ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільної 3D гри казуального жанру зі зміною фізичних властивостей об’єкта»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ігор ТИХОНЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Ігор ТИХОНЧЕНКО

Керівник: Данило КОВАЛЕНКО
доктор філософії (PhD)

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Тихонченку Ігорю Олександровичу

1. Тема кваліфікаційної роботи: «Розробка мобільної 3D гри казуального жанру зі зміною фізичних властивостей об'єкта»
керівник кваліфікаційної роботи філософії (PhD) Данило КОВАЛЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку мобільних 3D ігор, фізичне моделювання в ігрових застосунках, принципи побудови ігрових механік, документація ігрового рушія Unity, засоби розробки мобільних застосунків, технології створення 3D-моделей та організації системи збереження даних у мобільних іграх.

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

1. Аналіз існуючих аналогів мобільних 3D ігор казуального жанру зі зміною фізичних властивостей об'єкта та визначення вимог до мобільної гри.
2. Проектування архітектури мобільної 3D гри та структури її основних модулів.

3. Реалізація ігрових механік, системи рівнів, користувацького інтерфейсу та системи збереження даних мобільної 3D гри.
4. Тестування мобільної 3D гри та аналіз результатів тестування.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Концепт гри
3. Вимоги до програмного забезпечення.
4. Програмні засоби реалізації.
5. Діаграма діяльності гри.
6. Діаграма класів.
7. Екранні форми
8. Відео демонстрація гри.
9. Тестування
10. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Аналіз ПЗ для розробки мобільної 3D гри	10.03-18.03.2026	
4	Проектування архітектури мобільної 3D гри	19.03-11.04.2026	
5	Розробка користувацького інтерфейсу	12.04-17.04.2026	
6	Реалізація ігрових механік, системи рівнів та системи збереження даних	18.04-29.04.2026	
7	Тестування та оптимізація гри	30.04-02.05.2026	
8	Оформлення роботи: вступ, висновки, реферат	03.05-08.05.2026	
9	Розробка демонстраційних матеріалів	09.05-10.05.2026	
10	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Ігор ТИХОНЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Данило КОВАЛЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 43 стор., 3 табл., 22 рис., 21 джерел.

Мета роботи – підвищення інтерактивності мобільної 3D гри казуального жанру, шляхом впровадження динамічної зміни фізичних властивостей об’єкта.

Об’єкт дослідження – процес створення мобільної 3D-гри зі зміною фізичних властивостей об’єкта.

Предмет дослідження – методи реалізації ігрових механік зі зміною фізичних властивостей об’єкта мовою програмування C#, та оптимізації гри для платформи Android.

Короткий зміст роботи: В роботі досліджено особливості мобільних 3D ігор казуального жанру та розглянуто застосування фізичного моделювання у сучасних ігрових застосунках. Виконано аналіз існуючих ігор із фізичними механіками, а також програмних засобів, що використовуються для розробки мобільних ігор, зокрема Unity, Blender, Firebase, Figma та Audacity. Розроблено архітектуру мобільної 3D гри та програмно реалізовані основні функціональні можливості, зокрема: систему керування кулею, фізичні режими об’єкта, систему рівнів, користувацький інтерфейс та систему збереження прогресу гравця. Проведено функціональне тестування гри методом «чорної скриньки». В роботі використано ігровий рушій Unity, мову програмування C#, Blender для створення 3D-моделей, Firebase для збереження даних користувача, Figma для проектування інтерфейсу та Audacity для обробки аудіо.

Сферою використання застосунку є організація самостійної роботи студентів в процесі вивчення дисципліни «Робототехніка».

КЛЮЧОВІ СЛОВА: МОБІЛЬНА КАЗУАЛЬНА ГРА, UNITY, ФІЗИЧНІ ВЛАСТИВОСТІ, C#, ІГРОВІ МЕХАНІКИ, FIREBASE, BLENDER.

ЗМІСТ

ВСТУП

Актуальність дослідження

Сфера мобільних ігор є одним із найпопулярніших напрямів у сучасній індустрії програмного забезпечення. Щороку збільшується кількість користувачів, які віддають перевагу іграм на смартфонах як простому засобу відпочинку, швидкого занурення у розважальний процес та можливості проводити короткі ігрові сесії без прив'язки до спеціалізованого обладнання. Особливе місце на ринку займають казуальні ігри, що характеризуються простотою освоєння, інтуїтивним керуванням та різноманітністю механік, які не потребують складної підготовки.

Одним із перспективних напрямів розробки казуальних ігор є використання фізичних властивостей об'єктів як ключового елементу геймплею. Зміна маси, швидкості, сили тертя та інших параметрів дозволяє створювати унікальні сценарії проходження, відкривати нестандартні способи взаємодії з оточенням та підвищувати залученість користувачів. Попри існування багатьох мобільних 3D ігор, ще недостатньо досліджено застосування фізичних механік як основи геймплею казуальних проектів. Це обумовлює потребу у проведенні наукового дослідження з розробки таких ігор, що є актуальним для розвитку індустрії мобільного геймінгу та створення конкурентоспроможних продуктів в Україні.

Ступінь наукової розробки

Зарубіжні та вітчизняні дослідження з проблеми розробки мобільних ігор з фізичними механіками охоплюють здебільшого окремі аспекти: використання фізичних рушіїв у 2D та 3D проектах, оптимізацію геймплею, інтеграцію простих фізичних моделей. Однак систематичний підхід до побудови казуальної 3D гри, де зміна фізичних властивостей об'єктів є основою геймплею, ще не отримав достатнього висвітлення. Відсутня узагальнена методика проектування архітектури таких ігор із врахуванням масштабованості та варіативності проходження рівнів.

Практичне значення одержаних результатів

Результати дослідження можуть бути використані для:

розробки мобільних 3D казуальних ігор із фізичними механіками; фізичного моделювання, проектування архітектури гри та формування механік, що забезпечують варіативність проходження рівнів та масштабованість проекту.

Мета дослідження

Метою дослідження є підвищення інтерактивності мобільної 3D гри казуального жанру за допомогою ігрового рушія Unity, шляхом впровадження динамічної зміни фізичних властивостей об’єкта.

Результати дослідження

У процесі роботи планується створення концептуальної моделі гри та демонстраційного прототипу, що дозволить оцінити ефективність застосування фізичних механік у грі. Для цього використовується ігровий рушій Unity, який надає можливість моделювати фізичні властивості об’єкта та його взаємодію з оточенням у 3D-просторі. Розробка передбачає побудову логіки ігрового процесу, оптимізацію взаємодії об’єкта з перешкодами, організацію структури рівнів та створення інтерфейсу користувача з урахуванням масштабованості проекту.

Висновки та перспективи

Розробка мобільної 3D гри казуального жанру передбачає застосування зміни фізичних властивостей об’єкта для створення різноманітних сценаріїв проходження рівнів. Концепція гри спроектована з урахуванням масштабованості, що дозволяє додавати нові механіки та рівні без значної переробки коду. Перспективи розвитку гри включають впровадження нових рівнів, фізичних режимів, ігрових механік, системи досягнень, а також адаптацію гри під різні мобільні платформи, що сприятиме розширенню функціональності та підтриманню зацікавленості користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сутність та особливості мобільних ігор казуального жанру

На сьогоднішній день мобільні ігри є одним з найпопулярніших напрямів в розробці ігор. Головна особливість ігор на мобільні пристрої — це сенсорне керування та необхідність оптимізації під їх апаратні вимоги. На відміну від ПК чи консолей, мобільні пристрої мають слабші процесори та менший запас оперативної пам’яті. Тому розробнику завжди потрібно контролювати навантаження на залізо під час створення гри.

Серед усіх жанрів мобільних ігор найбільшу аудиторію мають казуальні ігри. Через те, що ігри такого жанру мають низький поріг входження, вони охоплюють майже всі вікові категорії. Гравцю не потрібно вивчати складні механіки або проходити довгі туторіали, тому що там вже все одразу зрозуміло.

Можна виділити основні ознаки казуальних ігор:

– короткі ігрові сесії;

– плавне та поступове зростання складності рівнів;

– простота геймплею;

– орієнтація на людей будь-якого віку та рівня ігрового досвіду.

Основою будь якої гри є її геймплей, тобто те, як гравець взаємодіє з грою. Якщо управління створено незручно, а ігровий процес нудний то, мало ймовірно, що хтось буде грати в це.

Зараз навіть прості мобільні ігри намагаються робити в 3D, тому що об’ємні моделі та гарна взаємодія зі світлом робить гру більш привабливою. Але, постійні розрахунки реалістичної фізики та гарної графіки сильно навантажують мобільні пристрої. Тому при розробці гри потрібно це враховувати та шукати баланс щоб гра виглядала привабливо і при цьому видавала стабільний FPS.

Щоб зробити геймплей цікавішим розробники додають в свої ігри якісь унікальні механіки. За допомогою цього вони виділяються на ринку і приваблюють більшу аудиторію, ніж ігри з простим ігровим процесом.

1.2 Використання фізичного моделювання у 3D іграх

Завдяки фізиці в тривимірних іграх гравець може легко передбачити поведінку будь-якого об’єкта, адже всі взаємодії симулюють реальний світ або є максимально наближеними до нього. Подібна симуляція руху, чітка обробка зіткнень та врахування сили тяжіння роблять гру живою, причому налаштування таких механік має особливе значення саме у 3D-проектах, оскільки весь ігровий процес відбувається у повноцінному об'ємному просторі за кількома напрямками одночасно.

За розрахунок усіх переміщень, швидкостей та взаємодій у грі відповідає фізичний рушій. Він автоматично обчислює поведінку об’єктів на сцені на основі закладених математичних законів. До ключових функцій такого рушія належать:

– симуляція кінематики: яка відповідає за прорахунок траєкторії та швидкості руху;

– обчислення динаміки: враховує масу тіл і прикладені до них сили;

– обробка колізій: яка фіксує самі моменти торкання чи ударів предметів;

– моделювання параметрів матеріалів: в основному визначення властивостей тертя, пружності або ковзання на поверхні.

Наразі є кілька популярних готових рішень, серед яких можна виділити PhysX, Havok, а також вбудовані фізичні модулі в ігрові рушії Unreal Engine або CryEngine. Проте для розробки мобільних ігор лідером залишається платформа Unity, через те що рушій забезпечує високу оптимізацію під мобільні пристрої, пропонує зручні інструменти для налаштування фізичних властивостей та має детальну документацію та велику спільноту розробників, що значно полегшує процес створення гри.

У процесі розробки тривимірних ігор зазвичай комбінують різні типи фізичних моделей.

Кінематичні моделі:

Змінюють свої координати у просторі суто через код або заздалегідь створену анімацію. Вони повністю ігнорують зовнішні удари чи силу тяжіння, що є зручним для програмування рухомих платформ або ліфтів;

Динамічні моделі:

Навпаки повністю підпорядковані законам фізики, оскільки вони мають вагу, падають під дією гравітації, реалістично відскакують від перешкод та можуть штовхати інші предмети на сцені;

Системи колізій або колайдери:

Відповідають за те, щоб ігрові об’єкти не проходили крізь стіни чи підлогу, чітко обмежуючи їхнє переміщення в межах карти.

Зазвичай під час створення 3D-ігор комбінують різні типи фізичних моделей:

– кінематичні моделі: змінюють позицію об’єкта суто через анімацію або код, ігноруючи на поштовхи інших об’єктів, що є зручним для створення рухомих платформ або перешкод;

– динамічні моделі: повністю працюють за законами фізики, адже вони мають масу, відстакують і можуть зсувати інші об’єкти;

– системи колізій: відповідають за те, щоб ігрови об’єкти не проходили кріз стіни чи підлогу, обмежуючи їх рух.

Додати таку логіку в гру можна за допомогою вже вбудованого фізичного рушія, або написати власні алгоритми для конкретних механік, другий варіант краще тим, що це дозволяє значно підвищити продуктивність гри. Але найчастіше використовується комбінований підхід, коли рушій відповідає за прості процеси на кшталт гравітації, а унікальні механіки створюються вручну за допомогою коду.

Оскільки смартфони мають слабші процесори та обмежений обсяг оперативної пам’яті, фізику доводиться обмежувати. У мобільних іграх не можна

дозволити розраховувати об’єку сотні інших об’єктів одночасно, через те що кожен об’єкт оновлює свою позицію і прораховує кожен дотик об’єктів, що сильно навантажує мобільний пристрій. Тому розробники спрощують колайдери і обмежують кількість оновлень фізики на секунду.

Загалом є три типи фізики в іграх:

– реалістична фізика: максимально точне відтворення законів природи, що частіше використовується в симуляторах або AAA-іграх;

– аркадна або ігрова фізика: коли закони всесвіту свідомо спотворюють або спрощують заради того, щоб гра залишалася динамічною та веселою;

– змішана: компроміс між стабільною частотою кадрів та реалістичною картинкою.

Ідея побудови геймплею гри навколо механіки зміни фізичних властивостей об’єкта може бути гарним рішенням. Це додає можливість створювати різні варіанти рівнів, цікаві пастки або ділянки рівнів, і при цьому це не перевантажує гравця складними механіками, та залишає зацікавленість проходження рівнів.

1.3 Аналіз існуючих аналогів ігор із фізичними механіками

Для аналізу було обрано ігри, у яких фізичні механіки суттєво впливають на проходження рівнів. Такі приклади дозволяють оцінити різні підходи до інтеграції фізичних моделей у 3D казуальні ігри та визначити можливості для використання цих механік у власному проекті.

Rolling Sky — мобільна гра, у якій гравець керує кулькою, що рухається по трасі з перешкодами. Фізика руху об’єкта визначає траєкторію та спосіб проходження рівня. На рисунку 1.1. показано приклад ігрового процесу.

