МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка кросплатформного ігрового застосунку на платформі Unity

Виконав студент групи **К-21і**
спеціальності 122 Комп'ютерні науки
Павлій Максим Андрійович

Керівник **ст. викладач**
Вохменцева Т.Б.

Консультант **д.ф., доцент**
Бучинська І.В.

Рецензент **заступник директора - начальник відділу Департаменту інформації та цифрових рішень Одеської міської ради**
Корчемний П.А.

Одеса 2023

ЗМІСТ

ТЕРМІНИ, СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Ігровий двигун (англ. game engine) – це комплексне програмне забезпечення, яке використовується для розробки, створення і відтворення комп'ютерних ігор. Він надає розробникам необхідні інструменти, функціонал і ресурси для створення графіки, фізики, звуку, штучного інтелекту, управління сценою, анімацій, взаємодії з користувачем та іншого функціоналу.

Інтегроване середовище розробки (англ. ICP або IDE) – це програмне забезпечення, яке об'єднує різні інструменти для розробки програмного коду в одному зручному інтерфейсі. Воно надає розробникам зручні і ефективні інструменти підсвічування синтаксису, автодоповнення коду, налагоджування та інших корисних функцій, що полегшують процес розробки.

Міжплатформовий застосунок – це програмний продукт або додаток, який розроблений таким чином, щоб працювати на різних платформах, таких як різні операційні системи, пристрої або середовища.

Рендерінг – це процес створення зображення або відео з вхідних даних за допомогою комп'ютерної графіки. У контексті комп'ютерних ігор, анімації, візуалізації або комп'ютерної графіки загалом, рендерінг означає обчислення та відображення графічних об'єктів, освітлення, тіней, текстур та інших візуальних ефектів.

Рефакторинг – це процес вдосконалення програмного коду, при якому вносяться зміни в його структуру, організацію та якість, з метою полегшення розуміння, підтримки та розширення програмного забезпечення без зміни його зовнішньої поведінки.

Фреймворк – це набір програмних компонентів, інструментів і бібліотек, які надають розробникам загальну структуру та функціональність для створення програмного забезпечення. Він визначає каркас, на якому можна будувати програми і додатки, надаючи готові рішення для розповсюджених завдань та проблем.

Плагін – це програмний модуль або компонент, який розширює функціональність основної програми або додатка. Він дозволяє додавати нові функції, функціональність або можливості до вихідного програмного забезпечення без необхідності змінювати саму основну програму.

ВАР – віртуальна апаратна реалізація.

ГП – графічний процесор.

ЗІ – загальний інтерфейс.

КП – крос-платформність.

МВМ – міжплатформовий віртуальний механізм.

МКЗ – мультимедійний крос-платформний застосунок.

СК – спільний код.

СПЗ – середовище програмування захищеної зони.

УК – універсальні компоненти.

AI – artificial intelligence.

API – application programming interface.

AR – augmented reality.

IDE – integrated development environment.

MVC – Model-View-Controller.

OS – operating system.

SDK – software development kit.

UI – user interface.

VR – virtual reality.

VS – visual studio.

ВСТУП

У сучасному цифровому світі мобільні пристрої, зокрема смартфони та планшети, стали невід'ємною частиною нашого повсякденного життя. Ці пристрої пропонують унікальні можливості для розваг, зокрема для ігор, які стали одним із найпопулярніших видів розваг для мільйонів користувачів по всьому світу. Однак, з розширенням різноманітності операційних систем та платформ мобільних пристроїв, зокрема Android і iOS, виникають виклики для розробників ігор, які прагнуть досягти максимальної аудиторії та комерційного успіху.

У зв'язку з розширенням ринку мобільних ігор та зростанням конкуренції, створення мобільних крос-платформних ігор стає надзвичайно актуальною темою. Крос-платформність означає розробку ігор, які можуть працювати на різних операційних системах та платформах без необхідності створення окремих версій гри для кожної з них. Це дозволяє розробникам досягти максимального охоплення аудиторії, залучаючи користувачів на різних мобільних платформах.

Метою кваліфікаційної роботи бакалавра є розробка крос-платформного мобільного ігрового застосунку з інтуїтивно зрозумілим та зручним інтерфейсом.

Актуальність створення мобільних крос-платформних ігор проявляється у кількох аспектах. Це дозволяє розробникам скоротити час та ресурси, які зазвичай потрібні для розробки та підтримки різних версій гри під різні платформи. Економія часу та зусиль забезпечує більш ефективний процес розробки та дозволяє зосередитись на якості та інноваціях ігрового контенту.

Також створення мобільних крос-платформних ігор відкриває широкі можливості для монетизації та комерційного успіху. Забезпечення доступності гри для користувачів на різних платформах дозволяє залучити більше гравців та розширити потенційну аудиторію. Це може призвести до збільшення доходів від реклами, мікротранзакцій, підписок або продажу самої гри.

Ціллю роботи є вивчення процесу створення крос-платформних мобільних ігор з використанням середовища розробки Unity та виявлення основних переваг і складнощів цього підходу. Мобільні крос-платформні ігри мають широкі сфери застосування, зокрема в ігровій індустрії, рекламі, освіті та тренінгах. У рамках роботи будуть розглянуті наступні задачі:

- вивчення основних принципів та концепцій крос-платформної розробки ігор;
- аналіз можливостей середовища розробки Unity для створення крос-платформних мобільних ігор;
- розробка прототипу крос-платформної мобільної гри з використанням Unity;
- тестування та оптимізація крос-платформного прототипу гри;
- виявлення переваг та складнощів крос-платформної розробки ігор.

Дана робота має взаємозв'язок з іншими дослідженнями та розробками в галузі крос-платформної розробки ігор. Вона базується на практичних розв'язаннях, здобутих вже в цій галузі, а також враховує світові тенденції і досягнення.

Дана кваліфікаційна робота бакалавра, складається з 56 сторінок, 11 рисунків та 9 джерел посилання.

1 АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ ІГРОВОГО ЗАСТОСУНКА

1.1 Характеристика гри Rolling Sky

"Rolling Sky" – це аркадна гра, в якій гравець керує кулькою і намагається провести її через різні траси та перешкоди. Головна особливість гри полягає в тому, що траси являють собою плаваючі платформи, по яким кулька рухається. Мета гри – подолати рівень, уникаючи перешкод, таких як прірви і вузькі проходи. Основні особливості гри "Rolling Sky" включають естетично привабливі рівні з різноманітними дизайнами, динамічну і захоплюючу ігрову механіку, а також можливість змагатися з іншими гравцями в режимі онлайн.

Однак "Rolling Sky" має наступні недоліки:

- одноманітність ігрового процесу (В "Rolling Sky" гравцеві пропонується долати різні рівні, але ігровий процес залишається приблизно однаковим на всіх рівнях. Це може призвести до відчуття монотонності та повторюваності, особливо при тривалій грі);
- обмежені варіанти управління (у "Rolling Sky" ігровий процес зводиться до нахилу пристрою або скидання по екрану для управління кулькою. У порівнянні з "Superball", де доступний контроль за допомогою джойстика, управління "Rolling Sky" може здатися менш точним і обмеженим);
- обмежені варіанти інтерактивності ("Rolling Sky" зосереджена на русі кульки по трасі та уникненні перешкод, але пропонує обмежені варіанти інтерактивності та взаємодії з навколишнім світом. У той же час "Superball" пропонує елементи, такі як прискорення, спеціальні платформи та взаємодія з об'єктами на рівні, що робить ігровий процес різноманітнішим та цікавішим).

В цілому, "Rolling Sky" – це цікава та захоплююча гра, але вона може залишати відчуття обмеженості та відсутності різноманітності в ігровому процесі (рис. 1).

