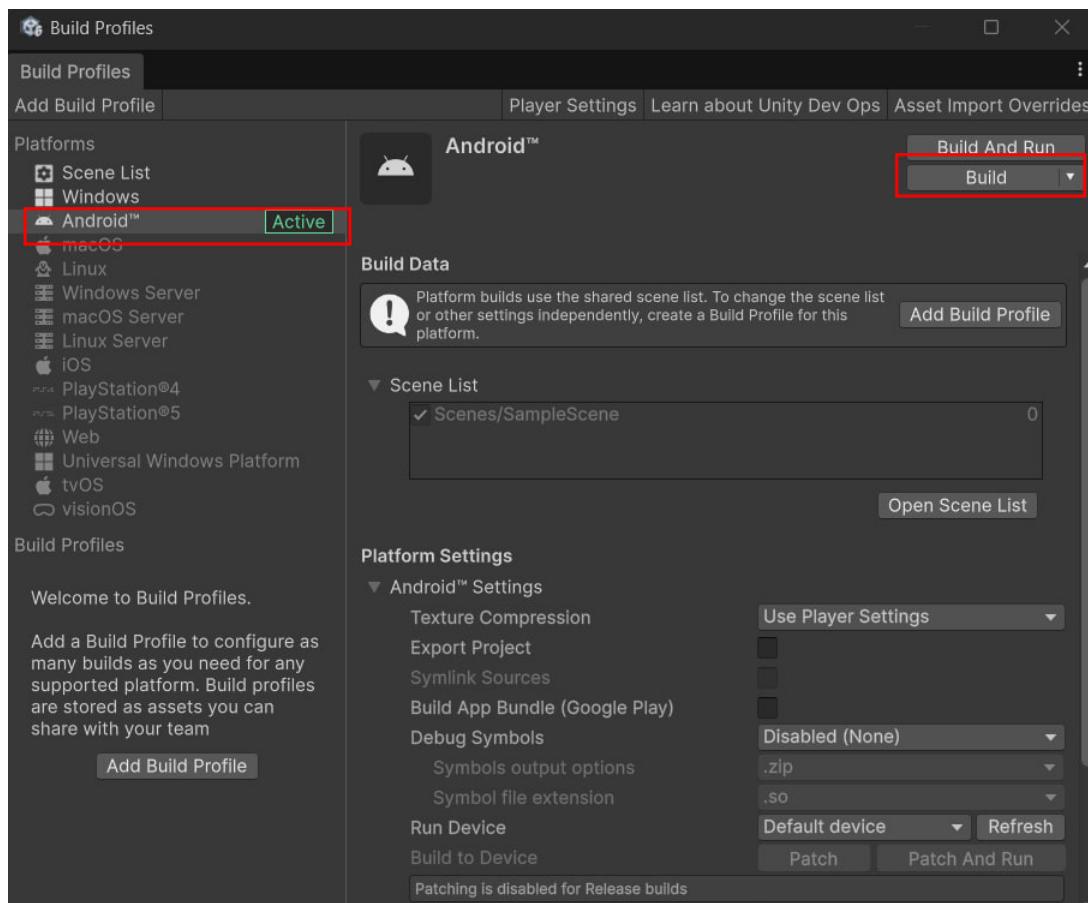
Скрипт FinishHandler додається до об'єкта FinishTrigger. У вікні Inspector в відповідні поля перетягуються об'єкти GamePanel, EndPanel, текстове поле фінального рахунку, а також об'єкти зі скриптами ScoreManager та SplineMove. Ігровий персонаж повинен мати тег Player, щоб коректно визначати завершення рівня.



Після проходження персонажем всього тунелю гра автоматично завершується. Рух припиняється, ігровий інтерфейс зникає, а на екрані з'являється фінальний екран із підсумковим рахунком. Таким чином забезпечується логічне завершення рівня та зрозумілий для гравця результат його проходження.

2.17 Побудова арк-модуля для завантаження на Android-смартфон

Для початку процесу Побудова арк-модуля для завантаження на Android-смартфон необхідно перейти за пунктом меню File-> Build Settings та натиснути кнопку «Build» (дивіться рисунок).