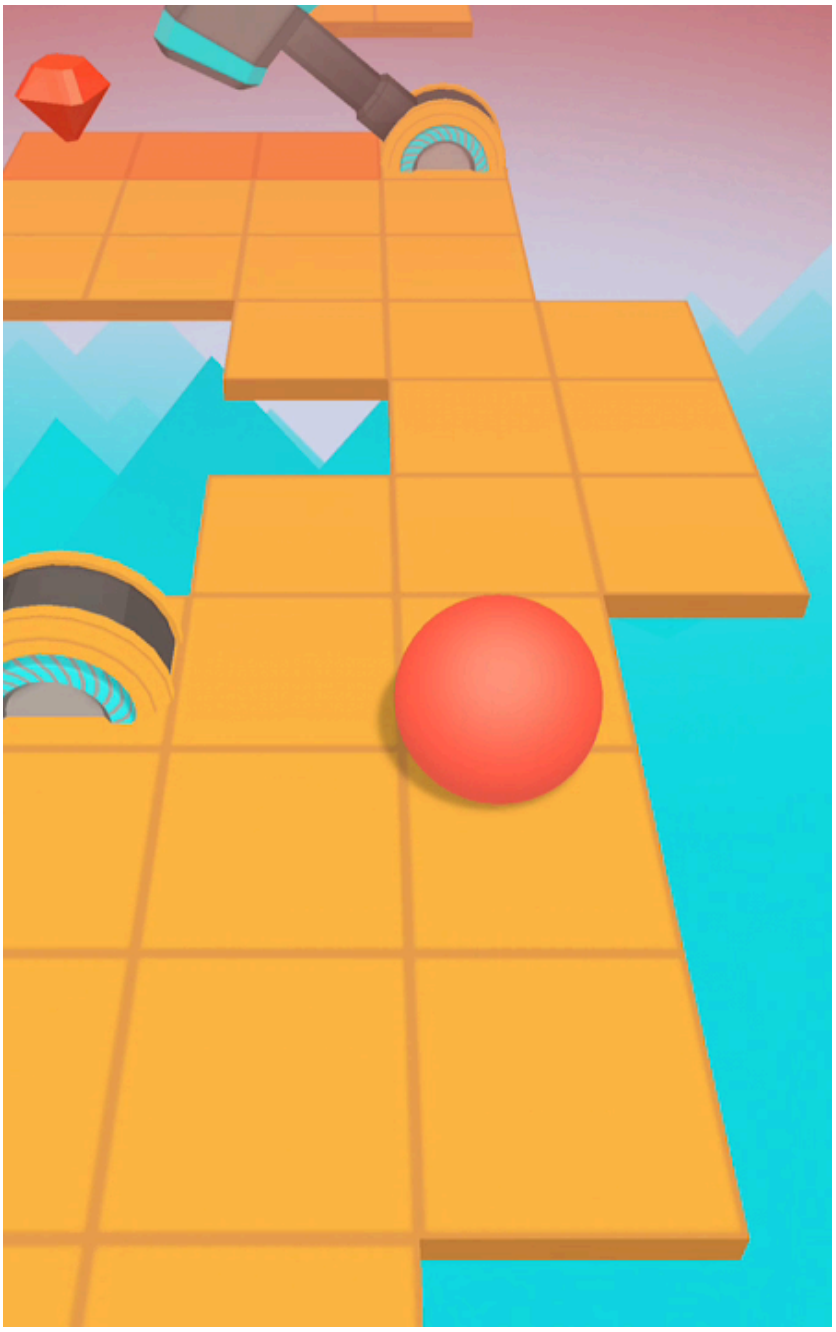


Рис. 1.1. Скріншот гри Rolling Sky

Going Balls — мобільна гра, де користувач керує кулькою, рухаючись по лабіринтах. Фізика руху та швидкість критично впливають на успіх, а різні рівні вимагають точного планування руху. Приклад геймплею наведено на рисунку 1.2.

Рис. 1.2. Скріншот гри Going Balls

Switchball — гра для ПК та мобільних платформ, у якій об’єкт змінює свої фізичні властивості для подолання складних рівнів. Різні властивості кульки створюють нові способи проходження рівнів та впливають на ігрову стратегію. Геймплей демонструється на рисунку 1.3.

Рис. 1.3. Скріншот гри Switchball

Super Monkey Ball — серія ігор для консолей та мобільних платформ, де гравець керує кулькою, у якій знаходиться персонаж. Фізика маси, швидкості та нахилу визначає результат проходження рівнів, створюючи динамічний та варіативний геймплей. Приклад ігрового процесу показано на рисунку 1.4.

Рис. 1.4. Скріншот гри Super Monkey Ball

Таблиця 1.1

Аналіз ігор

Назва гри	**Тип фізики**	**Основна механіка**	**Вплив фізики на геймплей**	**Особливості для мобільних платформ**	**Висновки для власного проекту**
Rolling Sky	Динамічна сфера	Керування кулькою	Середній – траєкторія визначається фізикою	Сенсорне керування, легка графіка	Механіка руху сфери центральна для геймплею
Ballance	Динаміка + гравітація	Балансування кульки	Високий – рух і баланс критичні	Сенсорне керування	Фізика руху і рівновага основою рівнів
Switchball	Динаміка + властивості об’єкта	Зміна властивостей кульки	Високий – властивості впливають на проходження	ПК/мобільна адаптація	Використання різних фізичних властивостей для геймплею
Super Monkey Ball	Кінематика + динаміка	Керування кулькою та збір предметів	Середній – маса і швидкість впливають на проходження	Консоль / мобільна адаптація	Контроль фізики для балансу рівнів

У таблиці 1.1 наведено аналіз ігор за типом фізики, основної механіки, впливом фізики на ігровий процес та особливостей управління на мобільних пристроях, що дозволяє виділити ключові підходи до інтеграції фізичних механік в іграх і сформувати висновки для власної гри.

На основі цього аналізу можна визначити плюси та мінуси існуючих ігор з фізичними властивостями. До переваг можна віднести:

– використання фізичних властивостей як ядра геймплею, що підвищує залученість гравця;

– варіативність проходження рівнів завдяки різним фізичним властивостям об’єктів;

– інтуїтивно зрозуміла взаємодія з об’єктами через природні закони фізики;

– можливість створення унікальних ігрових ситуацій без складного сценарного дизайну.

Водночас можна виділити такі недоліки:

– складність балансування фізики для забезпечення комфортного керування;

– підвищені вимоги до оптимізації на мобільних пристроях;

– ризик непередбачуваної поведінки об’єктів, що може ускладнювати проходження;

– обмеження у візуальній складовій рівнів через слабкість процесорів мобільних пристроїв.

Це підтверджує, що рішення використовувати зміну фізичних властивостей об’єкта головною механікою гри є виправданим. Практика показує, що користувачу цікавіше грати тоді, коли фізика прямо впливає на процес і дозволяти долати перешкоди на рівнях різними варіантами. Головним під час розробки є приділення максимальної уваги оптимізації гри під мобільні пристрої, щоб фізика розраховувалася стабільно а керування об’єктом залишалось простим і приємним для користувача.

2 ВИБІР ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Ігровий рушій

Створення мобільного 3D проекту з активним використанням фізики вимагає підбору сучасного ігрового рушія, який буде здатен стабільно працювати з 3D-графікою та прораховувати фізичне моделювання в реальному часі. Оскільки кінцевий продукт орієнтований на смартфони, обране середовище має забезпечувати високу продуктивність при обмежених апаратних ресурсах, а також надавати зручні інструменти для швидкого прототипування механік та розширення архітектури коду.

На сьогодні лідерами серед інструментів для створення мобільних ігор є Unreal Engine, Unity та Godot:

– Unreal Engine: пропонує інструменти для генерації реалістичної графіки, проте він висуває високі вимоги до апаратних ресурсів смартфона, через що частіше застосовується для масштабних проектів AAA-класу. Інтерфейс додатку показано на рисунку 2.1.

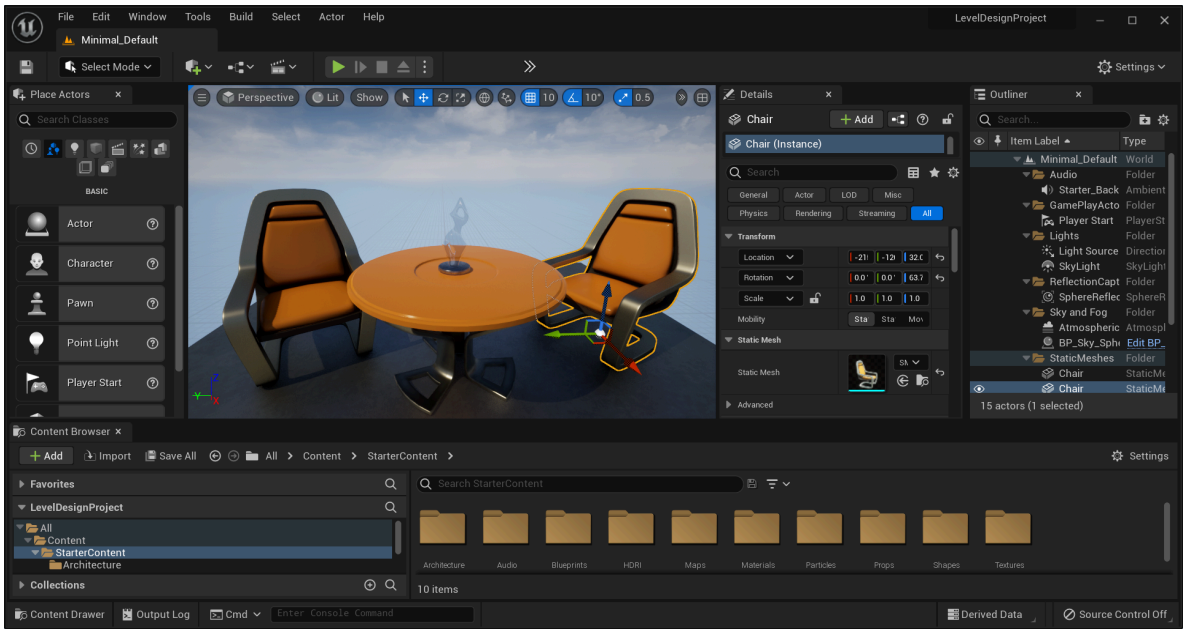


Рис. 2.1. Інтерфейс ігрового рушія Unreal Engine 5

– Godot: приваблює відкритим початковим кодом та низьким навантаженням на систему, проте на поточному етапі розвитку має обмежену базу готових інструментів для реалізації складних 3D-проектів мобільного сегмента. Інтерфейс додатку показано на рисунку 2.2.

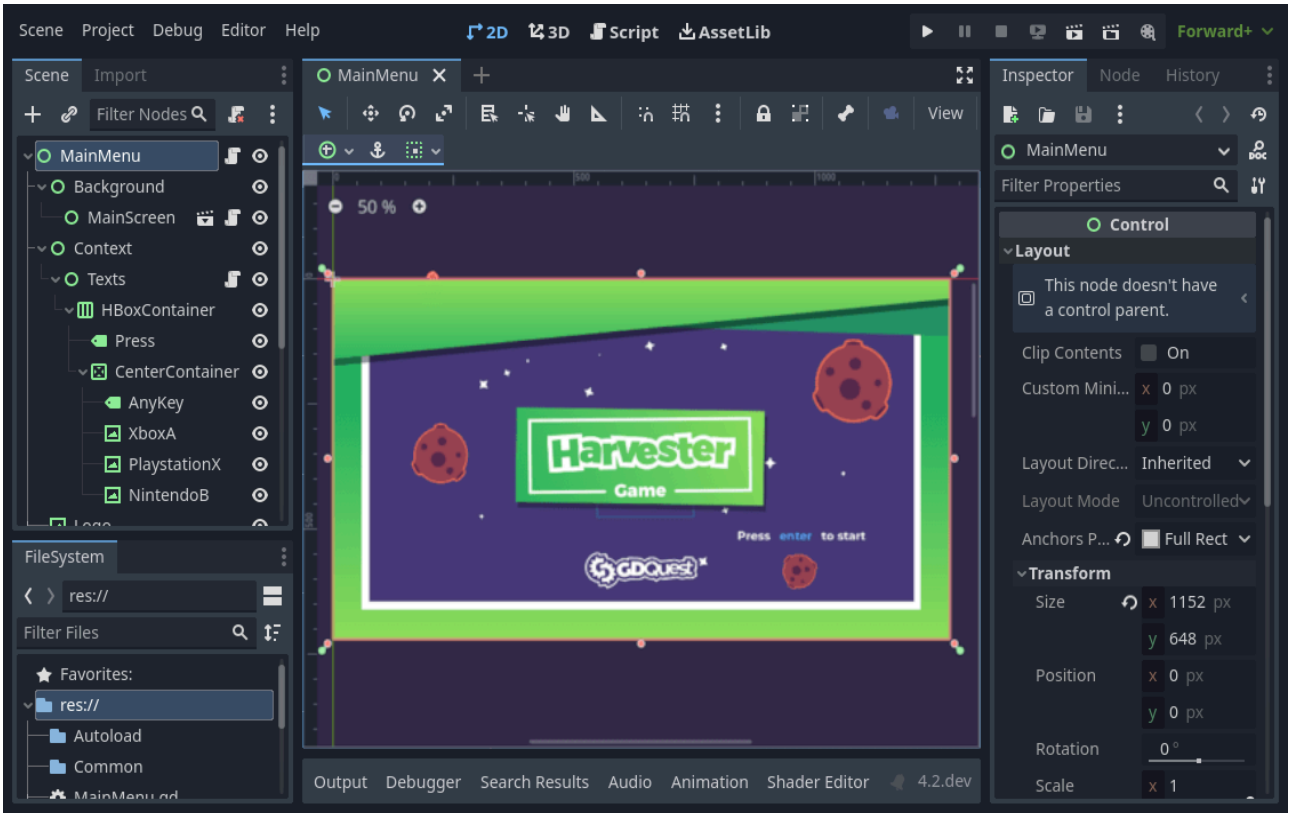


Рис. 2.2. Інтерфейс ігрового рушія Godot

– Unity: є найбільш раціональним вибором для казуальних мобільних ігор, тому що наявність великої кількості документації та великої бібліотеки готових рішень дозволяють ефективно оптимізувати гру під обмежені ресурси пристроїв та зосередитися на розробці унікальної ігрової логіки, замість витрачання часу на створення базових систем з нуля. Інтерфейс додатку показано на рисунку 2.3.

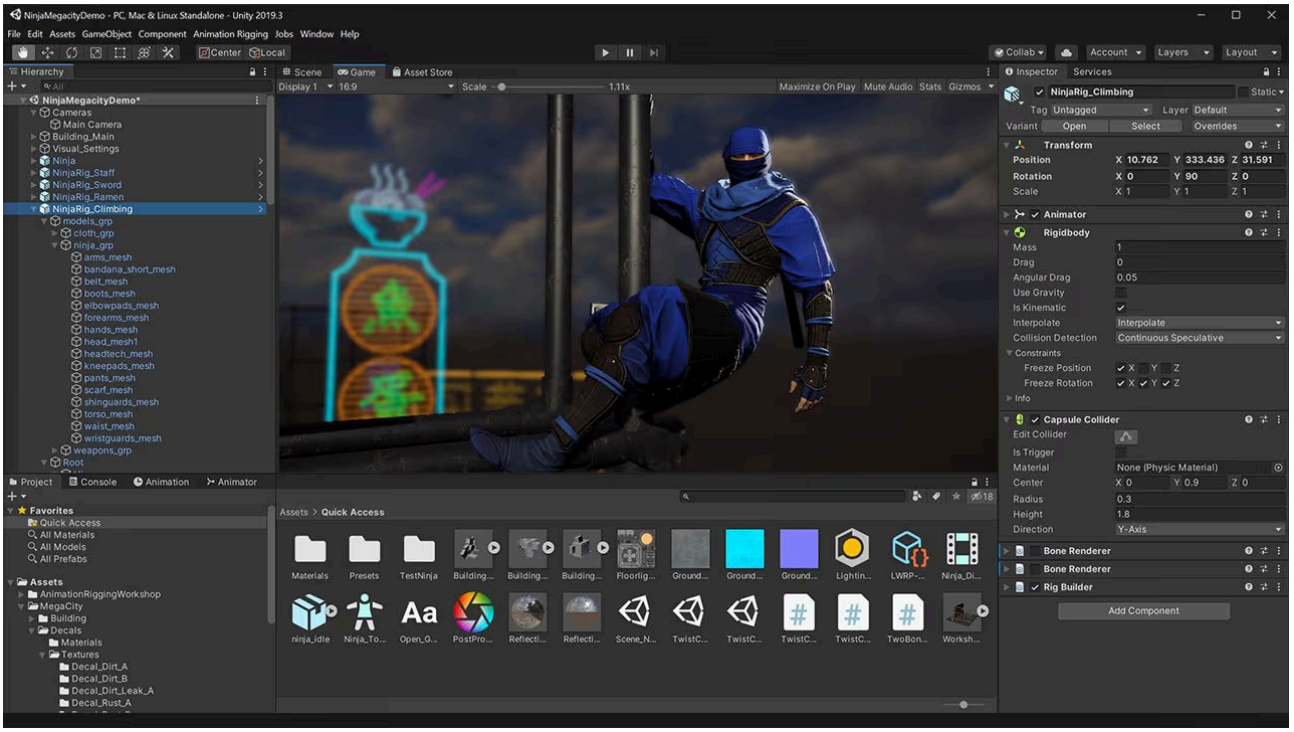


Рис. 2.3. Інтерфейс ігрового рушія Unity

Порівняльний аналіз ключових характеристик рушіїв наведено у таблиці 2.1.