Рисунок 1 – Гра "Rolling Sky"

1.2 Характеристика гри ZigZag

"ZigZag" – це аркадна гра, в якій гравець керує кулькою, намагаючись провести його по трасі, що нескінченно генерується та складається з різнокольорових блоків. Мета гри – подолати найбільшу відстань, повертаючи кульку в потрібний момент, щоб уникнути падіння з траси. Основні особливості гри "ZigZag" включають просту та привабливу графіку, простий та інтуїтивно зрозумілий ігровий процес, а також можливість змагатися з іншими гравцями, прагнучи досягти найвищого результату.

Однак, ZigZag також має деякі недоліки, а саме:

– одноманітність траси (у "ZigZag" траса складається з блоків, що повертаються, і гравцеві пропонується повертати кульку в потрібний момент, щоб залишитися на трасі. Однак згодом траса стає передбачуваною та монотонною, що може викликати відчуття повторюваності та відсутності різноманітності);
– обмеженість інтерактивності (у ZigZag гравцеві пропонується тільки одна дія – поворот кульки вліво або вправо, щоб залишитися на трасі);
– відсутність цілей та прогресу (у "ZigZag" основною метою є подолання відстані на трасі, але гра не пропонує явних цілей чи прогресивної системи досягнень. Це може призвести до відсутності почуття прогресу та мотивації на довгострокову гру).

В цілому, "ZigZag" – це проста та захоплююча гра, але в порівнянні з "Superball" вона може залишати відчуття одноманітності, обмеженості інтерактивності та відсутності цілей (рис. 2).

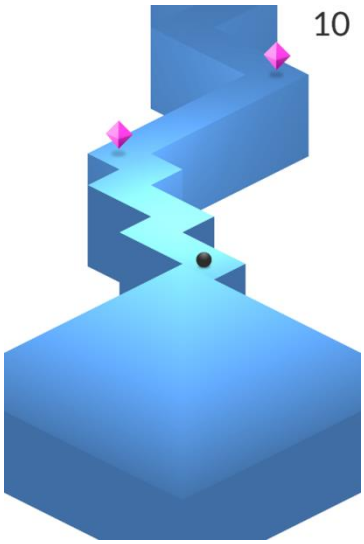

Рисунок 2 – Гра "ZigZag"

1.3 Характеристика гри Helix Jump

"Helix Jump" – це аркадна гра, в якій гравець керує кулькою, яка спускається по спіральній вежі з перешкодами. Мета гри – спуститися наскільки можна вниз, уникаючи перешкод і руйнуючи платформи.

Якщо порівнювати з Superball, то можна зробити наступні висновки. "Superball" пропонує унікальну механіку гри, де кулька рухається трубою і отримує прискорення з кожним проходженням через неї. Це створює динамічний і захоплюючий геймплей, який відрізняється від стандартної спіральної вежі, представленої в Helix Jump. Така механіка додає різноманітність та оригінальність у ігровий процес. "Superball" має велику кількість рівнів, кожен з яких пропонує свої унікальні перешкоди, дизайн та складність. Це дозволяє гравцям насолоджуватися ігровим досвідом та викликами на кожному рівні. Також, "Superball" пропонує все більш складні рівні зі зміною швидкості кульки, що збільшується, при кожному проходженні через трубу. Це створює відчуття прогресу та виклику для гравця, стимулюючи прагнути досягнення нових рекордів та подолання своїх попередніх результатів. У Helix Jump швидкість і складність рівнів не змінюються так динамічно. Управління у "Superball" здійснюється за допомогою джойстика або кнопок, що дозволяє гравцеві більш точно контролювати рух кульки. Це може сприяти більш точним маневрам та реакціям на перешкоди. У "Helix Jump" керування здійснюється шляхом свайпів по екрану, що може бути менш точним і вимагати більше часу для звикання (рис. 3).

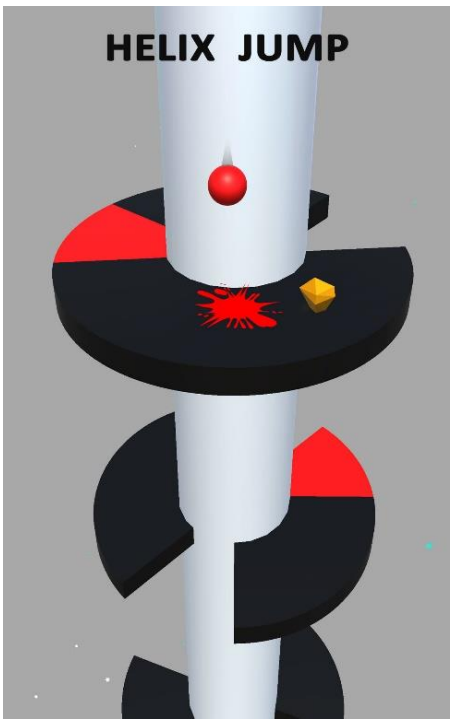

Рисунок 3 – Гра "Helix Jump"

Superball пропонує унікальну механіку, різноманітні рівні та перешкоди, інтуїтивне управління, візуальну привабливість та прогресію складності, що робить її захоплюючою грою. Порівняно з іграми Rolling Sky, ZigZag та Helix Jump, застосунок виділяється своєю оригінальністю та унікальними ігровими особливостями.

2 ОПИС ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ІГРОВОГО ЗАСТОСУНКУ

2.1 Характеристика ігрових двигунів

Unreal Engine – це потужний двигун для розробки ігор, який надає широкі можливості для створення високоякісних ігор з вражаючою графікою. Він підтримує різні платформи та мови програмування, включаючи C++. Unreal Engine пропонує багатий набір інструментів для моделювання, анімації, фізики та інших аспектів розробки ігор.

Godot Engine – це безкоштовний, з відкритим вихідним кодом ігровий движок, який надає інструменти для створення 2D та 3D ігор. Він має простий у використанні інтерфейс і підтримує сценарії мовою програмування GDScript, а також іншими мовами, таких як C# і C++. Godot Engine пропонує безліч функцій та можливостей, включаючи систему управління ресурсами, візуальний редактор та інструменти для створення анімацій.

CryEngine – це ще один потужний двигун для створення ігор, який спочатку був розроблений для гри Crysis. Він пропонує вражаючу графіку та підтримує різні платформи. CryEngine також має розвинений інструментарій для створення фізично реалістичних симуляцій, ефектів світла та інших візуальних елементів.

Середовище розробки ігор Unity є одним із найпопулярніших інструментів для створення ігрового контенту. Unity забезпечує потужні функціональні можливості та інтуїтивно зрозумілий інтерфейс, що дозволяє розробникам реалізовувати свої творчі ідеї та створювати високоякісні ігри. Unity підтримує різні платформи, включаючи мобільні пристрої (Android, iOS), настільні комп'ютери (Windows, macOS, Linux) та ігрові консолі (PlayStation, Xbox, Nintendo Switch). Це дозволяє розробникам досягти максимального охоплення аудиторії та забезпечити доступність гри для широкого кола користувачів. Unity надає інтегроване середовище розробки (IDE), яке поєднує у собі редактор ігрових сцен, систему управління

ресурсами, компоненти для моделювання фізичного та колізійного середовища, анімації, освітлення, звуку та інших важливих аспектів гри. Це дозволяє розробникам створювати складні ігрові світи, налаштовувати поведінку об'єктів, працювати зі спецефектами та досягати високого рівня візуальної якості [1].

Однією з головних переваг Unity є його програмування мовою C#. Це потужна та гнучка мова програмування, що надає розробникам потужні інструменти для розробки ігрової логіки, взаємодії з користувачем, обробки введення та інших аспектів гри. Завдяки C# розробники можуть створювати складні ігрові механіки та управляти поведінкою об'єктів в ігровому світі.