Система запропонує вказати назву файлу та його розташування у файловій системі на комп’ютері, після чого впродовж 10-20 хвилин буде побудовано файл. Цей файл треба перенести до смартфону.

2.18 Рекомендації щодо впровадження методичних вказівок у навчальний процес

З 2023 року Одеська політехніка у складі консорціуму з університетів України, Австрії, Чехії, Словаччини та Ірландії приймає участь у міжнародному проекті Erasmus+ K2 NEXT-«Цифрова трансформація для підтримки робочої сили наступного покоління» [1].

Одним із результатів проекту є навчальне середовище NEXT-Study як сховище різноманітних навчальних ресурсів, які студенти зможуть використовувати для підвищення знань у певній галузі, охоплюючи різні теми цифрових навичок і цифрової грамотності. Аналіз змісту навчальних курсів визначив наступні курси, які пов'язані із напрямком комп'ютерної ігровізації фізичних вправ:

- навчальний курс «Responses to special education needs» («Виклики на спеціальні освітні потреби»), який включає розпізнавання рухів та інтерфейси введення, розглядає комп'ютерну ігровізацію для адаптації навчання, може бути базою для розробки ігрових вправ, з орієнтацією на доступність та інклюзію [2];
- навчальний курс «Ментальне здоров'я в умовах цифровізації», який у розділі «4 Практичні аспекти цифрового детоксу, методів перезавантаження, технологій для сприяння психічному здоров'ю» демонструє приклади комп'ютерної ігровізації балансуючих дошок [3].

ВИСНОВКИ

У ході виконання даної методичної роботи було розроблено повноцінну тривимірну гру для мобільної платформи Android із використанням ігрового рушія Unity. Реалізований проект демонструє повний цикл створення ігрового застосунку — від підготовки середовища розробки та налаштування сцени до реалізації ігрової логіки, користувацького інтерфейсу та завершення рівня.

У процесі роботи було створено систему руху персонажа вздовж криволінійного маршруту на основі сплайна з можливістю керування відхиленням за допомогою нахилу мобільного пристрою. Такий підхід дозволяє забезпечити інтуїтивне управління, характерне для сучасних мобільних ігор, а також демонструє практичне застосування фізики Rigidbody у поєднанні з математичними моделями траєкторій.

Особливу увагу приділено організації ігрового простору та взаємодії з колайдерами. Було реалізовано систему зіткнень зі стінками тунелю з коректним визначенням штрафів, а також розділення колайдерів за призначенням для уникнення помилкових спрацювань. Це дозволило наочно продемонструвати принципи роботи тригерів та шарів у Unity.

У грі реалізовано гнучку систему підрахунку балів, яка базується на пройденій дистанції та налаштовується через Inspector без необхідності змінювати програмний код. Такий підхід відповідає принципам масштабованості та повторного використання компонентів. Додатково реалізовано штрафи за зіткнення, що підвищують складність та динамічність ігрового процесу.

Користувацький інтерфейс гри складається зі стартового екрану з можливістю вибору швидкості руху, ігрового екрану з відображенням поточного рахунку та фінального екрану з підсумковим результатом проходження рівня. Перемикання між екранами реалізовано в межах однієї сцени, що забезпечує стабільну роботу застосунку та спрощує логіку керування станами гри.

У результаті виконання методичної роботи було отримано структурований приклад створення 3D гри для мобільної платформи, який може бути використаний як навчальний матеріал для початківців. Запропонований підхід дозволяє легко розширювати функціональність гри, додавати нові рівні, змінювати складність та вдосконалювати візуальну складову без суттєвих змін базової архітектури проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Національний університет «Одеська політехніка». Цифрова трансформація для підтримки робочої сили наступного покоління / Національний університет «Одеська політехніка», Міжнародні грантові проекти. URL: <https://op.edu.ua/international/projects/next> (дата звернення: 14.01.2026).
2. Blazhko, Oleksandr. 304 – Responses to special education needs / Techpedia – освітній курс. URL: <https://techpedia.eu/topic/233> (дата звернення: 14.01.2026).
3. Вовкочин, Людмила. 301 – Ментальне здоров'я в умовах цифровізації / Techpedia – освітній курс. URL: <https://techpedia.eu/topic/289> (дата звернення: 14.01.2026).