Таблиця 2.1

Аналіз ігрових рушіїв

Показник	**Unity**	**Unreal Engine**	**Godot**
Підтримка мобільних платформ	+	+	+
Якість 3D графіки	висока	дуже висока	середня
Фізичний рушій	PhysX	Chaos	вбудований
Простота освоєння	середня	складна	проста
Оптимізація для мобільних	висока	середня	середня
Підходить для казуальних ігор	так	частково	так
Кросплатформеність	+	+	+

Продовження таблиці 2.1

Аналіз ігрових рушіїв

Показник	**Unity**	**Unreal Engine**	**Godot**
Наявність інструментів для 3D розробки	+	+	+
Підтримка розширень	+	+	+
Мова програмування	C#	C++ / Blueprints	GDScript / C#

Порівняння ігрових рушіїв у таблиці 2.1 показує, що для казуальної мобільної 3D-гри найкраще підходить саме Unit. ,На відміну від Unreal Engine, який є занадто вимогливим до апаратних характеристик смартфонів і швидко розряджає батарею, Unity забезпечує стабільну продуктивність на мобільних пристроях, тоді як рушію Godot все ще бракує готових інструментів для якісної роботи з тривимірною графікою.

Головні причини вибору Unity для цього проекту:

– Вбудована фізика (NVIDIA PhysX): рушій має готові алгоритми для розрахунку гравітації, маси та зіткнень, що дозволяє не писати математичні формули руху вручну, що дуже важливо для гри, де геймплей побудований на фізичних механіках.

– Компонентна структура: логіка гри розділена на окремі блоки (компоненти). Можна легко змінювати або додавати скрипти на один об’єкт, не порушуючи при цьому роботу інших елементів.

– Система префабів: дає можливість створювати готові шаблони об’єктів та швидко розставляти їх на ігрових рівнях. При зміні базового шаблону всі його копії на сценах оновлюються автоматично, що значно економить час розробки.

Unity дозволяє швидко реалізувати ігрову логіку та добре оптимізувати застосунок під мобільні пристрої Android та iOS, що повністю відповідає вимогам до проекту.

2.2 Мова програмування

Для написання ігрової логіки, налаштування фізики, інтерфейсу та обробки дій гравця потрібна мова програмування, яка працює швидко і при цьому дозволяє зручно будувати структуру коду. Від цього вибору безпосередньо залежить, чи буде гра працювати стабільно під час довгих сесій і чи легко буде додавати до неї нові функції в процесі розробки.

Основним інструментом в Unity є мова C#. З її допомогою пишуться всі скрипти, налаштовується взаємодія між об’єктами на сцені та керуються внутрішні процеси самого рушія.

До головних переваг C# можна віднести:

– Строга типізація: допомагає знаходити й виправляти помилки ще на етапі написання коду, що робить логіку гри більш надійною;

– Автоматичне керування пам’яттю: завдяки вбудованому збирачу сміття (Garbage Collector) розробнику не потрібно вручну очищати ресурси, що спрощує контроль за навантаженням на процесор;

– Сучасний синтаксис: мова має зручні конструкції, які дозволяють писати код швидше, а сам він залишається простим і зрозумілим для читання.

Оскільки C# тісно пов’язаний із самим рушієм, через код можна легко керувати всіма елементами на сцені. Зазвичай написані скрипти взаємодіють із такими базовими компонентами Unity:

– Transform: відповідає за координати, розмір та поворот об'єктів у тривимірному просторі;

– Rigidbody: додає об’єктам фізичні властивості, дозволяючи налаштувати їхню масу, гравітацію чи швидкість руху;

– Collider: задає фізичні межі предметів і фіксує моменти, коли вони стикаються між собою на рівні;

– Animator: керує станами об’єктів і в потрібний момент запускає готові анімації;

– UI-система: відповідає за роботу меню, кнопок та інших елементів інтерфейсу, зчитуючи дотики до екрана смартфона.

Оскільки гра розробляється для мобільних пристроїв, її стабільна робота без гальмувань є головним пріоритетом. Для цього під час збирання проекту застосовується технологія IL2CPP, яка переводить весь написаний код у C++, що дуже підвищує загальну продуктивність. Щоб уникнути зависань екрана та падіння частоти кадрів під час завантаження нових рівнів чи збереження прогресу, використовуються корутини та асинхронні функції, за допомогою яких всі ці фонові процеси проходять абсолютно непомітно для гравця.

Одна з переваг C# це величезна база знань та активна спільнота, що допоможе швидко знайти вже готове рішення або підказку.

Саме тому для написання логіки гри було обрано мову C#. Вона без затримок обробляє зіткнення між об’єктами і загалом чудово підходить для оптимізації гри під Android та iOS.

2.3 Вибір допоміжних інструментів та сервісів

Повноцінне проектування мобільного 3D проекту, крім базового ігрового рушія, вимагає залучення комплексу спеціалізованого програмного забезпечення. Ці інструменти необхідні для створення тривимірного візуалу, підготовки елементів користувацького інтерфейсу, редагування аудіосупроводу та налаштування хмарної архітектури для збереження даних. Спільне використання сторонніх сервісів дозволяє автоматизувати рутинні процеси розробки та гарантує високу якість кінцевого програмного продукту.

Весь стек додаткових інструментів та сервісів, залучених до розробки гри, можна розділити за функціональним призначенням на такі категорії:

Графічне моделювання та проектування інтерфейсу

Blender: професійний пакет для роботи з тривимірною графікою з відкритим кодом. У поточному проекті він застосовується для побудови низькополігональних (Low-Poly) моделей ігрових об’єктів, налаштування розгортки, текстурування та

оптимізації полігональної сітки. Оскільки мобільні процесори мають обмежені ресурси, у Blender проводиться ручний контроль кількості вершин та трикутників, щоб зберегти баланс між деталізацією візуалу та продуктивністю. Готові асети експортуються у стандартний формат .fbx для безперешкодного імпорту в Unity із збереженням опорних точок (Pivot Points).

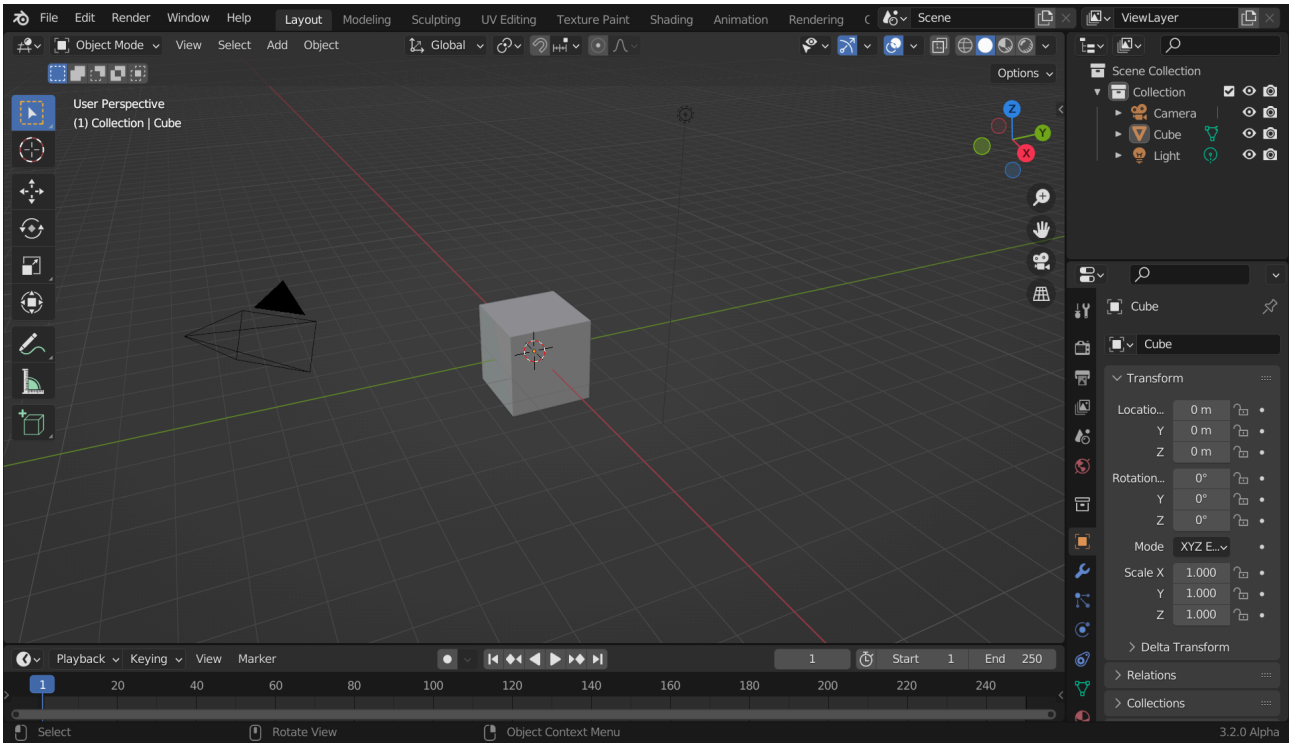


Рис. 2.4. Інтерфейс середовища Blender

Figma: хмарне середовище для векторного дизайну та прототипування. Інструмент використовується для створення макетів графічного інтерфейсу (UI), включаючи головне меню, панелі налаштувань, HUD-елементи ігрового екрана та вікна завершення рівня. Попереднє макетування у Figma дозволяє заздалегідь протестувати зручність керування, вибудувати правильну навігаційну логіку та підготувати графічні нарізки кнопок та іконок для їх подальшого перенесення в Unity UI-систему.

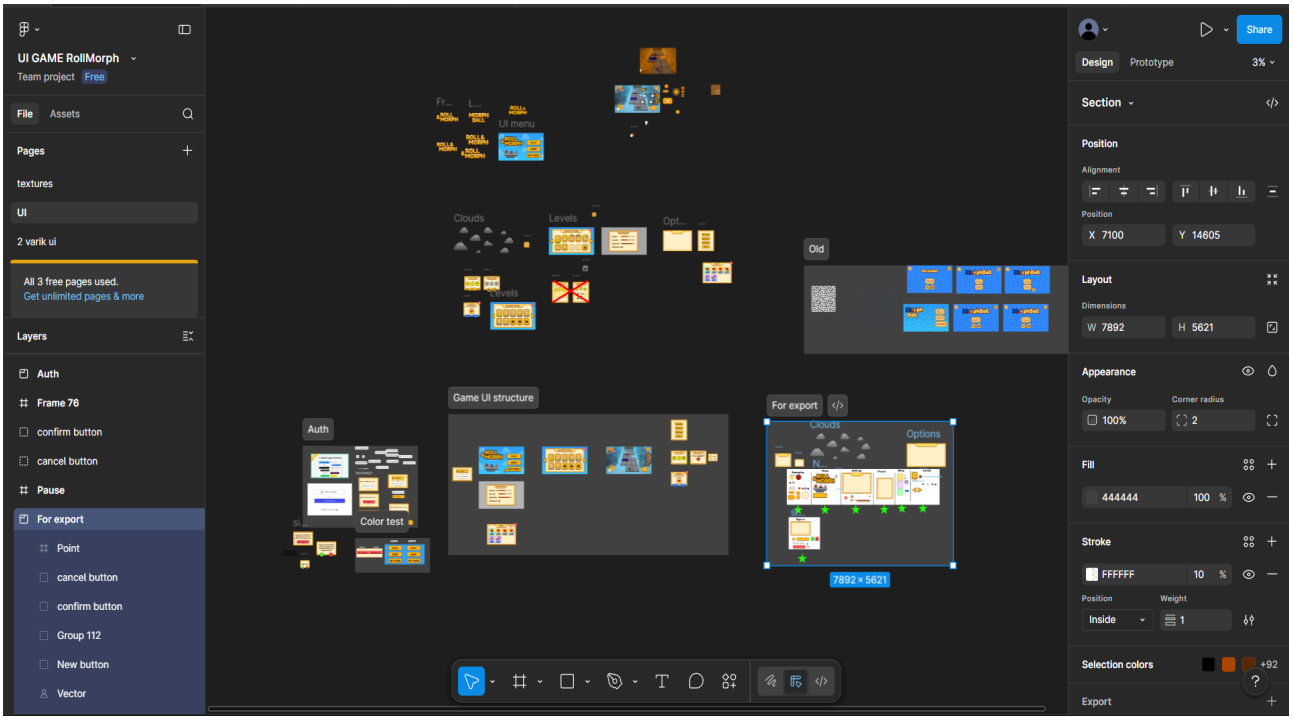


Рис. 2.5. Інтерфейс середовища графічного дизайну Figma

Серверна частина та збереження даних

Google Firebase: комплексна хмарна платформа, інтегрована в проект для реалізації серверних функцій без необхідності розгортання власного заліза. Вона закриває такі технічні завдання:

– Realtime Database: хмарне збереження ігрового прогресу, поточного рівня, кількості зібраних предметів та налаштувань користувача;

– Authentication: базова ідентифікація гравців для синхронізації їхніх досягнень між різними пристроями;

– Analytics: збір технічної інформації про поведінку користувачів, тривалість сесій та можливі критичні помилки, що необхідно для подальшого балансування рівнів та оптимізації коду.

Робота зі звуком та аудіоефектами

Audacity: кросплатформений аудіоредактор, залучений для підготовки звукового середовища гри. За допомогою цього інструменту виконується обрізка, очищення від шумів та нормалізація гучності звукових ефектів (SFX) — наприклад, звуків зіткнення сфери з перешкодами, збору бонусів чи активації

тригерів. Оброблені аудіодоріжки конвертуються у формати .wav або .mp3 (залежно від вимог до стиснення) для подальшого відтворення через компоненти AudioSource в Unity.