Unity також пропонує безліч додаткових ресурсів та можливостей, які можуть бути використані для покращення процесу розробки. Unity Asset Store пропонує широкий вибір готових асетів, таких як моделі, текстури, анімації, звукові ефекти та інші, які можуть бути інтегровані у гру та значно прискорити розробку. Unity також підтримує різні фреймворки та плагіни, що розширюють функціональність гри, такі як фізичні двигуни, інструменти для штучного інтелекту, системи анімації тощо.

Крім того, Unity має активну спільноту розробників, яка надає підтримку, документацію, навчальні матеріали та безліч онлайн-ресурсів. Це допомагає розробникам отримати відповіді на питання, обмінятися досвідом та покращити свої навички розробки ігор. Unity є потужним інструментом для розробки крос-платформних ігор, що дозволяє створювати ігри високої якості, досягати широкої аудиторії та успішно монетизувати свої проєкти. З використанням Unity, розробники мають усі необхідні інструменти для реалізації ідей та створення захоплюючих ігрових проєктів.

Один із ключових аспектів Unity, який робить його особливо привабливим для розробників ігор, це його мультиплатформність. Unity дозволяє створювати ігри, які можуть бути запущені на різних операційних системах та пристроях без необхідності переписувати код із нуля. Міжплатформовий віртуальний механізм (MBM) уможливлює виконання

одного коду на різних платформах. Це значно спрощує і прискорює процес розробки гри для різних платформ і дозволяє максимально використовувати потенціал кожної платформи [2].

Один з популярних фреймворків для розробки крос-платформних застосунків на Unity – мультимедійний крос-платформний застосунок (МКЗ).

Unity також має інтегрований інструмент для створення та управління анімаціями. Розробники можуть створювати складні анімаційні переходи, керувати рухами персонажів та об'єктів, а також створювати ефекти анімації, які додають жвавості та реалізму в ігровий світ. Важливим аспектом Unity є можливість підтримки різних розширень і плагінів. Розробники можуть використовувати готові плагіни для вирішення специфічних завдань, таких як робота зі штучним інтелектом, створення фізично реалістичних симуляцій, інтеграція соціальних функцій та ін. Це дає розробникам більшу свободу та гнучкість у виборі функціональності, яку вони хочуть включити у свою гру.

Завдяки потужним інструментам Unity для налагодження та профілювання, розробники можуть ефективно оптимізувати гру та виявляти і виправляти помилки та проблеми продуктивності. Це допомагає створювати більш плавний та оптимізований ігровий досвід для користувачів.

Unity пропонує можливість інтеграції з різними сервісами та платформами, такими як магазини додатків, рекламні мережі, аналітичні інструменти та ін. Це дозволяє розробникам легко монетизувати свої ігри, збирати аналітичні дані та взаємодіяти з користувачами. У підсумку, вибір Unity є кращим через його потужні функціональні можливості, підтримку безлічі платформ, зручне програмування на C#, інтегроване середовище розробки та доступність ресурсів для розширення та покращення ігрового контенту.

2.2 Характеристика середовищ розробки

JetBrains Rider – це потужне крос-платформне середовище розробки, яке призначене для роботи з різними мовами програмування, включаючи C# для розробки ігор. Вона пропонує багатий набір інструментів для написання, налагодження та профілювання коду, а також підтримує інтеграцію з Unity та іншими ігровими двигунами.

Eclipse – це популярне середовище розробки, яке широко використовується для розробки Java-застосунків, але також підтримує плагіни для розробки ігор іншими мовами, таких як C++ та Python. Eclipse надає інструменти для написання коду, налагодження та управління проєктами, а також має активну спільноту розробників.

Xcode – це IDE, розроблена компанією Apple для створення програм під операційні системи iOS, macOS, watchOS і tvOS. Вона надає інструменти для розробки ігор Swift і підтримує інтеграцію з ігровими двигунами, такими як Unity і Unreal Engine.

MonoDevelop – це безкоштовне середовище розробки, яке призначене для роботи з мовою програмування C# та іншими мовами, які підтримує фреймворк Mono. Вона має основні функції, такі як редактор коду, налагодження та підтримка систем контролю версій.

NetBeans – це платформо незалежне середовище розробки, яке підтримує різні мови програмування, включаючи Java та C/C++. Вона надає інструменти для створення ігрового контенту, таких як редактори коду, налагодження та профілювання.

Visual Studio – це потужне інтегроване середовище розробки (IDE), яке надає розробникам широкий спектр інструментів та функціональних можливостей для створення різних типів програм, включаючи ігри. Вона має кілька переваг у контексті розробки ігор.

Середовище розробки Visual Studio має інтуїтивно зрозумілий і зручний інтерфейс, який полегшує навігацію за проєктом, редагування коду та роботу

з різними інструментами. Це допомагає розробникам підвищити продуктивність та зосередитися на сутності розробки гри. Можливості розширення за допомогою плагінів дозволяють розробникам налаштовувати середовище розробки під свої потреби. Наприклад, для роботи з Unity можна встановити плагін Visual Studio Tools for Unity, який надає додаткові функціональні можливості та інструменти спеціально розроблені для роботи з Unity. Також Visual Studio має потужний редактор коду з функціями автодоповнення та перевіркою синтаксису. Це спрощує та прискорює процес написання коду, а також допомагає виявляти помилки та пропонувати виправлення ще до запуску програми. Віртуальна апаратна реалізація (ВАР) дозволяє емулювати різні пристрої і платформи під час розробки застосунків, що допомогає перевірити ефективність оптимізації.

Потужні інструменти налагодження, які дозволяють розробникам шукати та виправляти помилки в коді, надають можливості покрокового налагодження, встановлення точок зупинки та аналізу значень змінних під час виконання програми. Середовище програмування захищеної зони (СПЗ) допомагає розробникам забезпечити безпеку свого застосунку.

Крім того, integrated development environment (IDE) надає інструменти профілювання, які допомагають виявити вузькі місця та оптимізувати продуктивність програми. Проста інтеграція з популярними системами контролю версій, такими як Git, дозволяє розробникам ефективно управляти версіями коду, контролювати зміни та спільно працювати в команді над проєктом. Велика документація та активна спільнота розробників, надають підтримку, навчальні матеріали та відповіді на запитання. Розробники можуть швидко знайти інформацію, вирішити проблеми та отримати додаткові ресурси для покращення навичок розробки ігор.

Unity забезпечує інтеграцію із популярним середовищем розробки Visual Studio, що дозволяє розробникам використовувати потужні інструменти Visual Studio для розробки ігрового контенту. Visual Studio надає багатий набір

функцій для написання коду, налагодження, профілювання та управління проєктами, що спрощує процес розробки ігор.

За допомогою інтеграції Unity та Visual Studio розробники можуть створювати та редагувати скрипти мовою C# безпосередньо у середовищі Visual Studio. Це дозволяє використовувати потужні функції автодоповнення коду, перевірки синтаксису, рефакторингу та налагодження, що значно підвищує продуктивність та ефективність розробників.

Крім того, Visual Studio забезпечує інтеграцію із системою контролю версій, що дозволяє команді розробників ефективно керувати та спільно працювати над проєктом Unity. Розробники можуть використовувати функції командної розробки, злиття та контролю версій для зручної та безпечної роботи з кодом проєкту.

2.3 Характеристика мов програмування

JavaScript, також відомий як UnityScript у контексті Unity, є однією з мов програмування, що підтримуються в Unity.