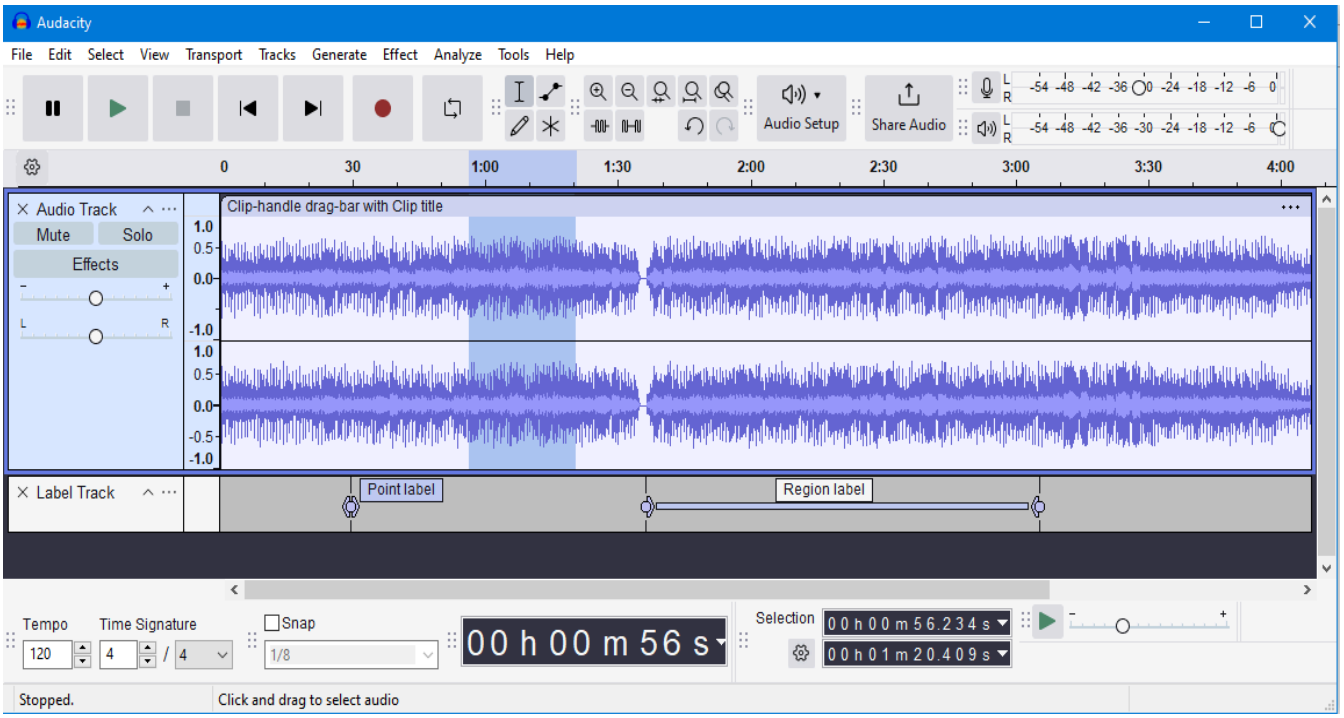


Рис. 2.6. Інтерфейс середовища обробки звуку Audacity

Сформований набір допоміжних засобів, що включає Blender, Figma, Firebase та Audacity, дозволяє організувати повний, замкнений цикл розробки мобільної гри. Інтеграція цих рішень забезпечує гнучкість на кожному етапі створення продукту — від первинного візуального начерку до хмарної синхронізації даних на смартфонах користувачів.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОЇ 3D ГРИ

3.1 Загальна архітектура мобільної 3D гри

В основі розробленої мобільної 3D гри лежить компонентна архітектура ігрового рушія Unity. Такий підхід забезпечує модульність системи, високу швидкість налагодження коду та можливість гнучкого масштабування функціоналу без потреби в глобальному переписуванні програмної логіки. Вся структура проекту розділена на логічні сцени та функціональні модулі, що дозволяє ізолювати окремі етапи ігрового процесу.

Архітектурно проект поділяється на кілька ключових рівнів:

– Рівень інтерфейсу: стартовий вузол системи. Сюди включено головне меню, панелі налаштувань та систему вибору рівнів. Ця сцена виконує роль центрального диспетчера, що керує навігацією та завантаженням контенту.

– Рівень ігрового процесу: кожна сцена рівня є самостійним середовищем із власною геометрією, набором пасток, об’єктів та тригерів. Така ізоляція дозволяє додавати нові рівні як окремі файли сцени, не впливаючи на стабільність основної логіки гри.

– Модуль керування гравцем: окремий компонент, що інкапсулює логіку руху, реакцію на тач-ввід та фізичну взаємодію сфери з ігровим оточенням.

– Система камери: модуль, що забезпечує плавне слідування за об’єктом гравця, підтримуючи стабільний кут огляду незалежно від динаміки руху.

– Система інтерфейсу користувача: шар відображення, що відповідає за виведення поточної статистики, прогресу та обробку подій натискання на елементи екрана.

Для візуалізації логічних переходів та послідовності дій гравця в системі було побудовано діаграму діяльності.

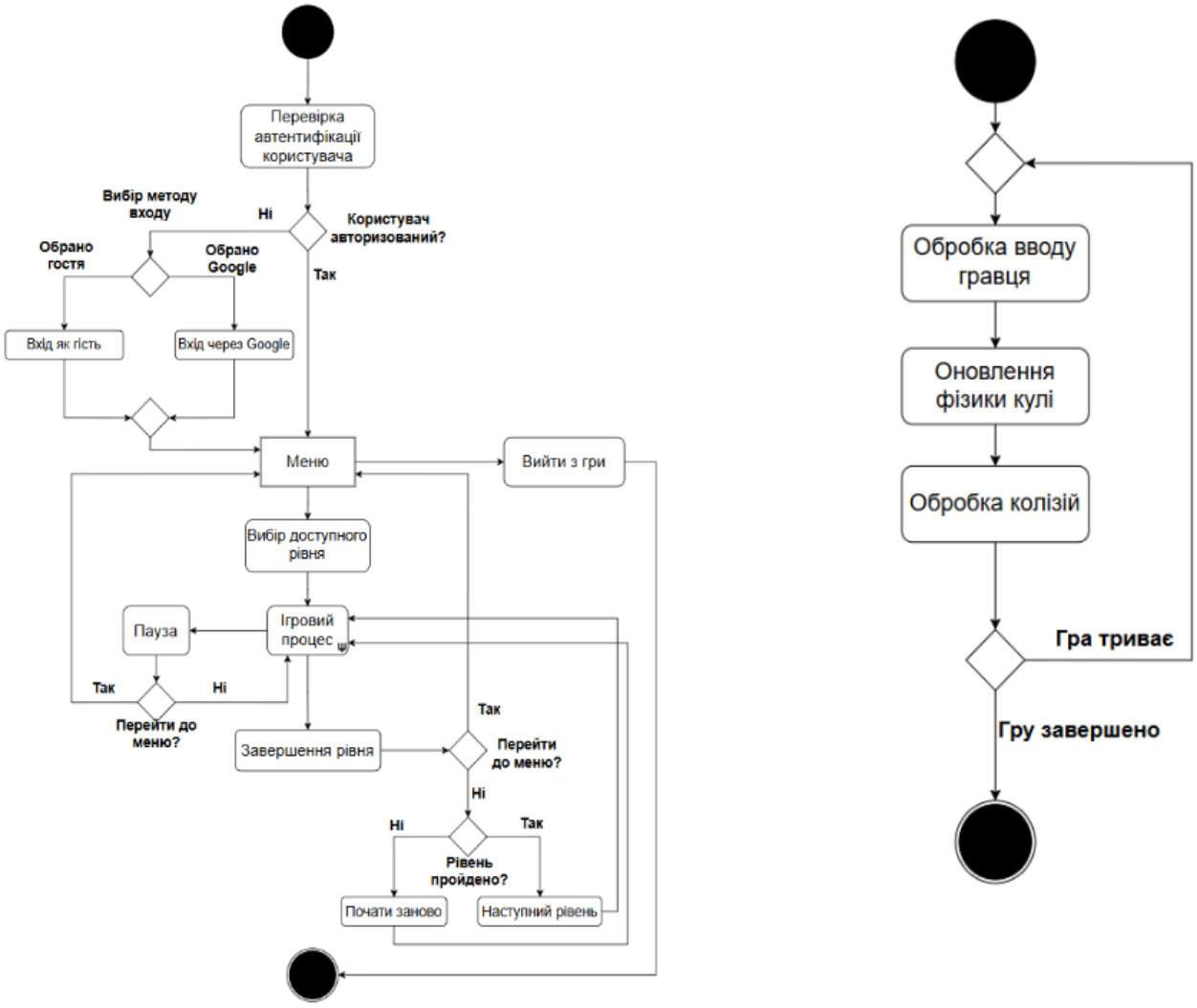


Рис. 3.1. Діаграма діяльності ігрового процесу

Діаграма відображає послідовність дій користувача: запуск гри, вибір рівня, проходження ігрового рівня та перевірку умови завершення. У разі поразки рівень перезапускається, а при успішному проходженні гравець переходить до наступного рівня або повертається до сцени вибору рівнів.

Використання компонентного підходу та чіткий поділ на сцени створюють стабільний архітектурний каркас. Це мінімізує зв’язність між модулями, що критично важливо для подальшого розвитку проекту, оскільки будь-які зміни в геометрії чи логіці конкретного рівня не призводять до порушення роботи глобальних скриптів керування.

3.2 Проектування ігрових механік

Основа ігрового процесу — це керування кулею, яка повинна долати рівні та оминати різні перешкоди. Користувач взаємодіє з грою за допомогою віртуального джойстика. Рух ігрового об’єкта побудований на вбудованій фізиці, яка прораховує поведінку кулі в реальному часі. Система враховує масу об’єкта, силу поштовху та дію гравітації. Через це куля котиться дуже природно та має власну інерцію.

Головна фішка геймплею, яка додає грі інтерактивності — це можливість змінювати фізичні властивості кулі. Для цього було створено три режими, які мають різну вагу, швидкість та розміри:

– швидкий режим: куля стає легкою і рухається на максимальній швидкості;

– важкий режим: куля стає більшою і котиться повільніше, але вона більш стійка на платформі й може легко розштовхувати або зсувати легкі блоки та перешкоди попереду себе;

– магнітний режим: поступається іншим режимам у швидкості, але дає кулі унікальну властивість притягуватися до залізних чи магнітних стін та платформ, що дозволяє долати вертикальні ділянки або котитися по стіні.

Вся взаємодія у грі побудована зіткненнях об’єктів. Перешкоди на рівнях бувають різними: одні об’єкти просто блокують рух або відбивають кулю, а інші — рухаються, тому гравцеві доводиться чекати правильного моменту, щоб проскочити повз них. Якщо куля торкається пастки, гравець одразу програє. Після цього рівень перезапускається з самого початку, або з останнього активованого чекпоїнту.

Рівні ускладнюються поступово, від першого до останнього. На перших локаціях немає нічого складного, щоб гравець міг спокійно розібратися, як куля реагує на дотики і як вона котиться. Але далі на рівнях з’являються важкі комбінації з рухомих блоків та пасток. Щоб пройти такі місця, доводиться швидко реагувати й постійно перемикати режими кулі (наприклад, ставати важким чи магнітним). Завдяки такому плавному переходу гра не набридає і в неї стає цікавіше грати.

3.3 Реалізація ігрової логіки

Код було реалізовано за допомогою мови програмування C# в Unity, розділивши код на окремі модулі. За допомогою такого підходу обов’язки між класами розподіляються чітко, де окремий модуль керує фізикою руху, а інші відповідають за обробку зіткнень. Це дозволяє модифікувати код без втручання в інші модулі, що значно спростить роботу.

Система керування ігровим об’єктом (кулею) базується на взаємодії таких класів:

– BallSettings: об’єкт типу ScriptableObject, що зберігає набір фізичних характеристик (маса, швидкість, розмір, сила стрибка). Використання цього класу дозволяє гнучко налаштовувати баланс гри через інспектор Unity без зміни коду.

– BallTypeSettings: контейнер, який пов’язує конкретний фізичний режим (наприклад, «важкий» або «магнітний») з відповідними налаштуваннями BallSettings та візуальним матеріалом.

– BallController: контроллер, що зберігає посилання на поточний стан сфери та ініціює перемикання фізичних конфігурацій під час проходження рівня.

– BallMovement: клас, який отримує вхідні дані від користувача, обробляє вектори сили та передає їх на компонент Rigidbody для здійснення фізичного руху.

– BallCollision: модуль, що відповідає за реєстрацію зіткнень через методи OnCollisionEnter та OnTriggerEnter, передаючи події до системи керування рівнем при контакті з пастками чи фінішними зонами.

Ця структура дозволяє незалежно тестувати кожен окремий клас (детальніший код наведено у Додатку Б).

Рух об’єкта в ігровому світі базується на прямому керуванні фізичним рушієм Unity:

– Застосування сил: вектори вводу користувача з тач-інтерфейсу трансформуються у фізичні сили, що діють на центр мас Rigidbody.

– Динамічне оновлення: при зміні фізичного режиму BallController миттєво оновлює параметри в BallMovement (масу, швидкість, чутливість), що дозволяє гравцю відчувати зміну характеристик кулі в режимі реального часу без зупинки гри.

– Фізична інерція: об’єкт враховує сили тертя та гравітації, що створює природну для 3D-середовища поведінку.

Організація взаємодії між компонентами побудована на принципах інкапсуляції:

– Керуючі класи лише передають параметри, а виконавчі класи BallMovement, BallCollision безпосередньо впливають на стан ігрового об’єкта.

– При контакті з перешкодами BallCollision звертається до LevelStateManager для фіксації результату (програш або перехід на наступний рівень).

Діаграма показує структуру класів і як вони взаємодіють між собою. Головну роль відіграє клас BallController, який контролює зміну фізичних режимів об’єкта. Клас BallMovement відповідає за рух об’єкта і взаємодіє з BallController для зміни режимів. А клас BallCollision обробляє зіткнення об’єкта з іншими об’єктами.

Завдяки такій структурі класи які відповідають за рух об’єкта працюють окремо від інших процесів і не потребують зміни коду в інших класах якщо потрібно щось виправити або додати нове. Це дуже спрощує подальшу підтримку.

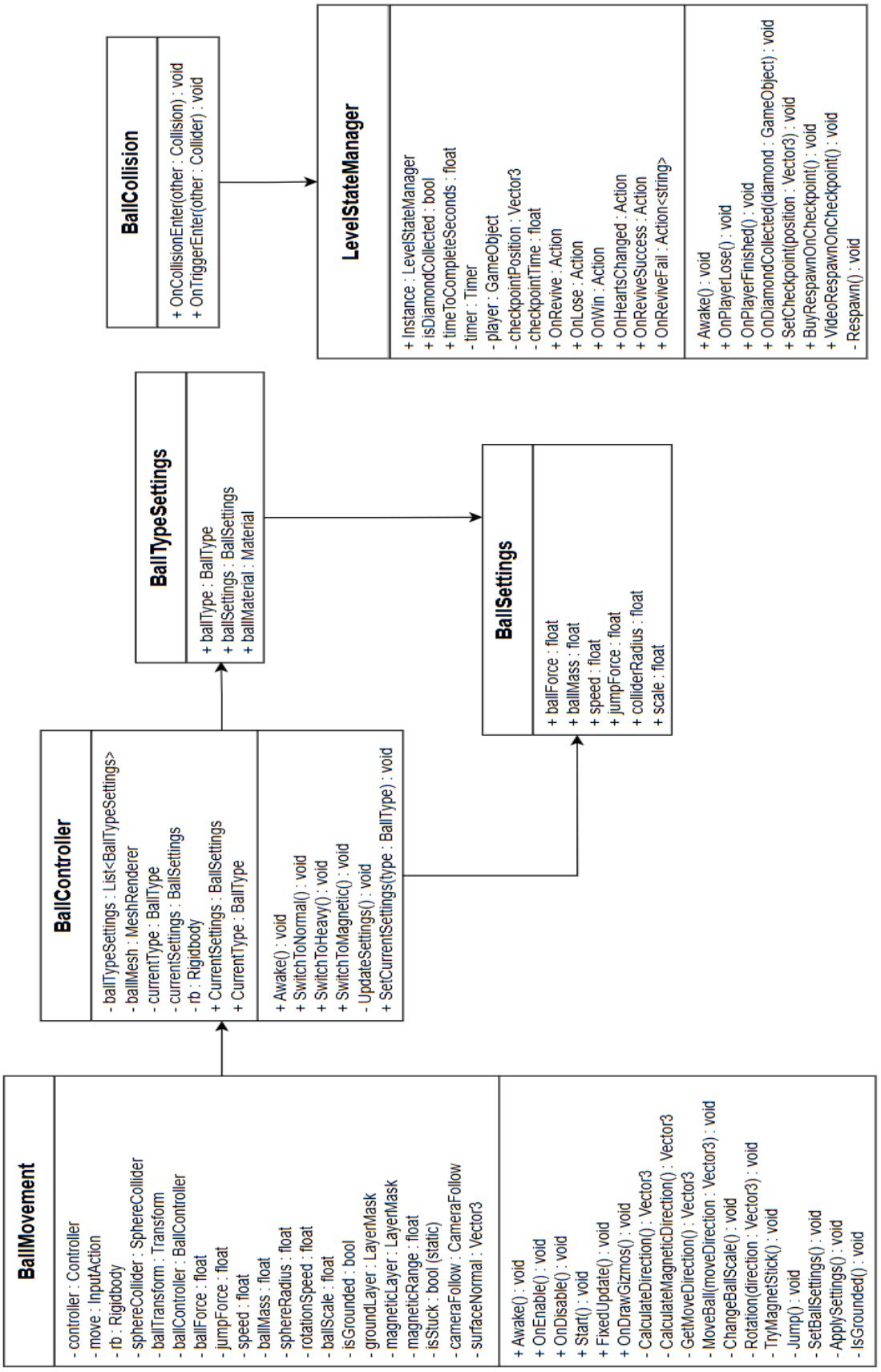


Рис. 3.2. Діаграма класів системи керування кулею та фізикою

3.4 Реалізація інтерфейсу користувача

Користувацький інтерфейс (UI) гри розроблений на базі стандартної системи Canvas в Unity. Архітектура інтерфейсу побудована за принципом розбиття на окремі логічні екрани, кожен з яких відповідає за певний стан ігрового циклу (меню, HUD, вікна налаштувань тощо). Це забезпечує зручність керування та інтуїтивно зрозумілу навігацію.

Головне меню та навігація

Стартова сцена гри слугує «хабом», з якого гравець отримує доступ до всіх функціональних розділів:

– Кнопка Play: ініціює перехід до вибору ігрових рівнів.

– Кнопка Shop: відкриває меню вибору зовнішнього вигляду об’єкта (скинів).

– Кнопка Settings: активує налаштування параметрів гри.

Рис. 3.3. Головне меню

При переході до вибору рівня відкривається окремий інтерфейсний шар, що відображає прогрес гравця та дозволяє обрати доступну для проходження локацію.

Рис. 3.4. Екран вибору рівнів

Налаштування

Меню налаштувань дозволяє користувачеві адаптувати технічні та графічні параметри під апаратні можливості смартфона:

– Music / SFX: слайдери для окремого регулювання гучності музичного супроводу та звукових ефектів.

– Shadows / Effects: перемикачі для ввімкнення/вимкнення тіней та додаткових візуальних ефектів (пост-обробки), що дозволяє оптимізувати продуктивність на слабких пристроях.

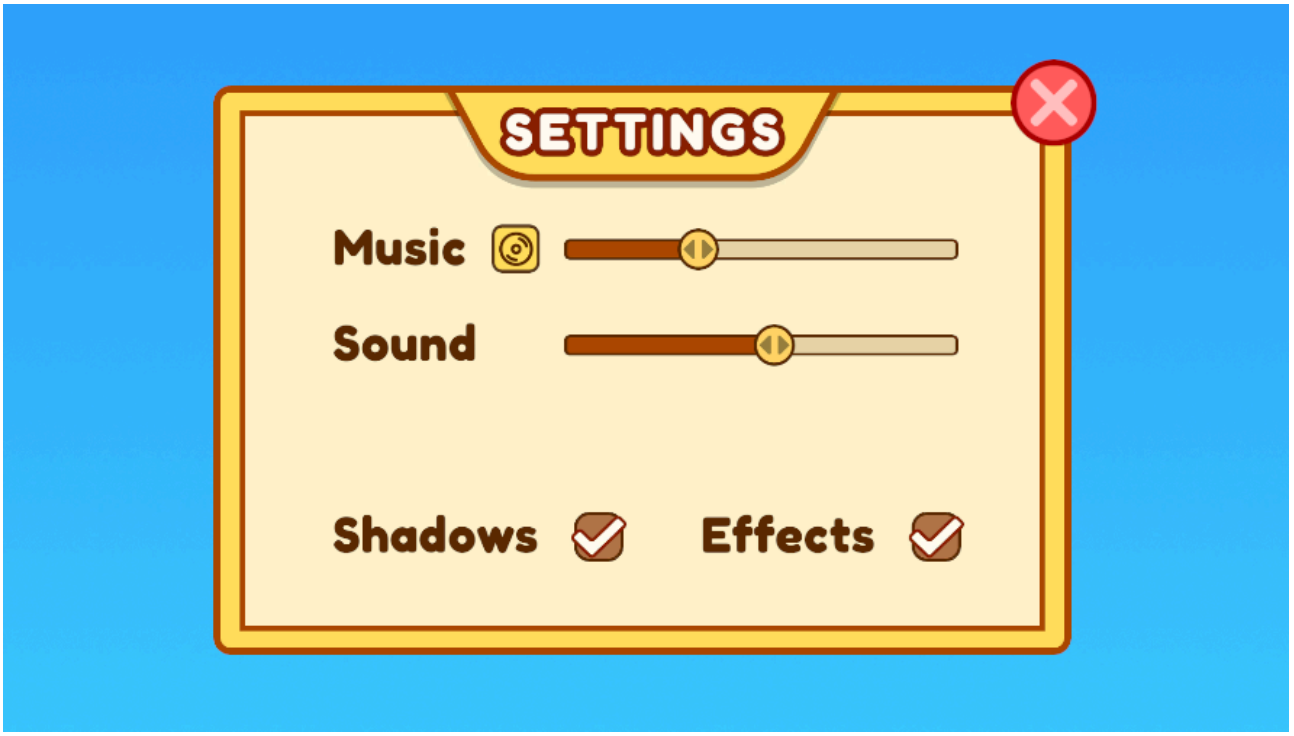


Рис. 3.5. Меню налаштувань

Ігровий інтерфейс (HUD)

Під час проходження рівня на екрані відображається HUD (Heads-Up Display), що містить інструменти керування та індикатори стану:

– Керування: віртуальний джойстик (напрямок руху) та кнопка стрибка.

– Фізичні режими: блок із трьох кнопок для миттєвого перемикання фізичних властивостей кулі.

– Індикація: таймер проходження, лічильник зібраних предметів та індикатор життів.

– Системне: кнопка виклику меню паузи.

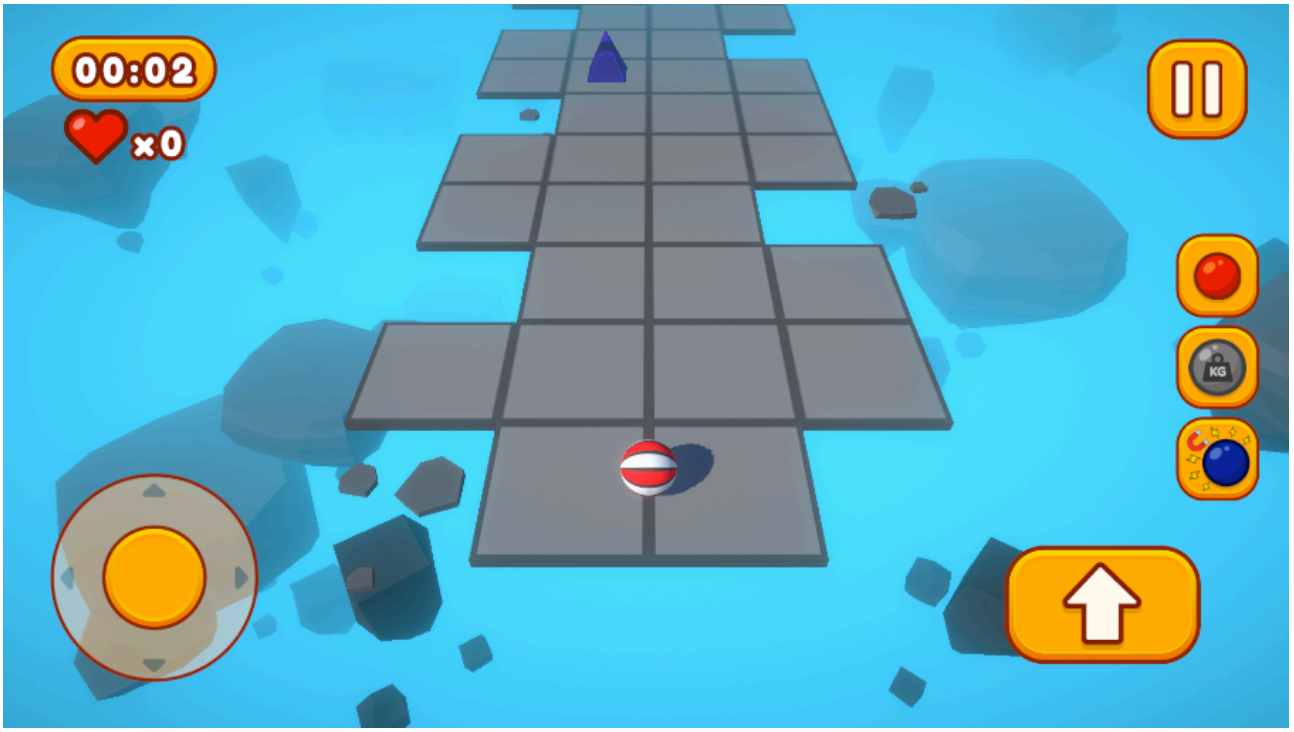


Рис. 3.6. Ігровий інтерфейс (HUD)

Системні екрани (Пауза та стани рівня)

– Меню паузи: зупиняє фізичну симуляцію (Time.timeScale = 0) і пропонує опції відновлення, перезапуску або виходу в головне меню.

– Екрани завершення: окремі інтерфейсні префаби для станів «Перемога» та «Поразка», що з’являються залежно від результату LevelStateManager.

Рис. 3.7. Меню паузи

Рис. 3.8. Екран перемоги

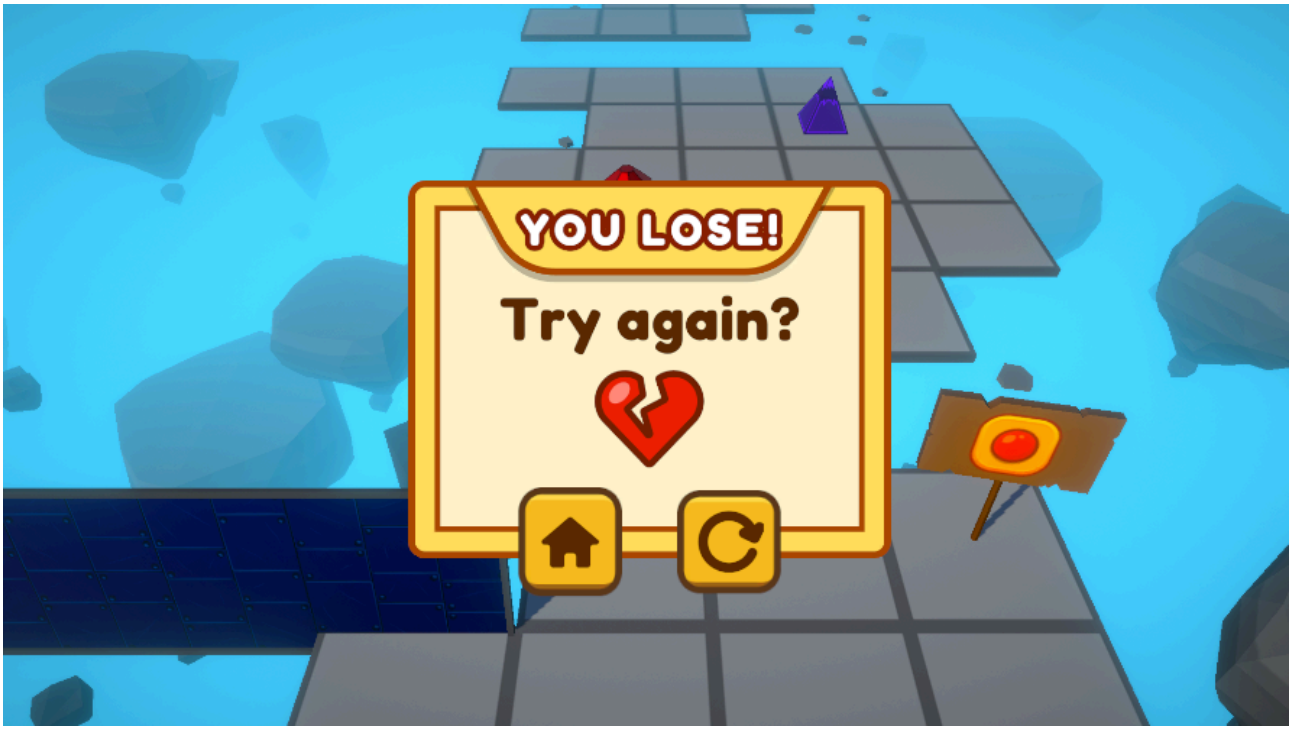


Рис. 3.9. Екран поразки

Використання модульної системи Canvas дозволяє легко модифікувати зовнішній вигляд інтерфейсу, додавати нові кнопки чи анімації, не зачіпаючи при цьому основну ігрову логіку. Всі елементи UI налаштовані з урахуванням адаптації під різні співвідношення сторін екранів сучасних мобільних пристроїв.

3.5 Розробка 3D-моделей

Візуальна складова гри реалізована за допомогою тривимірних моделей, які формують ігрове середовище та забезпечують сприйняття ігрового процесу.

Основні об’єкти сцени, включаючи платформи, перешкоди та допоміжні елементи, були створені з урахуванням вимог до продуктивності мобільних пристроїв.

Для моделювання було використано середовище Blender, що дозволило створити оптимізовані low-poly моделі з невеликою кількістю полігонів. Цей підхід забезпечує стабільну роботу гри на різних пристроях без значного навантаження на систему.

У процесі розробки моделей було враховано:

– спрощену геометрію об’єктів для підвищення продуктивності;

– використання базових матеріалів без складних шейдерів;

– узгоджений стиль усіх елементів середовища.

Створені моделі імпортуються у Unity та використовуються як складові ігрових сцен. Після імпорту для них налаштовуються матеріали, колайдери та, за необхідності, фізичні параметри.

На рисунку 3.10 представлено приклад набору 3D-моделей, що використовуються у грі.