JavaScript має простий і зрозумілий синтаксис. Синтаксична схожість дозволяє відносно легко освоїти UnityScript та почати розробляти ігрову логіку. JavaScript є відносно простою мовою програмування, і його можна швидко освоїти, особливо для розробників-початківців. Він не вимагає глибокого розуміння об'єктно-орієнтованого програмування та може бути використаний для швидкого прототипування та створення простих ігор. Також він тісно інтегрований з Unity і надає доступ до всіх функцій та можливостей двигуна. Він має доступ до application programming interface (API) Unity та може взаємодіяти з компонентами, об'єктами, анімацією та іншими елементами гри. UnityScript зазвичай виконується швидше, ніж C# Unity. Це може бути корисним для простих та легковажних ігор, де продуктивність є важливим аспектом.

Він є широко використовуваною мовою програмування, особливо у веб-розробці. Якщо у вже є існуючий код JavaScript, його можна використовувати в Unity, перевикористовуючи деякі функції або алгоритми, щоб прискорити розробку гри. Однак, слід зазначити, що JavaScript (UnityScript) не так активно підтримується та розвивається, як C#. Unity Technologies рекомендує використовувати C# як основну мову програмування для розробки ігор в Unity, оскільки він пропонує ширші можливості та повнішу спільноту розробників.

C# – це потужна та гнучка мова програмування, яка широко використовується в розробці ігор. У контексті створення крос-платформних ігор з використанням Unity, C# є однією з основних мов програмування, які використовуються розробниками для написання ігрової логіки, управління поведінкою об'єктів та взаємодії з користувачем. До переваг використання C# у розробці ігор, можна віднести наступне. C# пропонує простий і зрозумілий синтаксис, який робить код більш читабельним і зрозумілим. Це спрощує розробку та підтримку ігрового коду, особливо під час роботи в команді. C# має велику стандартну бібліотеку, яка надає різні класи та функції для роботи з файлами, мережею, графікою, звуком та іншими аспектами гри. Це дозволяє розробникам зосередитися на логіці гри, використовуючи готові інструменти та функціональність [3]. C# повністю підтримує концепції об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція та абстракція. Це дозволяє розробникам створювати чисту та модульну архітектуру ігрового коду, що полегшує його підтримку та розширення [4].

Спільний код (СК) дозволяє використовувати однаковий код для різних платформ, що спрощує розробку крос-платформних застосунків.

C# щільно інтегрований із популярними інтегрованими середовищами розробки (IDE), такими як Visual Studio. Це забезпечує розробникам зручне середовище для написання, налагодження та профілювання коду, а також надає різні інструменти для підвищення продуктивності та ефективності розробки. Також, C# є основною мовою програмування розробки ігор на платформі Unity. Unity надає багаті інструменти та бібліотеки для роботи з C#,

такі як компоненти, системи, модель подій та інші. Це дозволяє розробникам створювати складну ігрову логіку, керувати поведінкою об'єктів та реалізовувати інтерактивність гри. C# є популярною мовою програмування з широкою спільнотою розробників. Це забезпечує доступ до безлічі додаткових ресурсів, бібліотек, фреймворків та інструментів, які можуть бути використані для покращення розробки ігор. Також існує безліч онлайн-ресурсів, форумів та спільнот, де розробники можуть обмінюватися досвідом, ставити запитання та отримувати підтримку. Дана мова надає механізми автоматичного керування пам'яттю, такі як складання сміття, що дозволяє розробникам уникнути багатьох проблем, пов'язаних з витоками пам'яті та некоректною роботою з покажчиками. Це робить розробку ігор безпечнішою та зручнішою. C# має зручні засоби для роботи з багатопоточністю, що дозволяє створювати ігри, що використовують паралельні обчислення та асинхронні операції. Це особливо корисно для створення ігор з високою продуктивністю та обробкою великої кількості даних. Мова C# має багату екосистему бібліотек та фреймворків, які можуть значно спростити розробку ігор. Наприклад, існують спеціалізовані бібліотеки для роботи з графікою, фізикою, штучним інтелектом та іншими аспектами гри. Такі інструменти допомагають прискорити розробку та додати додаткові функціональні можливості для гри. C# є крос-платформною мовою програмування, що означає, що ви можете використовувати її для розробки ігор, що працюють на різних операційних системах, таких як Windows, MacOS, Linux, iOS та Android. Це дозволяє досягти більшої аудиторії та розширити охоплення гри. C# легко інтегрується з іншими мовами програмування, такими як C++ або JavaScript. Це особливо корисно, коли потрібно використовувати існуючий код іншою мовою або взаємодіяти з іншими компонентами гри, написаними різними мовами [5]. C# надає розробникам усі необхідні можливості для створення складної та інтерактивної ігрової логіки. Його простота, потужність та інтеграція з Unity роблять його ідеальним вибором для розробників, які прагнуть створити якісні та захоплюючі крос-платформні ігри. Таким чином

вибір Unity з Visual Studio та мови програмування C# як інструменти для розробки ігор надає ряд переваг. Unity є потужним і популярним ігровим двигуном, що має широкий спектр функцій і можливостей для створення високоякісних ігор на різних платформах. Разом з тим, інтеграція з Visual Studio, є одним з найпопулярніших інтегрованих середовищ розробки, надає зручне та продуктивне середовище розробки, полегшуючи процес написання, налагодження та профілювання коду. Вибір C# також зумовлений його широкою підтримкою Unity, де він є основною мовою програмування. Unity надає безліч інструментів та бібліотек для роботи з C#, таких як компоненти, системи, подієва модель та інші, що дозволяє створювати складну ігрову логіку, керувати поведінкою об'єктів та реалізовувати інтерактивність гри.

3 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

3.1 Опис архітектурного патерну Model-View-Controller та його застосування у застосунку

Патерн MVC (Model-View-Controller) є структурним шаблоном проєктування, який пропонує поділ програми на три основні компоненти:

- модель (Model);
- подання (View);
- контролер (Controller).

Кожен з цих компонентів має свою відповідальність і виконує конкретні завдання, що забезпечує поділ логіки програми та підвищує його гнучкість та підтримуванність [6].

Модель є основною частиною програми, що містить дані та бізнес-логіку. Вона представляє реальний об'єкт чи концепцію у застосунку та відповідає за обробку даних та виконання операцій над ними. Модель може містити класи, структури, бази даних, API та інші компоненти, які надають доступ до даних та їхньої обробки. Модель залежить від інших компонентів і може існувати незалежно. Подання відповідає за відображення даних з моделі користувача. Це компонент, який відображає інтерфейс користувача, графічні елементи та інші засоби взаємодії з користувачем. Подання отримує дані з моделі та відображає їх у зручній для користувача формі. У патерні MVC подання пасивне і не містить логіки обробки даних, воно просто відображає дані, отримані від моделі.

Контролер є посередником між моделлю та поданням. Він відповідає за обробку введення користувача, зміна стану моделі та оновлення подання. Контролер отримує введення від подання і виконує відповідні операції над моделлю. Він також оновлює подання, щоб відобразити зміни моделі. Контролер може містити логіку обробки подій, перевірку правил та інші операції, пов'язані із взаємодією моделі та уявлення.

Переваги використання патерну MVC:

- патерн MVC допомагає розділити логіку програми на окремі компоненти, що полегшує розуміння та підтримку коду;
- завдяки розділенню компонентів, кожен з них може бути змінений або замінений незалежно, не торкаючись інших компонентів;
- компоненти моделі та подання можуть бути використані в різних контекстах або програмах без змін, оскільки вони незалежні від інших компонентів;
- кожен компонент може бути протестований окремо, що полегшує розробку модульних тестів.