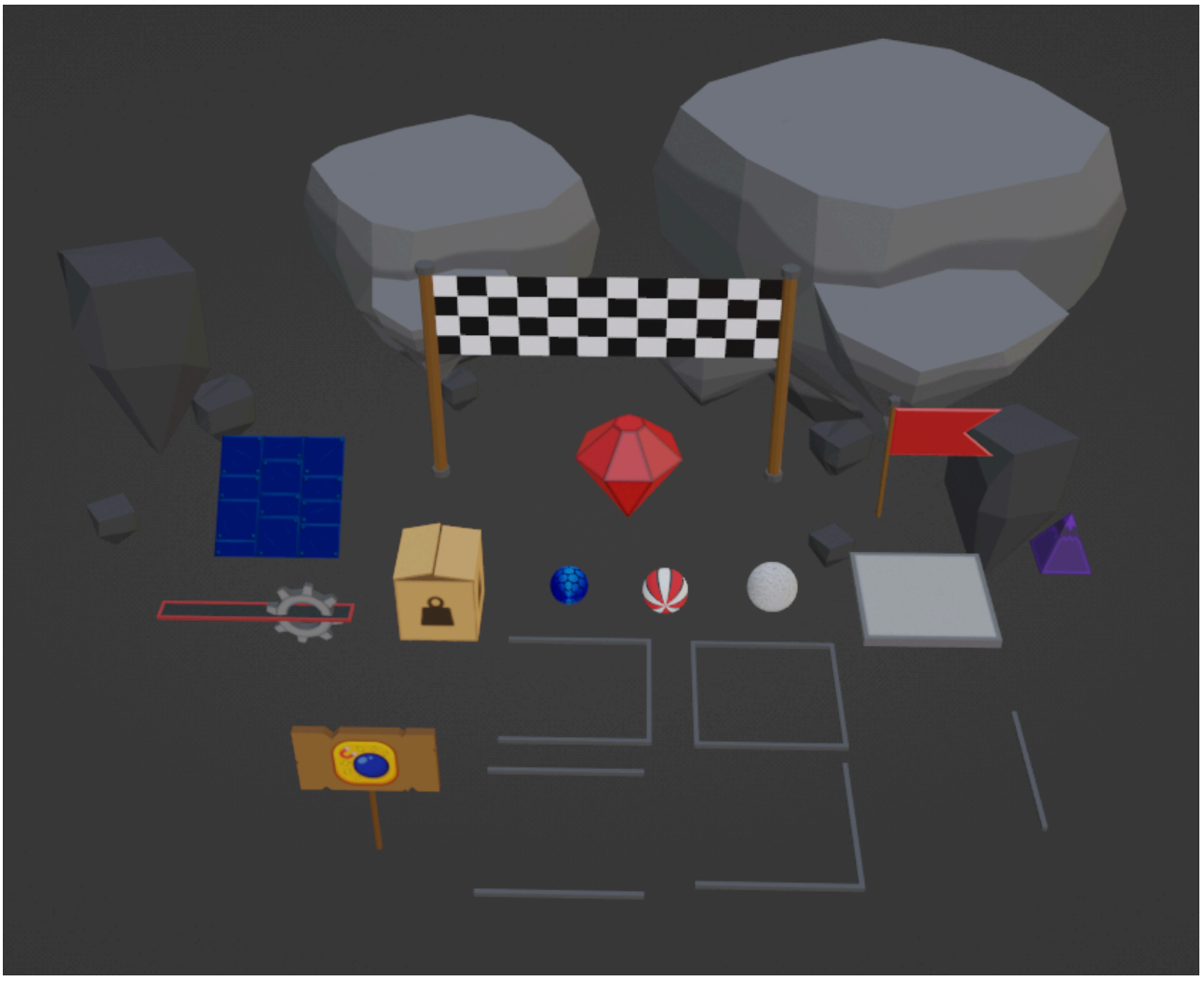


Рис. 3.10. Набор 3D-моделей гри

Завдяки оптимізованим топологіям об’єктів гра не втрачає візуальної складової, зменшуючи при цьому навантаження на слабкі мобільні пристрої з обмеженими апаратними ресурсами. Це дозволяє гравцеві комфортно взаємодіяти з об’єктами без затримок у рендерингу сцен.

3.6 Реалізація системи рівнів

Система рівнів у грі побудована на використанні окремих сцен у середовищі Unity. Кожен рівень реалізовано як самостійна сцена, що містить власну структуру об’єктів, перешкод та умов проходження. Це забезпечує незалежність рівнів і спрощує додавання нових етапів без внесення змін у загальну архітектуру проекту.

Перехід на рівні здійснюється через меню вибору рівнів, де гравець може обрати необхідний йому рівень. Завантаження рівня реалізовано за допомогою стандартних засобів Unity.

Кожен рівень має типову структуру, яка включає:

– стартову позицію гравця;

– ігрову поверхню (платформи);

– перешкоди, що ускладнюють проходження;

– фінішну зону, досягнення якої означає завершення рівня.

На рисунку 3.11 представлено приклад ігрового рівня.

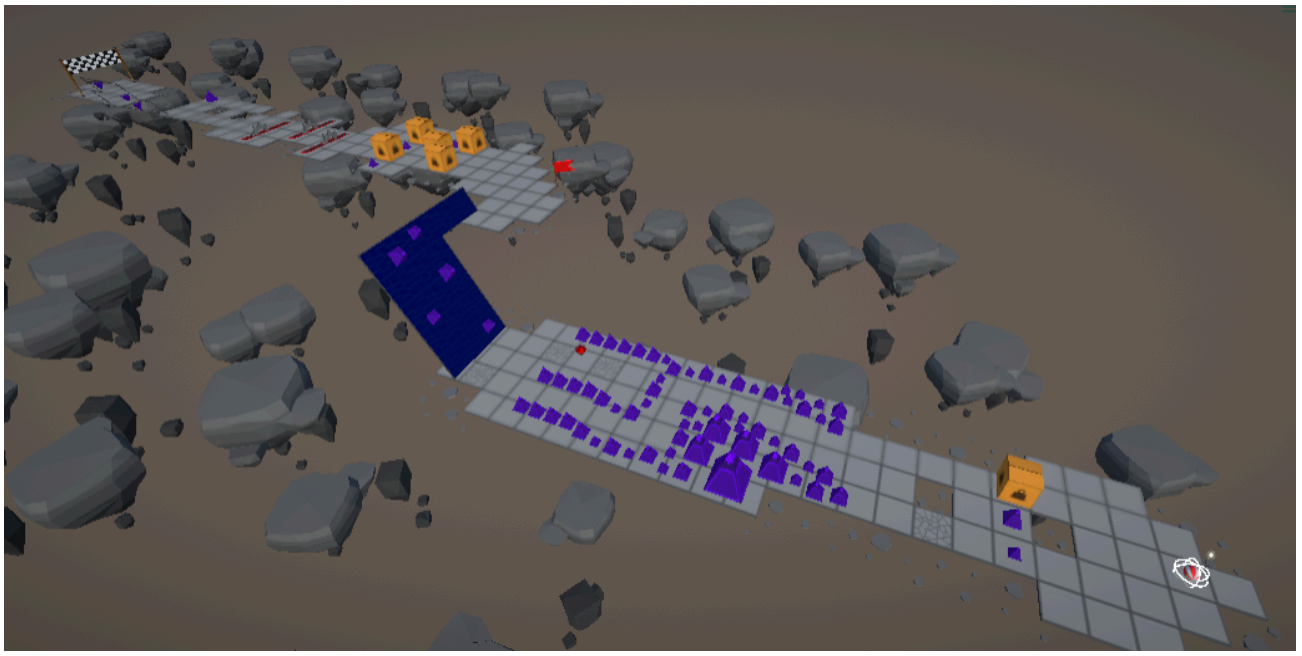


Рис. 3.11. Приклад одного з рівнів

Логіка проходження рівня базується на досягненні кінцевої точки без зіткнення з небезпечними об’єктами. У разі досягнення фінішу гравцю відображається екран перемоги, тоді як зіткнення з перешкодами призводить до завершення спроби та відображення екрана поразки.

Перезапуск рівня може виконуватися:

– автоматично після поразки;

– вручну через меню паузи;

– шляхом повторного вибору рівня в меню.

Для керування станами рівня використовується скрипт LevelStateManager, який обробляє події завершення рівня, його перезапуску та переходів між сценами.

Зі збільшенням номера рівня складність поступово зростає за рахунок ускладнення структури рівня, збільшення кількості перешкод та необхідності використання різних фізичних режимів кулі.

3.7 Реалізація системи збереження даних та безпеки

Для забезпечення хмарної синхронізації ігрового прогресу та ідентифікації користувачів у проекті інтегровано платформу Google Firebase. Використання цього сервісу дозволяє гравцям зберігати досягнення та продовжувати гру на різних пристроях, зберігаючи цілісність даних.

Архітектура системи збереження

Система оперує двома рівнями зберігання інформації:

– Хмарне сховище (Firebase Realtime Database): призначене для критично важливих даних, що прив’язані до облікового запису користувача (поточний рівень, статус розблокованих локацій, ігрові досягнення).

– Локальне сховище (PlayerPrefs): використовується для зберігання налаштувань графіки та звуку (рівень гучності, перемикачі тіней та ефектів). Локальне зберігання цих параметрів забезпечує миттєвий відгук інтерфейсу без потреби в мережевих запитах.

Механізм аутентифікації та ідентифікації

Безпека даних базується на унікальній ідентифікації користувача (UID). Процес працює за таким алгоритмом:

1. При запуску гри система ініціює запит до модуля Firebase Auth.
2. Після успішної авторизації користувач отримує унікальний UID, який слугує ключем для всіх операцій з базою даних.
3. Всі запити на запис або читання автоматично фільтруються сервером згідно з цим ідентифікатором.

Правила безпеки

Контроль доступу до даних реалізовано безпосередньо на стороні сервера Firebase. Правила безпеки конфігуруються таким чином, щоб унеможливити доступ одного користувача до інформації іншого:

– Читання: дозволено лише в тому випадку, якщо request.auth.uid співпадає з ідентифікатором, під яким зберігаються дані.

– Запис: обмежено аналогічно, операція доступна лише власнику запису (власнику UID).

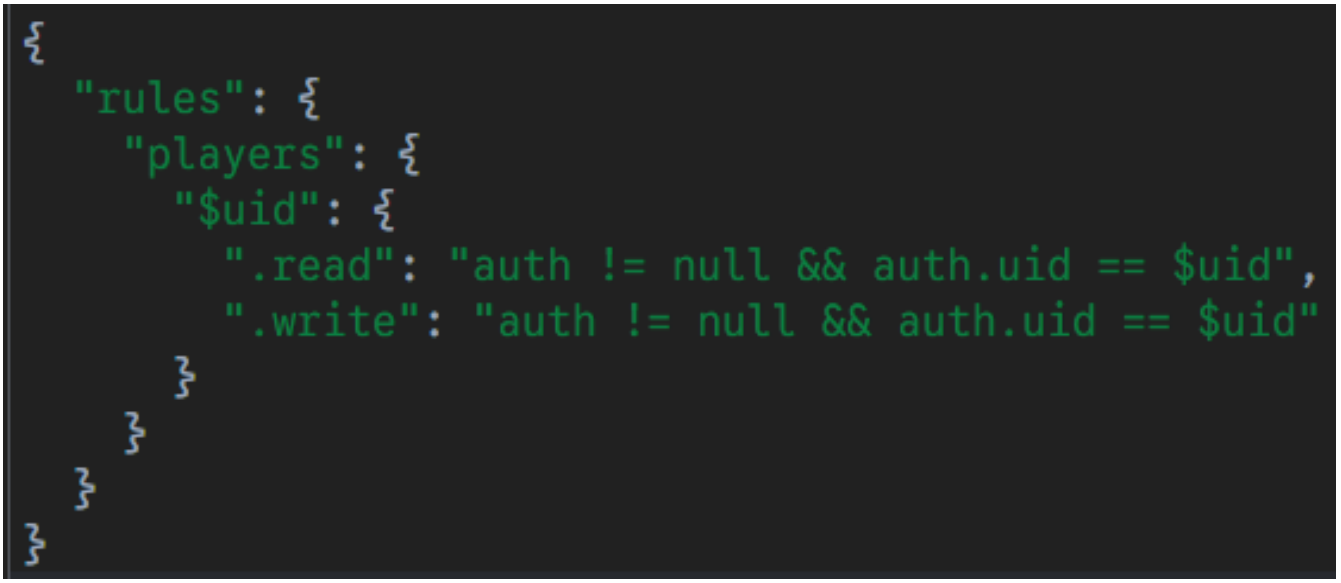


Рис. 3.12. Правила доступу до даних у Firebase

Показані правила означають, що користувач може читати та змінювати лише ті дані, які належать його обліковому запису. Будь-які спроби доступу до даних інших користувачів блокуються на рівні бази даних. Це забезпечує базовий рівень захисту даних і запобігає несанкціонованому доступу до інформації інших гравців.

4 ТЕСТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Особливості тестування мобільної 3D гри

Тестування мобільної 3D гри є важливим етапом розробки, оскільки дозволяє перевірити стабільність роботи застосунку, коректність реалізації ігрової логіки та зручність взаємодії користувача з елементами керування.

Особливу увагу під час тестування було приділено:

– роботі фізичних режимів кулі;

– коректності перемикання між режимами;

– роботі системи зіткнень;

– стабільності завантаження рівнів;

– коректності роботи користувацького інтерфейсу;

– роботі системи збереження прогресу через Firebase;

– адаптації гри до мобільного керування.

Для перевірки функціональності гри використовувалось функціональне мануальне тестування. Тестування проводилось шляхом виконання основних ігрових сценаріїв та перевірки реакції системи на дії користувача.

4.2 Тестові сценарії гри

Тестування гри виконувалось за допомогою ручного функціонального тестування із застосуванням методу "чорної скриньки". Основна увага приділялась перевірці коректності роботи ігрових механік, інтерфейсу користувача та системи взаємодії об’єктів. Тестування проводилось шляхом виконання типових дій користувача в грі та перевірки відповідності фактичного результату очікуваному.

Для перевірки працездатності основних функцій було сформовано набір тестових сценаріїв, що охоплюють ключові ігрові механіки, взаємодію з інтерфейсом, систему рівнів та систему збереження прогресу.

Тестові сценарії та результати їх виконання наведено в таблиці 4.1.

Таблиця 4.1.

Тестові сценарії гри

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Переміщення джойстика керування	Куля рухається у відповідному напрямку	Відповідає очікуваному
2	Перемикання у швидкий режим	Швидкість кулі збільшується	Відповідає очікуваному
3	Натискання кнопки стрибка	Куля виконує стрибок	Відповідає очікуваному
4	Перемикання у важкий режим	Куля рухається повільніше та може штовхати об’єкти	Відповідає очікуваному
5	Перемикання у магнітний режим	Куля притягується до магнітних поверхонь	Відповідає очікуваному
6	Зіткнення з перешкодою	Відображається екран поразки	Відповідає очікуваному
7	Зіткнення з перешкодою після досягнення чекпоінту	Відображається екран поразки з можливістю відновлення на чекпоінті за одне життя	Відповідає очікуваному

Продовження таблиці 4.1.