Для проєкту патерн MVC може надати цілу низку гнучких архітектурних рішень. Модель може надавати основні дані гри, такі як рівні, ігрові об'єкти, рахунок і т.д. Вона може містити класи, що представляють кожен із цих елементів та обробні операції з ними. Подання може бути відповідальним за відображення графічного інтерфейсу, включаючи елементи меню, ігрові об'єкти та інші елементи інтерфейсу. Воно отримує дані з моделі та відображає їх для користувача. Контролер може обробляти введення користувача, такий як натискання кнопок, переміщення ігрового персонажа та інші події. Він взаємодіє з моделлю, оновлює її стан та оновлює уявлення для відображення змін. Всі ці компоненти працюють разом, щоб забезпечити правильне функціонування гри, поділ логіки та уявлення, а також обробку введення користувача. Це дозволяє гнучкіше керувати ігровим процесом, покращувати перевикористання коду та полегшувати тестування та підтримку програми [7].

3.2 Опис модулів, необхідних під час проєктування класів ігрових об'єктів

Модуль Mesh Renderer є важливою складовою проєктування гри та відповідає за візуальне відображення 3D-моделей на екрані.

Він надає такі особливості:

- дозволяє відобразити 3D-модель на екрані за допомогою геометричних даних (мешей) і матеріалів (він працює з вершинами меша, визначаючи їх становище у просторі та їхню взаємодію з освітленням);
  - використовує матеріали, які визначають зовнішній вигляд об'єкта. (матеріали можуть містити текстури, кольори, відображення та інші властивості, що впливають на візуальне представлення моделі. Модуль дозволяє застосовувати різні матеріали до різних частин моделі або об'єкта в цілому);
  - враховує освітлення сцени та застосовує його до моделі (він обробляє інформацію про спрямоване, точкове або плямисте освітлення, щоб створити реалістичні тіні, відображення та підсвічування на поверхні моделі);
  - дозволяє відобразити об'єкти на основі шарів (це дозволяє контролювати, які об'єкти будуть відображатись перед або за іншими об'єктами, що корисно при створенні складних сцен та ефектів прозорості).

Хоча основна функція Mesh Renderer є відображенням моделей, він також може бути використаний для виявлення колізій між об'єктами. При використанні спільно з іншими модулями, такими як колайдери, Mesh Renderer дозволяє визначити, чи об'єкти стикаються і як вони повинні взаємодіяти в результаті зіткнень.

Ці особливості Mesh Renderer важливі при проєктуванні гри, оскільки вони забезпечують візуальне представлення ігрових об'єктів та створюють реалістичні та переконливі сцени. Правильне використання Mesh Renderer у поєднанні з іншими модулями, такими як анімація, освітлення та фізика, дозволяє створити цікаві та візуально привабливі ігрові світи.

Модуль Transform є одним з основних модулів, що відповідає за позицію, масштабування та обертання ігрового об'єкта у тривимірному просторі. Він має наступні основні властивості. Позиція визначає місце

розташування ігрового об'єкта в тривимірному просторі. Вона задається у вигляді трьох координат (x, y, z), де x – горизонтальна вісь, y – вертикальна вісь, а z – вісь глибини. Зміна позиції призводить до переміщення об'єкта простором. Масштабування – дозволяє змінювати розміри ігрового об'єкта. Воно також задається у вигляді трьох координат (x, y, z), де кожна координата визначає масштабування відповідної осі. Зміна масштабу впливає на розміри об'єкта та його дочірніх елементів. Обертання – визначає орієнтацію ігрового об'єкта у просторі. Воно задається за допомогою кватерніону, який є комплексним числом з чотирма компонентами: x, y, z і w. Обертання дозволяє змінювати напрямок об'єкта навколо всіх трьох осей (x, y, z) [8].

Також Transform зберігає свою позицію, масштаб та обертання у власній локальній системі координат. Це означає, що значення Transform відносяться до власного простору ієрархії. Наприклад, позиція (0, 0, 0) у локальному просторі Transform означає, що об'єкт знаходиться у його батьківському об'єкті або сцені.

Rigidbody є одним із основних компонентів фізики в Unity. Він дозволяє об'єктам взаємодіяти з фізичним світом та отримувати реалістичну поведінку. При додаванні компонента Rigidbody до ігрового об'єкта, об'єкт стає динамічним і починає реагувати на сили та зіткнення. Маса визначає кількість речовини в об'єкті та її інерцію. Чим більша маса, тим складніше змінити швидкість об'єкта. Маса впливає на взаємодію об'єкта з іншими об'єктами при зіткненнях. Демпінг визначає силу опору, що діє об'єкт під час руху. Збільшення значення демпінгу може уповільнити об'єкт, а зменшення – прискорити рух. Кутовий демпінг визначає силу опору, яка діє на об'єкт під час обертання. Аналогічно демпінгу збільшення значення кутового демпінгу сповільнить обертання об'єкта, а зменшення – прискорить його обертання.

Параметр "Використання гравітації" визначає, чи об'єкт буде підданий силі гравітації. Якщо прапор встановлений, об'єкт падатиме вниз під впливом гравітації. Якщо прапор знятий, об'єкт залишатиметься у повітрі, незалежно від гравітації. Заморожування осей дозволяє заморозити певні осі позиції та

обертання об'єкта. Це корисно, коли потрібно зберегти об'єкт у певному положенні або обмежити його рух чи обертання певними осями.

Застосування сил надає імітацію дії фізичних сил на об'єкт. Для цього слід використовувати методи, такі як AddForce, AddTorque та інші, щоб прикладати сили до об'єкта у різних напрямках та моментах.

Зіткнення обробляє взаємодію з іншими об'єктами. Він розраховує реакцію на зіткнення, включаючи відскок, пружність та втрату енергії при зіткненні. Rigidbody також надає події, які дозволяють відстежувати зіткнення та виконувати додаткові дії у відповідь на них.

Компонент Rigidbody є важливою частиною створення реалістичної фізики у грі. Він дозволяє об'єктам рухатися, взаємодіяти з навколишнім середовищем та реагувати на сили та зіткнення в ігровому світі. Використання компонента Rigidbody може значно підвищити реалізм та геймплейність гри, особливо в іграх, де фізика відіграє важливу роль, таких як симулятори, платформери чи головоломки.

Універсальні компоненти (УК) дозволяють розробникам швидко створювати крос-платформні функції для своїх застосунків.

Collider – це компонент, який визначає геометричну форму та межі об'єкта для цілей фізичної взаємодії та виявлення зіткнень. Він використовується у поєднанні з компонентом Rigidbody для створення реалістичної фізики та обробки зіткнень в ігровому світі. Кожен колайдер має форму, яка визначає геометричну форму об'єкта. Форма може бути прямокутником (BoxCollider), сферою (SphereCollider), капсулою (CapsuleCollider), мешем (MeshCollider) тощо. Для деяких типів колайдерів, таких як BoxCollider або SphereCollider можна налаштувати розміри форми. Розміри дозволяють визначити розмір колайдера у тривимірному просторі, такий як ширина, висота та глибина для BoxCollider або радіус для SphereCollider. Параметр "Тригер" визначає, що колайдер не імітує фізичне зіткнення але може повідомити про нього. Якщо прапор встановлено,

колайдер діє як тригер і не викликає реакцію на фізичні зіткнення. Натомість, він генерує події при вході і виході інших об'єктів у його область.

Колайдери можуть мати фізичні матеріали, які визначають їх властивості зіткнення, такі як тертя, пружність та міцність. Матеріали можуть бути налаштовані для досягнення бажаної поведінки зіткнень об'єктів.

Unity пропонує кілька методів виявлення зіткнень, таких як дискретне виявлення (Discrete Collision Detection) та безперервне виявлення (Continuous Collision Detection).