Тестові сценарії гри

8	Відновлення після поразки	Гравець продовжує проходження з останнього чекпоінту	Відповідає очікуваному
9	Досягнення фінішної зони	Відображається екран перемоги	Відповідає очікуваному
10	Переміщення повзунка Music у меню Settings	Рівень гучності музики змінюється	Відповідає очікуваному
11	Переміщення повзунка Sound у меню Settings	Рівень гучності звуків змінюється	Відповідає очікуваному
12	Натискання на перемикач Shadows у меню Settings	Тіні у грі вмикаються або вимикаються	Відповідає очікуваному
13	Натискання на перемикач Effects у меню Settings	Візуальні ефекти вмикаються або вимикаються	Відповідає очікуваному
14	Натискання кнопки Pause	Відкривається меню паузи, та зупиняється час у грі	Відповідає очікуваному
15	Натискання кнопки Restart у паузі	Поточний рівень перезапускається	Відповідає очікуваному
16	Натискання кнопки Menu у паузі	Виконується перехід у головне меню	Відповідає очікуваному

Продовження таблиці 4.1.

Тестові сценарії гри

17	Вибір рівня у меню	Завантажується обраний рівень	Відповідає очікуваному
18	Авторизація користувача	Виконується вхід через Firebase Authentication	Відповідає очікуваному
19	Повторний запуск гри	Прогрес користувача завантажується з БД	Відповідає очікуваному

4.3 Результати тестування

У результаті тестування було підтверджено коректність роботи гри без збоїв. Для тестування гри, було використано 2 мобільні пристрої на Android. Це допомогло переконатися, що інтерфейс гри адаптований під різні розміри екранів, а елементи інтерфейсу реагують на дотики користувача. Не було виявлено ніяких серйозних багів, зависань або вильотів з гри.

Перевірка показала, що всі частини гри працюють коректно:

– Рух об’єкту: куля чітко котиться за направленням джойстику та стрибає при натисканні на кнопку стрибку;

– Фізичні режими: швидкий, важкий та магнітні режими перемикаються без затримок і відповідають своїм характеристикам;

– Інтерфейс: всі кнопки, повзунки гучності та перемикачі в налаштуваннях працюють без помилок і виконують свої функції;

– Збереження та завантаження даних: система завантажує та зберігає ігровий процес користувача без втрати інформації.

Завдяки оптимізації 3D-об’єктів, матеріалів та коду гра працює без затримок. Фонові процеси, такі як запис та завантаження даних користувача у базу даних, не викликає зависань чи падіння продуктивності гри.

ВИСНОВКИ

У результаті виконання кваліфікаційній роботі було проаналізовано аналоги мобільних казуальних ігор і на основі аналізу розроблено та протестовано мобільну казуальну 3D-гру з динамічною зміною фізичних властивостей об'єкта. Зміна фізичних властивостей об’єкта у грі додає інтерактивності у гру, що підвищує зацікавленість користувача. На основі проведеної роботи можна сформувати такі підсумки:

1. Проведено комплексний аналіз казуальних мобільних 3D ігор, у ході якого досліджено специфіку використання фізичних механік у геймдеві. На основі аналізу існуючих аналогів сформовано технічні вимоги до реалізації власного програмного продукту.

2. Обґрунтовано доцільність використання ігрового рушія Unity. Доведено, що поєднання вбудованої фізичної системи PhysX та компонентної архітектури дозволяє ефективно моделювати взаємодію об’єктів, забезпечуючи при цьому високу продуктивність застосунку на мобільних пристроях.

3. Сформовано набір програмних інструментів для реалізації проекту. Для написання ігрової логіки обрано мову програмування C#, 3D-моделювання виконано у середовищі Blender, проектування інтерфейсу — у Figma, а інтеграцію хмарних сервісів та аутентифікації користувачів забезпечено платформою Firebase.

4. Спроектовано та реалізовано архітектуру гри, яка базується на поділі логіки за сценами та використанням модульних компонентів. Такий підхід забезпечив високу ступінь ізоляції систем, що дозволяє масштабувати проект та додавати нові рівні без втручання в основну структуру коду.

5. Розроблено та інтегровано ключові ігрові механіки, що ґрунтуються на динамічній зміні фізичних властивостей головного об’єкта. Реалізовані режими руху забезпечують варіативність ігрового процесу та адаптацію проходження рівнів до умов оточення.

6. Створено повноцінну систему рівнів та користувацький інтерфейс (HUD). Реалізовано функціональні екрани (меню, пауза, результати проходження) та налаштовано надійний механізм синхронізації прогресу гравця за допомогою хмарних сервісів Firebase.

7. Проведено функціональне тестування розробленого застосунку методом «чорної скриньки». Результати перевірки підтвердили стабільність роботи всіх ігрових механік, коректність обробки зіткнень, ергономічність керування та безпечність збереження даних користувача, що свідчить про готовність продукту до експлуатації.

Робота пройшла апробацію на двох наукових конференціях:

1. Тихонченко І.О., Коваленко Д.С. Розробка Мобільнаї 3D гри казуального жанру зі зміною фізичних властивостей об’єкта. Матеріали ІІ Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.414-416.

2. Тихонченко І.О. Забезпечення безпеки інформації у мобільній 3D грі зі зміною фізичних властивостей об’єкта. Матеріали Всеукраїнської науково-технічної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, ДУІКТ. Збірник тез. К.: ДУІКТ, 2026. С.77-78.

ПЕРЕЛІК ПОСИЛАНЬ

1. Дібрівний О.А., Гребенюк В.В. Вступ до об’єктно орієнтованого програмування C#: Навчальний посібник. – К.: Державний університет телекомунікацій, 2018, 190 с. – Дата звернення: 01.04.2026

2. Казуальні ігри: що це, якими вони бувають і як розвивається жанр. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://vokigames.com/ua/kazualni-igri-shho-cze-yakimi-voni-buvayut-i-yak-rozvivayetsya-zhanr/ – Дата звернення: 01.04.2026

3. Rolling Sky. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://rolling-sky.com – Дата звернення: 02.04.2026

4. Going Balls. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://going-balls.com – Дата звернення: 02.04.2026

5. Switchball HD [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://store.steampowered.com/app/1588760/Switchball_HD/ – Дата звернення: 02.04.2026

6. Firebase. Add Firebase to your Unity project. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://firebase.google.com/docs/unity/setup – Дата звернення: 03.04.2026

7. Unity Technologies. Physics in Unity (PhysX integration). [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://docs.unity3d.com/Manual/PhysicsSection.html – Дата звернення: 05.04.2026.

8. Epic Games. Unreal Engine Documentation – Physics (Chaos). [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/physics-in-unreal-engine – Дата звернення: 05.04.2026.

9. Godot Engine Documentation. 3D and Physics Overview. [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://docs.godotengine.org/en/stable/tutorials/physics/index.html – Дата звернення: 05.04.2026.

10. Your Full Guide to Unity Performance Optimization. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://game-ace.com/blog/unity-performance-optimization/ – Дата звернення: 07.04.2026

11. Optimize your mobile game performance: Tips on profiling, memory, and code architecture from Unity's top engineers. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://unity.com/blog/games/optimize-your-mobile-game-performance-tips – Дата звернення: 12.04.2026

12. Unity Technologies. Graphics performance and profiling. Unity User Manual, 2019. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://docs.unity3d.com/2019.4/Documentation/Manual/OptimizingGraphicsPerformance. – Дата звернення: 15.04.2026

13. Level Design Basics. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://gamedev.dou.ua/blogs/level-design-basics-part-1/?hl=en – Дата звернення: 18.04.2026

14. Unity. What is Unity? [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://learn.unity.com/tutorial/what-is-unity?version=undefined – Дата звернення: 20.04.2026

15. Microsoft. Visual Studio Code Documentation. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://code.visualstudio.com – Дата звернення: 20.04.2026

16. Figma. Figma Help Center. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma – Дата звернення: 20.04.2026

17. Microsoft. C# Documentation. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp – Дата звернення: 20.04.2026

18. Blender Manual. About Blender [Електронний ресурс]: [Веб-сайт] – Режим доступу:

https://docs.blender.org/manual/en/latest/getting_started/about/index.html – Дата звернення: 20.04.2026

19. Audacity Manual. About Audacity [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://www.audacityteam.org/faq/ – Дата звернення: 20.04.2026

20. Google. Firebase Documentation. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://firebase.google.com/docs – Дата звернення: 20.04.2026

21. Meet Google Play's target API level requirement. [Електронний ресурс]: [Веб-сайт] – Режим доступу: https://developer.android.com/google/play/requirements/target-sdk – Дата звернення: 21.04.2026

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільної 3D гри казуального жанру зі зміною фізичних властивостей об’єкта

Виконав студент 4 курсу
групи ПД-41
Тихонченко Ігор Олександрович
Керівник роботи
ст. викл. каф. ІПЗ, доктор філософії (PhD)
Коваленко Данило Сергійович

Київ – 2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення інтерактивності мобільної 3D гри казуального жанру, шляхом впровадження динамічної зміни фізичних властивостей об’єкта.

Об’єкт дослідження: процес створення мобільної 3D-гри зі зміною фізичних властивостей об’єкта.

Предмет дослідження: методи реалізації ігрових механік зі зміною фізичних властивостей об’єкта мовою програмування C#, та оптимізації гри для платформи Android.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз існуючих аналогів мобільних 3D ігор казуального жанру зі зміною фізичних властивостей об’єкта.
2. Визначити функціональні та нефункціональні вимоги до гри.
3. Проаналізувати програмні засоби та технології для розробки мобільної 3D гри.
4. Обрати програмні засоби та технології для розробки мобільної 3D гри.
5. Спроєктувати архітектуру мобільної 3D гри та структуру її основних модулів.
6. Реалізувати ігрові механіки, систему рівнів, користувацький інтерфейс та систему збереження даних.
7. Провести тестування мобільної 3D гри на коректність роботи основних механік, інтерфейсу та системи збереження прогресу.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Super Monkey Ball	Rolling Sky	Going Balls	Switch Ball	**Roll&Morph**
Реалістична 3D фізика	+	+	+	+	+
Зміна фізичних властивостей об’єкта	-	-	-	+	+
Динамічна взаємодія з перешкодами	+	+	+	+	+
Сумісність з ОС Android	+	+	+	-	+
Оптимізація під слабкі пристрої	-	-	-	-	+

4

КОНЦЕПТ ГРИ

Після старту рівня гравець керує кулькою в тривимірному просторі та проходить маршрут, уникаючи перешкод і небезпечних зон.

Ігровий процес базується на зміні фізичних властивостей об’єкта, що впливає на швидкість руху, масу та взаємодію з оточенням.

Доступні три фізичні режими: швидкий (підвищена швидкість), важкий (зміна маси та вплив на об’єкти), магнітний (притягування до спеціальних поверхонь).

Рівень завершується після досягнення фінішної зони або у разі зіткнення граця з перешкодою.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Керування кулькою як основним ігровим об’єктом.
2. Зміна фізичних властивостей об’єкта під час гри.
3. Забезпечення рівнів з перешкодами різної складності.
4. Взаємодія об’єкта з перешкодами та елементами рівня.
5. Збір бонусів та проходження рівня за певний час.
6. Збереження та завантаження прогресу користувача.

Нефункціональні вимоги:

1. Сумісність з ОС Android.
2. Стабільна продуктивність без затримок (45–60 к/с).
3. Мінімальні апаратні вимоги: Процесор: Snapdragon 410 (або еквівалент), RAM 2 ГБ.
4. Рекомендовані апаратні вимоги: Процесор: Snapdragon 660 (або еквівалент), RAM 4 ГБ.
5. Адаптивний інтерфейс для різних розмірів екранів.
6. Оптимізація гри за рахунок спрощення матеріалів і повторного використання матеріалів (batching).

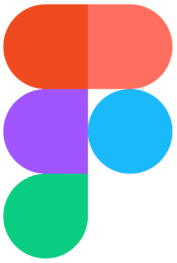
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Unity

VS Code

Firebase

Blender

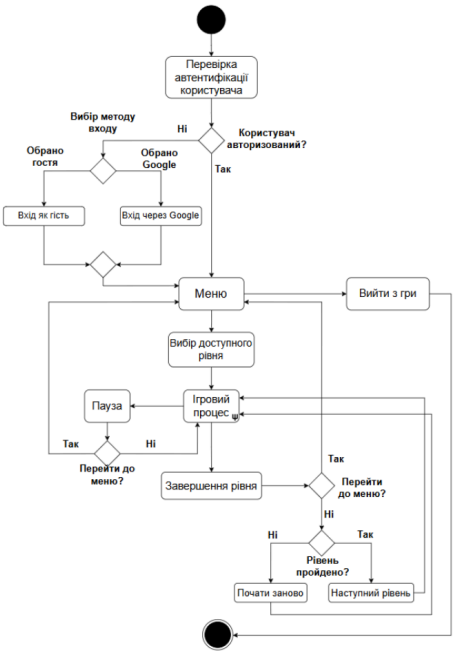


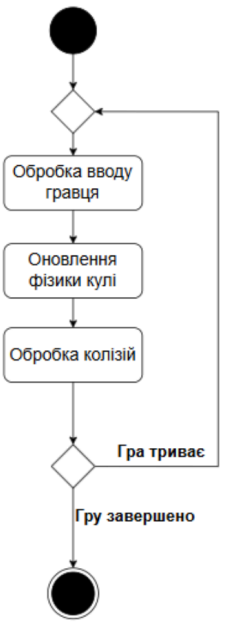
Figma

Audacity

7

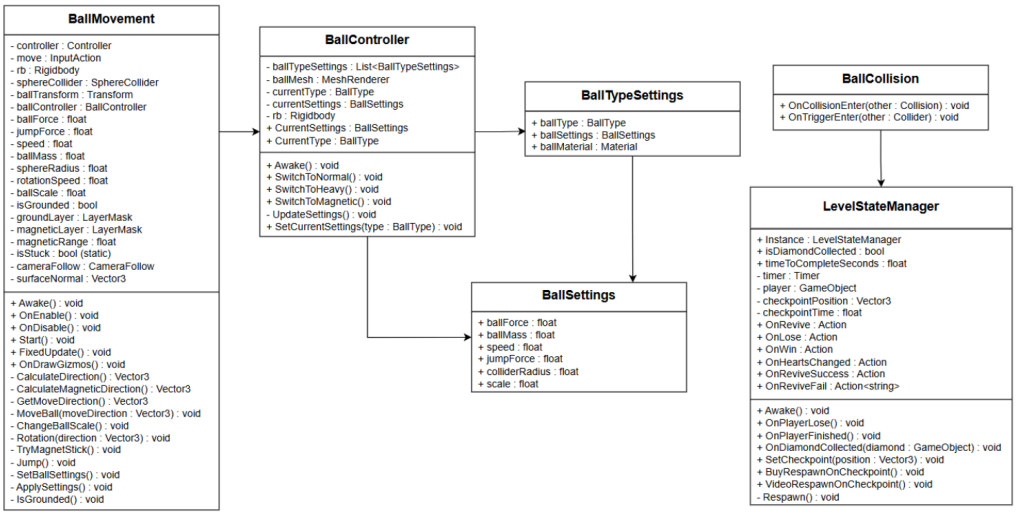
ДІАГРАМА ДІЯЛЬНОСТІ





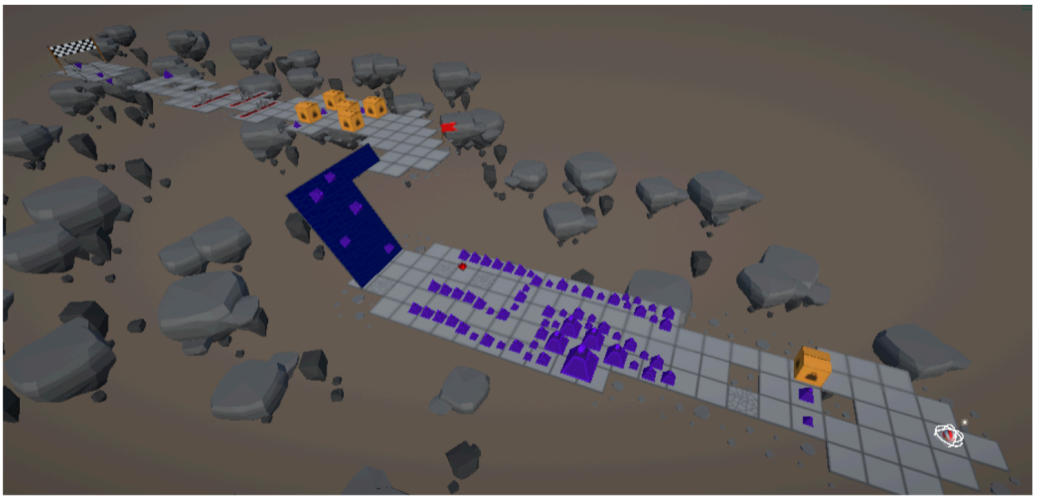
8

ДІАГРАМА КЛАСІВ КЕРУВАННЯ ОБ’ЄКТОМ ТА СТАНОМ РІВНЯ



9

ЕКРАННІ ФОРМИ



Вигляд одного з рівнів в Unity

10

ВІДЕО РОБОТИ ЗАСТОСУНКУ

11

ТЕСТУВАННЯ

Було протестовано:

1. Роботу керування кулькою та стрибка;
2. Перемикання фізичних режимів;
3. Систему зіткнень;
4. Завантаження та проходження рівнів;
5. Роботу користувацького інтерфейсу;
6. Систему авторизації;
7. Збереження та завантаження прогресу користувача.