Ігнорування зіткнень дозволяє налаштовувати ігнорування зіткнень з певними шарами або об'єктами в сцені. Це корисно, коли необхідно запобігти зіткненням між певними об'єктами або певними групами об'єктів.

3.3 Проєктування менеджерів та контролерів

Клас GameManager є центральним компонентом гри, що відповідає за управління ігровим процесом та пов'язаними з ним елементами. Він реалізує основну логіку гри та взаємодіє з іншими компонентами та системами гри.

Всередині класу визначені приватні змінні, включаючи посилання на різні ігрові об'єкти та компоненти, такі як ball, selector, pipe, camera, levelGenerator, finishLine, а також змінні стану гри, такі як isPlaying, jumpCounter, themeIndex та CoinsAmount.

Метод Start викликається при старті гри та виконує ініціалізацію та налаштування ігрових компонентів. Він встановлює початкові значення змінних, активує необхідні ігрові елементи та налаштовує інтерфейс гри.

Метод Update викликається кожен кадр гри та відстежує стан гри та ігрових об'єктів. У цьому випадку він оновлює позицію кулі та оновлює інформацію про пройдену відстань на інтерфейсі. Клас також містить кілька публічних методів, таких як OnPlay, AddCoins, OnLevelComplete та OnLose. Ці методи реагують на певні події у грі та виконують відповідні дії, такі як початок гри, додавання монет, завершення рівня чи поразка.

В цілому, клас GameManager є ключовим компонентом гри, що забезпечує управління ігровим процесом та взаємодію з іншими елементами гри. Його методи та властивості визначені таким чином, щоб забезпечити правильне виконання логіки гри та взаємодію з ігровими об'єктами та інтерфейсом.

Клас LevelGeneratorController містить кілька структур, які представляють компоненти даних для ігрових об'єктів рівня. Наприклад, LevelObjectFlag, GeneratedLevelData, PipeData, ObstacleData, RotatorData та CoinData визначають різні характеристики та параметри об'єктів рівня, таких як труби, перешкоди та монети.

LevelGeneratorController має методи та властивості для управління генерацією рівня. Наприклад, OnInited викликається при ініціалізації контролера та створює сутність для представлення згенерованого рівня. Методи SetCurrentLevelAsGenerated і ClearLevelObjectsData служать для встановлення поточного рівня як згенерованого та очищення даних про об'єкти рівня відповідно. Клас також надає методи для створення конкретних об'єктів рівня, таких як CreateObstacle, CreatePipe, AddPipeRotator та CreateCoin. Ці методи використовуються для створення сутностей із відповідними компонентами даних. Додатково, клас надає методи отримання даних про об'єкти рівня, такі як GetLevelCoinsData, GetPipesData, GetPipes і GetObstaclesData. Ці методи повертають списки відповідних типів об'єктів.

Загалом клас LevelGeneratorController відіграє важливу роль у генерації та управлінні об'єктами ігрового рівня. Він взаємодіє з іншими модулями та системами, надаючи методи для створення, отримання та управління об'єктами рівня, що дозволяє динамічно генерувати рівні та забезпечувати різноманітність ігрового процесу.

HighScoreController успадковується від базового класу Controller та перевизначає метод InitDefault, який створює сутність для зберігання вищого рахунку за умовчанням. Клас надає кілька методів для роботи з найвищим рахунком гри. Метод TrySaveScore дозволяє зберегти новий рахунок, якщо він

перевищує поточний вищий рахунок. Метод SaveNewScore використовується для збереження нового рахунку, а GetScore повертає поточний вищий рахунок гри. В цілому HightScoreController взаємодіє з іншими модулями та системами, забезпечуючи збереження та відстеження найвищого досягнутого ігрового рахунку.

Клас SuperballLevelsController є похідним від базового класу LevelBasedController і представляє контролер управління рівнями гри Superball. У даному класі визначено кілька методів та властивостей, які використовуються для перевірки умов перемоги чи поразки, визначення виграшних умов, підрахунку кількості зірок за перемогу та інших ігрових механік. Клас містить посилання на об'єкт SuperballMoneyConfig, який представляє конфігурацію ігрової валюти. Властивість levelWinMoney обчислює суму виграшної валюти на основі базового значення та поточної кількості монет гравця.

Якщо визначено макрос ADS, то клас також містить властивість levelWinX3Chance, що визначає можливість отримання виграшної валюти помноженої на три. Властивість winStarsCount визначає кількість зірок, яка буде присуджена при перемозі на рівні. У поточній реалізації значення завжди дорівнює 3. Клас також містить прапор LastGameIsWin, який вказує, чи була остання гра переможною. Методи OnWin та OnLose викликаються при перемозі та поразці відповідно, і встановлюють відповідне значення прапора.

Методи RestartLocation та Play викликаються при перезапуску рівня та початку гри відповідно. Вони виконують необхідні дії для скидання стану гри та сповіщення інших систем та модулів.

Метод DebugChangeLocation використовується для зміни поточного рівня в режимі налагодження та викликає перехід до головного меню.

Загалом, SuperballLevelsController надає функціональність для управління рівнями гри Superball, перевірки умов перемоги чи поразки, обчислення виграшної валюти та зірок, а також управління станом гри та обробкою подій під час перемоги чи поразки.

3.4 Опис ігрових об'єктів

Основним компонентом гри є куля, якою керує гравець. Клас "Ball" є компонентом ігрового об'єкта, що представляє кулю у грі. Він реалізує інтерфейс "IControllable", що дозволяє контролювати його поведінку.

Всередині класу визначено перелік "BallState", що містить можливі стани кулі: "FreeFlight" (вільний політ), "EnteringPipe" (вхід у трубу), "InPipe" (всередині труби) та "LeavingPipe" (вихід із труби). Властивість "State" дозволяє отримати стан кулі. Клас містить ряд приватних змінних, таких як посилання на ігрові об'єкти (наприклад, земля), компоненти (наприклад, Rigidbody2D) та вектори (наприклад, швидкість та гравітація). Вони використовуються для управління та відстеження стану кулі. Також у класі визначено події та делегати, які дозволяють іншим компонентам підписуватись на певні події кулі, такі як успішний стрибок, зміна можливості управління та інші.

Методи класу включають функціональність, пов'язану з різними аспектами поведінки кулі. Наприклад, методи "OnPlay" та "LevelDone" відповідають за початок гри та завершення рівня відповідно. Методи "OnDestroy" та "Update" обробляють руйнування кулі та її оновлення в кожному кадрі гри.

Клас " Ball " також містить методи, відповідальні за взаємодію Космосу з іншими об'єктами у грі. Наприклад, метод OnTriggerEnter2D обробляє подію зіткнення кулі з іншими колайдерами, такими як Finish (кінцева точка рівня), pipeEntrance (вхід у трубу), obstacle (перешкода) і coin (монета).

Клас "Ball" є ключовим компонентом гри, який забезпечує функціональність та керування кулею. Його методи та властивості визначені таким чином, щоб забезпечити плавний рух, обробку взаємодій з іншими об'єктами та відповідність правилам та умовам гри.

Клас "Pipe" є компонентом ігрового об'єкта, що представляє трубу у грі. Він відповідає за керування поведінкою та властивостями труби.

Усередині класу визначено такі змінні:

- typeIndex – індекс типу труби;
- entrances – масив входів труби;
- materials – масив матеріалів труби;
- spline – крива для моделювання форми труби;
- bounds – прямокутник, що визначає межі труби.