Результати тестування:

1. Усі тестові сценарії виконані успішно;
2. Критичних помилок та збоїв не виявлено;
3. Інтерфейс коректно адаптується під різні розміри екранів;
4. Гра стабільно працює на Android-пристроях без зависань та вильотів;
5. Система збереження даних працює коректно без втрати прогресу.

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Тихонченко І.О., Коваленко Д.С. Розробка Мобільнаї 3D гри казуального жанру зі зміною фізичних властивостей об’єкта. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.414-416.
2. Тихонченко І.О. Забезпечення безпеки інформації у мобільній 3D грі зі зміною фізичних властивостей об’єкта. Матеріали Всеукраїнської науково-технічної конференції «Цифрова трансформація кібербезпеки», 29 квітня 2026 року, Київ, ДУІКТ. Збірник тез. К.: ДУІКТ, 2026. С.77-78.

ВИСНОВКИ

1. Проведено аналіз існуючих аналогів мобільних 3D ігор казуального жанру зі зміною фізичних властивостей об’єкта та визначено основні функціональні та нефункціональні вимоги до мобільної гри.
2. Обрано програмні засоби та технології розробки мобільної 3D гри, зокрема Unity, C#, Blender, Firebase, Figma та Audacity, що забезпечили реалізацію основних ігрових механік та підтримку мобільної платформи Android.
3. Спроектовано архітектуру мобільної 3D гри та структуру її функціональних модулів, включаючи систему рівнів, користувацький інтерфейс і систему збереження даних.
4. Реалізовано основні ігрові механіки, засновані на зміні фізичних властивостей об’єкта, зокрема швидкий, важкий та магнітний режими взаємодії з ігровим середовищем.
5. Проведено тестування мобільної 3D гри, результати якого підтвердили коректність роботи основних механік, інтерфейсу користувача та системи збереження прогресу.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
// BallMovement.cs
using System;
using UnityEngine;
using UnityEngine.InputSystem;

public class BallMovement : MonoBehaviour
{
    private Controller controller;
    private InputAction move;
    private Rigidbody rb;
    private SphereCollider sphereCollider;

    [SerializeField] private Transform ballTransform;
    [SerializeField] private BallController ballController;

    [Header("BallSettings")]
    [SerializeField] private float ballForce = 5f;
    [SerializeField] private float jumpForce = 5f;
    [SerializeField] private float speed = 10f;
    [SerializeField] private float ballMass = 1f;
    [SerializeField] private float sphereRadius = 0.5f;
    [SerializeField] private float rotationSpeed = 5f;
    [SerializeField] private float ballScale = 1f;

    private bool isGrounded = true;
    [SerializeField] private LayerMask groundLayer;

    [Header("Magnetic")]
    [SerializeField] private LayerMask magneticLayer;
    [SerializeField] private float magneticRange = 1.5f;

    [HideInInspector] public static bool isStuck = false;

    [Header("Camera")]
    [SerializeField] private CameraFollow cameraFollow;
    private Vector3 surfaceNormal = Vector3.up;

    private void Awake()
    {
        controller = new Controller();
        rb = GetComponent<Rigidbody>();
        sphereCollider = GetComponent<SphereCollider>();
    }

    private void OnEnable()
    {
        controller.Enable();
        move = controller.Player.Move;
        controller.Player.Jump.performed += OnJumpPerformed;
            controller.Player.SwitchToNormalMode.performed +=
OnSwitchNormalPerformed;
             controller.Player.SwitchToHeavyMode.performed +=
OnSwitchHeavyPerformed;
          controller.Player.SwitchToMagneticMode.performed +=
OnSwitchMagneticPerformed;
    }

    private void OnDisable()
    {
        controller.Player.Jump.performed -= OnJumpPerformed;
            controller.Player.SwitchToNormalMode.performed -=
OnSwitchNormalPerformed;
             controller.Player.SwitchToHeavyMode.performed -=
OnSwitchHeavyPerformed;
          controller.Player.SwitchToMagneticMode.performed -=
OnSwitchMagneticPerformed;
        controller.Disable();
    }

    private void Start()
    {
        SetBallSettings();
    }


    private void FixedUpdate()
    {
            Debug.DrawRay(rb.worldCenterOfMass, -transform.up,
Color.green);

        Vector3 moveDirection = GetMoveDirection();

        IsGrounded();
        MoveBall(moveDirection);
```

```
    ChangeBallScale();
  }

  private void OnDrawGizmos()
  {
                Gizmos.DrawWireSphere(transform.position,
magneticRange);
  }

  //Ball direction
  private Vector3 CalculateDirection()
  {
    Vector2 moveInput = move.ReadValue<Vector2>();
    moveInput = Vector2.ClampMagnitude(moveInput, 1f);

    Vector3 camForward = cameraFollow.transform.forward;
    Vector3 camRight = cameraFollow.transform.right;

    camForward.y = 0;
    camRight.y = 0;

    camForward.Normalize();
    camRight.Normalize();

        Vector3 moveDir = camForward* moveInput.y +
camRight* moveInput.x;

    if (!isGrounded)
      return moveDir/2.3f;

    return moveDir;
  }

  //Magnetic Ball direction
  private Vector3 CalculateMagneticDirection()
  {
    Vector2 moveInput = move.ReadValue<Vector2>();
    moveInput = Vector2.ClampMagnitude(moveInput, 1f);

    Vector3 camRight = cameraFollow.transform.right;

    camRight.y = 0;
    camRight.Normalize();

            Vector3 forward = Vector3.Cross(camRight,
surfaceNormal).normalized;

            Vector3 right = Vector3.Cross(surfaceNormal,
forward).normalized;

     Vector3 moveDir = forward * moveInput.y + right *
moveInput.x;

    if (moveDir.sqrMagnitude > 1f)
    {
      moveDir.Normalize();
    }

    cameraFollow.SetRotationSide(surfaceNormal.x);

    return moveDir;
  }

  private Vector3 GetMoveDirection()
  {
    Vector3 moveDirection;
    surfaceNormal = Vector3.up;

    if (ballController.CurrentType == BallType.Magnetic)
    {
      TryMagnetStick();

                ballController.SetCurrentSettings(isStuck ?
BallType.Magnetic : BallType.MagneticGrounded);

      SetBallSettings();
                        moveDirection = isStuck ?
CalculateMagneticDirection() : CalculateDirection();

    }
    else
    {
      moveDirection = CalculateDirection();
    }

    return moveDirection;
  }

  private void MoveBall(Vector3 moveDirection)
  {
    Vector3 targetVelocity = moveDirection * speed;

    if (!isStuck)
      targetVelocity.y = rb.velocity.y;
```

```
        rb.velocity = Vector3.Lerp(rb.velocity, targetVelocity, ballForce * Time.fixedDeltaTime);

    if (rb.velocity.sqrMagnitude < 0.01f)
    {
      rb.velocity = Vector3.zero;
    }

    Rotation(rb.velocity);
  }

  private void ChangeBallScale()
  {
                                    float newScale = Mathf.MoveTowards(ballTransform.localScale.x, ballScale, Time.fixedDeltaTime * 3f);
        ballTransform.localScale = new(newScale, newScale, newScale);
  }

  private void Rotation(Vector3 direction)
  {
    if (direction.sqrMagnitude < 0.01f)
      return;

    if (surfaceNormal == Vector3.up)
                                    transform.rotation = Quaternion.Euler(transform.eulerAngles.x,transform.eulerAngles.y,0);

                                Quaternion targetRotation = Quaternion.LookRotation(direction, surfaceNormal);
      transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, rotationSpeed * Time.fixedDeltaTime);

                float distance = rb.velocity.magnitude * Time.fixedDeltaTime;
            float angle = ((distance / sphereRadius) ) * Mathf.Rad2Deg;

    ballTransform.Rotate(Vector3.right, angle, Space.Self);

  }

  private void TryMagnetStick()
  {
                                                Collider[] hits = Physics.OverlapSphere(rb.worldCenterOfMass, magneticRange, magneticLayer);
    float closestDistance = Mathf.Infinity;
    Vector3 closestNormal = Vector3.up;
    Vector3 contactPoint = Vector3.zero;
    bool found = false;

    foreach (var col in hits)
    {
                                        Vector3 closest = col.ClosestPoint(rb.worldCenterOfMass);
      Vector3 dir = rb.worldCenterOfMass - closest;
      float distance = dir.magnitude;

      if (distance < closestDistance)
      {
        RaycastHit hit;
                if (Physics.Raycast(rb.worldCenterOfMass, -dir.normalized, out hit, magneticRange + 0.1f, magneticLayer))
        {
          closestDistance = distance;
          closestNormal = hit.normal;
          contactPoint = hit.point;
          found = true;
        }
      }
    }

    if (found)
    {
      rb.useGravity = false;
      rb.velocity = Vector3.zero;
      rb.angularVelocity = Vector3.zero;

                                Quaternion targetRotation = Quaternion.FromToRotation(transform.up, closestNormal) * transform.rotation;
                                    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, 20f * Time.fixedDeltaTime);

            rb.MovePosition(contactPoint + closestNormal * sphereRadius * ballTransform.localScale.x);

      surfaceNormal = closestNormal;
```

```
            isStuck = true;
        }
        else
        {
            rb.useGravity = true;
            isStuck = false;
        }
    }


    private void Jump()
    {
        if (isGrounded)
                        rb.AddForce((Vector3.up) * jumpForce,
ForceMode.Impulse);
    }

    private void SetBallSettings()
    {
        ballForce = ballController.CurrentSettings.ballForce;
        jumpForce = ballController.CurrentSettings.jumpForce;
        speed = ballController.CurrentSettings.speed;
        ballMass = ballController.CurrentSettings.ballMass;
                                            sphereRadius =
ballController.CurrentSettings.colliderRadius;
        ballScale = ballController.CurrentSettings.scale;

        ApplySettings();
    }

    private void ApplySettings()
    {
        sphereCollider.radius = sphereRadius;
        rb.mass = ballMass;
    }

    private void IsGrounded()
    {
            isGrounded = Physics.Raycast(transform.position,
Vector3.down, (0.1f + sphereRadius), groundLayer);
    }

                                            private void
OnJumpPerformed(InputAction.CallbackContext context)
    {
        Jump();
    }
                                                    private void
OnSwitchNormalPerformed(InputAction.CallbackContext
context)
    {
        isStuck = false;
        ballController.SwitchToNormal();
        SetBallSettings();
    }
                                                    private void
OnSwitchHeavyPerformed(InputAction.CallbackContext
context)
    {
        isStuck = false;
        ballController.SwitchToHeavy();
        SetBallSettings();
    }
                                                    private void
OnSwitchMagneticPerformed(InputAction.CallbackContext
context)
    {
        ballController.SwitchToMagnetic();
        SetBallSettings();
    }
}

// BallController.cs
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallController : MonoBehaviour
{
            [SerializeField] private List<BallTypeSettings>
ballTypeSettings;
    [SerializeField] private MeshRenderer ballMesh;
    private BallType currentType;
    private BallSettings currentSettings;

    public BallSettings CurrentSettings => currentSettings;
    public BallType CurrentType => currentType;

    private Rigidbody rb;
    private void Awake()
    {
        rb = GetComponent<Rigidbody>();
```

```
    currentType = BallType.Normal;
    UpdateSettings();
}


public void SwitchToNormal()
{
    currentType = BallType.Normal;
    UpdateSettings();
}

public void SwitchToHeavy()
{
    currentType = BallType.Heavy;
    UpdateSettings();
}

public void SwitchToMagnetic()
{
    currentType = BallType.Magnetic;
    UpdateSettings();
}

private void UpdateSettings()
{
    rb.useGravity = true;
    foreach (var item in ballTypeSettings)
    {
        if (item.ballType == currentType)
        {
            if (currentType == BallType.Magnetic)
                rb.useGravity = false;

            ballMesh.material = item.ballMaterial;
            currentSettings = item.ballSettings;
            Debug.Log(currentType);
            break;
        }
    }
}

public void SetCurrentSettings(BallType type)
{
    foreach (var item in ballTypeSettings)
    {
        if (item.ballType == type)
        {
            currentSettings = item.ballSettings;
            break;
        }
    }
}

// BallCollision.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallCollision : MonoBehaviour
{
    private void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.CompareTag("Trap"))
        {
            LevelStateManager.Instance.OnPlayerLose();
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Diamond"))
        {

LevelStateManager.Instance.OnDiamondCollected(other.game
Object);
        }
        else if (other.gameObject.CompareTag("Finish"))
        {
            LevelStateManager.Instance.OnPlayerFinished();
        }
        else if (other.gameObject.CompareTag("Trap"))
        {
            LevelStateManager.Instance.OnPlayerLose();
        }
    }
}

// BallType.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public enum BallType
{
    Normal,
    Heavy,
    Magnetic,
    MagneticGrounded
}

// BallSettings.cs
using System;
using UnityEngine;

[CreateAssetMenu(fileName = "BallSettings")]
public class BallSettings : ScriptableObject
{
    public float ballForce;
    public float ballMass;
    public float speed;
    public float jumpForce;
    public float colliderRadius;
    public float scale;
}

// BallTypeSettings.cs
using System;
using UnityEngine;

[Serializable]
public class BallTypeSettings
{
    public BallType ballType;
    public BallSettings ballSettings;
    public Material ballMaterial;
}
```