Клас "Pipe" також містить такі методи:

- getSampleAtDistance – повертає інформацію про точку на кривій трубі на вказаній відстані;
- getClosestSample – повертає інформацію про найближчу точку на кривій трубі до зазначеної позиції;
- getSampleWorldPosition – повертає світову позицію точки на кривій трубі;
- getSampleWorldDirection – повертає світовий напрямок точки на кривій труби;
- length – властивість, що повертає довжину труби;
- spawn – метод, який використовується для встановлення позиції труби.

Клас "Pipe" також має методи Start та OnValidate, які викликаються при запуску гри та при валідації даних у редакторі Unity відповідно. У Start вибирається випадковий матеріал з масиву матеріалів для відображення труби, а OnValidate перевіряє наявність компонента Spline і присвоюється значення, якщо воно не було встановлено. Клас "Pipe" відіграє важливу роль у грі, визначаючи форму та властивості труби. Він надає методи для отримання інформації про точки на кривій трубі та взаємодії з іншими об'єктами у грі.

Клас CoinGeneration відповідає за генерацію монет у грі. Він є компонентом ігрового об'єкта та містить логіку, пов'язану з обробкою зіткнень та збільшенням кількості монет.

Усередині класу визначено такі змінні:

- coinAmount – кількість монет;

- rb – посилання компонент Rigidbody2D поточного об'єкта;
- ball – посилання на компонент Ball;

Клас CoinGeneration також містить методи для контролю процесу гри.

Метод Start, який викликається при старті гри (всередині цього методу ініціалізується змінна coinAmount за допомогою значення, збереженого в PlayerPrefs, і виходить посилання на компонент Rigidbody2D поточного об'єкта);

Метод OnTriggerEnter2D, який викликається при зіткненні об'єкта з іншим колайдером (усередині цього методу перевіряється, чи зіткнення сталося з "leftTube" або "rightTube" і якщо куля рухається вгору, то кількість монет збільшується. Якщо jumpCounter (лічильник стрибків кулі) ділиться на 5 без залишку, то збільшення кількості монет відбувається на 3, інакше на 1. Потім оновлена кількість монет зберігається в PlayerPrefs).

Клас CoinGeneration відіграє важливу роль у генерації монет у грі. Він обробляє зіткнення кулі з певними об'єктами ("leftTube" та "rightTube") та збільшує кількість монет залежно від умов. Цей клас дозволяє гравцеві отримувати монети та керувати ними в ігровому процесі.

Клас "GameSettings" успадковує від "SettingsScriptable<GameSettings>" і містить різні налаштування та методи, пов'язані з ігровими налаштуваннями.

Усередині класу визначено такі змінні та властивості:

- spawnPos – вектор, що вказує позиції спавна об'єктів у грі;
- gravity – приватне поле, що має значення гравітації у грі.

Клас "GameSettings" також містить методи для контролювання фізичної поведінки.

Метод GetRequiredForce повертає необхідну силу для стрибка (він приймає початкову позицію (startPoint), кінцеву позицію (endPoint) та тривалість стрибка (duration). Всередині методу розраховується і повертається необхідна сила для стрибка у вигляді вектора force, ґрунтуючись на різниці між координатами початкової та кінцевої позиції та тривалістю стрибка);

Метод GetTrajectoryPoint повертає позицію стрибка у заданий час (він приймає початкову позицію (startPoint), кінцеву позицію (targetPoint), час, що минув з початку стрибка (time) та множник (multiplier, за умовчанням дорівнює 1). Всередині методу розраховується і повертається позиція на траєкторії стрибка в заданий час, використовуючи обчислену необхідну силу для стрибка).

Клас "GameSettings" надає налаштування та методи, пов'язані з ігровими параметрами, такими як позиція спавна об'єктів, гравітація та розрахунок стрибкової траєкторії. Він забезпечує зручний доступ до цих параметрів та функціональність, пов'язану з фізичними властивостями гри.

Клас "JoystickShower" відповідає за відображення та оновлення елементів, пов'язаних з джойстиком на екрані гри.

Усередині класу визначено такі змінні та властивості:

- images – масив зображень (Image), які відображатимуться на екрані в парі з джойстиком;
- joystick – посилання на компонент Joystick, який є джойстиком для управління ігровим персонажем;
- originTr – посилання на Transform, що представляє позицію та обертання базової точки джойстика;
- pointerTr – посилання на Transform, що представляє позицію покажчика джойстика;
- joyBg – посилання на Transform, що представляє позицію та обертання заднього фону джойстика.

Методи класу включають функціональність, пов'язану з оновленням та відображенням елементів джойстика:

- метод Update виконується для кожного кадру гри (всередині методу оновлюється максимальна відстань (maxDistance) для джойстика, і вмикаються/вимикаються зображення (images) залежно від значення showJoystick із конфігурації BallConfig);

- метод LateUpdate викликається після оновлення кадру (всередині методу відбувається коригування позиції покажчика джойстика (pointerTr), щоб він був на одній висоті з базовою точкою джойстика (originTr).
- Задня частина джойстика (joyBg) також встановлюється в позицію базової точки).

Клас "JoystickShower" забезпечує коректне відображення елементів джойстика на екрані гри та оновлення їх стану відповідно до конфігурації BallConfig. Це дозволяє гравцеві зручно керувати персонажем за допомогою джойстика у грі. Ігровий об'єкт "Obstacle" є перешкодою. Цей клас успадковується від MonoBehaviour з Unity та містить деякі серіалізовані поля та методи для роботи з колайдером та межами перешкоди.

Поля класу:

- collider – посилання на компонент Collider2D, який визначає форму та область дії перешкоди;
- bounds – обмежує прямокутник (кордону) перешкоди.

Метод OnValidate викликається Unity за зміни значень у редакторі. У цьому методі відбувається перевірка, якщо колайдер не заданий, він отримує посилання на об'єкт Collider2D з дочірніх об'єктів перешкоди, і якщо такий компонент знайдений, то оновлюються межі перешкоди.

Мета ігрового об'єкта "Obstacle" у грі Superball полягає у створенні перешкоди, яка взаємодіятиме з іншими ігровими елементами. Це може бути, наприклад, стіна, платформа, шипи і т. д. Перешкода може мати колайдер для визначення зіткнень з іншими об'єктами та використовуватись у логіці гри для створення складних рівнів, додавання виклику певних подій чи перешкод для гравця. Приклад використання "Obstacle" у грі може бути наступним:

- створення стіни, через яку куля неспроможна пройти;
- розміщення шипів на платформі, щоб гравець не зміг наступити на них;
- розміщення перешкоди, що блокує проходження до наступного рівня до виконання певної умови.

Загалом, "Obstacle" це ігровий об'єкт, який створює перешкоду і використовується для додавання складності та інтерактивності в ігровий процес. Клас "FinishLine" є компонентом ігрового об'єкта, який позначає кінцеву точку рівня або мету, яку гравець повинен досягти. Він визначає поведінку та умови, пов'язані із завершенням рівня. Клас "CameraFollow" є компонентом ігрової камери, який відповідає за її позиціонування та слідування за цільовим об'єктом у грі. Він забезпечує плавне відстеження руху об'єкта та підтримку його у центрі кадру.

Діаграма класів для ігрових об'єктів представлена на рис. 4.

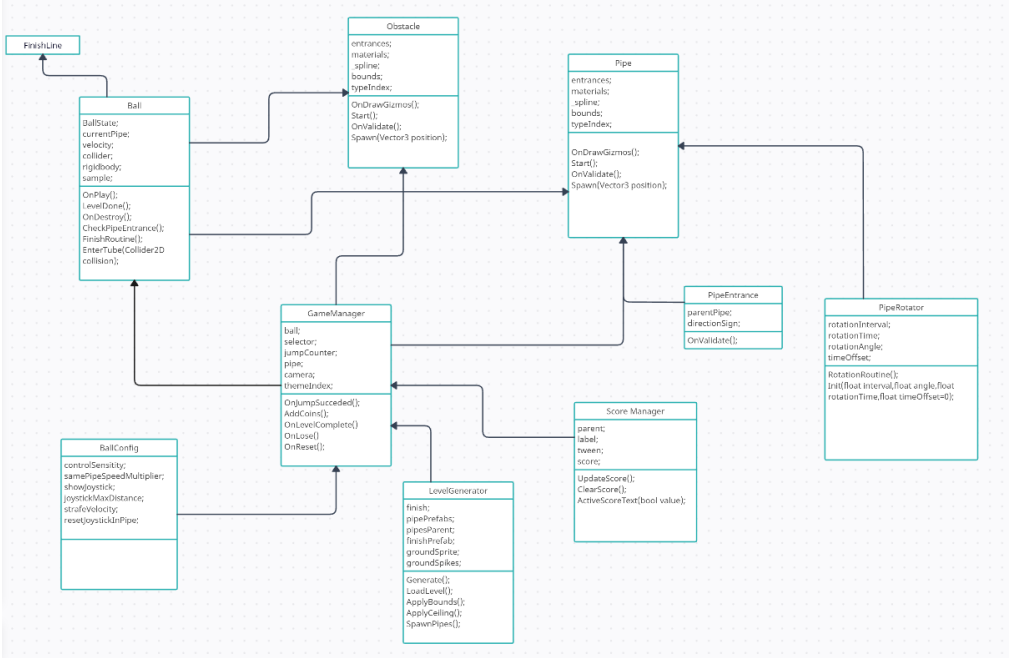

Рисунок 4 – Діаграма класів

3.5 Опис ігрового інтерфейсу

Ігровий інтерфейс повинен містити наступний набір меню:

- меню перемоги;
- меню програшу;
- меню паузи та налаштувань;

– меню придбання варіацій зовнішнього вигляду.

Меню перемоги має включати такі елементи:

– заголовок, що повідомляє гравця про перемогу;.
– рахунок, який відображую набраних очок або досягнень;
– кнопка "Продовжити", яка дозволяє гравцеві продовжити на наступний рівень або повертатися до ігрового меню;
– кнопка "Поділитись", яка дозволяє гравцеві поділитися своїм результатом у соціальних мережах або з друзями.

Поле містить кількість монет, отриманих за рівень, а також текстури у вигляді 3 зірок, які заповнюються відповідно до відсотка проходження рівня та зібраних монет. Кнопка "Перегратии" , яка дозволяє гравцеві розпочати рівень заново.

Меню програшу має включати такі елементи:

– заголовок, який повідомляє гравця про програш;
– рахунок, який відображує набрані очки або досягнення;
– кнопка "Спробувати знову", яка дозволяє гравцеві спробувати рівень знову;
– кнопка "Вийти в меню", яка дозволяє гравцеві повернутися до головного меню гри.

Меню паузи та налаштувань має включати такі елементи:

– заголовок, який повідомляє гравця про паузу;
– кнопка "Продовжити", яка дозволяє гравцеві продовжити гру після паузи;
– кнопка "Вібрація", яка дозволяє гравцеві увімкнути або вимкнути вібрацію;
– кнопка "FPS60", яка дозволяє гравцеві встановити обмеження кадрів на секунду до 60 (підходить лише для гравців із продуктивними пристроями);

- кнопка "terms of use", необхідна для ознайомлення з умовами користування продуктом (необхідна для подальшого додавання микротранзакцій);
- кнопка "privacy policy", необхідна для ознайомлення з умовами приватності політики продукту, а також майданчиків, на яких продукт буде доступний для завантаження (необхідна для подальшого додавання мікротранзакцій та додавання гри в магазини та ігрові майданчики);
- кнопка "Вийти в меню", яка дозволяє гравцеві повернутися до головного меню гри;
- кнопка "Параметри", яка дозволяє гравцеві настроїти параметри гри, такі як гучність або керування.

Меню покупки моделей персонажа має включати такі елементи:

- заголовок, який повідомляє гравця про меню покупки скінів;
- список доступних скінів, який відображує доступні дизайни кулі з їх назвами, зображеннями та цінами;
- кнопка "Купити", яка дозволяє гравцеві купити обраний скін;
- кнопка "Назад", яка дозволяє гравцеві повернутися до попереднього меню або головного меню гри.

Кожне меню може мати свій унікальний дизайн, який відповідає загальному стилю гри. Важливо забезпечити зручну навігацію та ясні інструкції для гравця, щоб він міг легко зрозуміти, як використовувати інтерфейс та виконувати необхідні дії, такі як продовження гри, перегравання рівня чи покупка скінів.

4 РЕАЛІЗАЦІЯ ІГРОВОГО ЗАСТОСУНКУ

4.1 Реалізація руху м'яча по трубі за допомогою сплайн-інтерполяції

Сплайн-інтерполяція – метод, що використовується для плавного та безперервного інтерполювання значень між набором точок, що управляють. У контексті руху м'ячика вздовж труби сплайн-інтерполяція може бути застосована для створення гладкого шляху руху м'ячика від однієї точки труби до іншої. У цій кваліфікаційній роботі використовується кубічний сплайн, який використовує поліноми третього ступеня інтерполяції між керуючими точками. Кубічний сплайн забезпечує гладкість та безперервність кривої, що є важливим для створення плавного руху м'ячика.

Формула кубічного сплайну виглядає так:

$$S(x) = a + b*(x - x_i) + c*(x - x_i)^2 + d*(x - x_i)^3, \quad (1)$$

де S(x) – інтерпольоване значення у точці x;

x – незалежна змінна, що визначає положення на сплайні;

x_i – i-я точка, що управляє;

a, b, c, d – коефіцієнти, які потрібно знайти для кожного інтервалу між точками, що управляють.

Для знаходження коефіцієнтів використовуються умови, які забезпечують гладкість та безперервність сплайну. Зазвичай використовуються умови природного сплайну, які наказують встановлення другої похідної рівної нулю на краях інтервалів. Ці коефіцієнти можуть бути обчислені з використанням алгоритму, такого як метод тридіагональних матриць (tridiagonal matrix algorithm) або метод розв'язання системи лінійних рівнянь. Після знаходження коефіцієнтів для всіх інтервалів між керуючими точками, можна використати формулу сплайну для обчислення інтерпольованих значень між керуючими точками в кожній точці часу або відстані вздовж труби, як слідує з (1).

При вході кульки у трубу та її подальше пересування виконується за допомогою інтерполяції. Коли кулька знаходиться всередині труби (BallState.InPipe), вона переміщається кривою труби із заданою швидкістю (pipeVelocity). Кожен кадр значення tubeDistance збільшується або зменшується на основі швидкості і напряму руху кульки. Потім для кожного значення tubeDistance утворюється відповідний семпл кривої труби за допомогою методу GetSampleAtDistance з класу Pipe. Цей семпл містить інформацію про позицію та направлення в даній точці кривої.

Потім позиція кульки оновлюється, щоб відповідати позиції кривої семпла. Таким чином, кулька переміщається вздовж труби із заданою швидкістю та в потрібному напрямку. Також усередині труби кулька повертається навколо своєї осі за допомогою методу Rotate. Кутова швидкість повороту визначається змінною angularvelocity, яка залежить від напрямку руху кульки. Вся ця логіка виконується в методі Update, який викликається кожним кадром, щоб оновити стан кульки всередині труби. При виході з труби (BallState.LeavingPipe) кулька продовжує рух по прямій лінії із заданою швидкістю та напрямком, визначеними на момент виходу з труби. Таким чином, інтерполяція в трубі заснована на переміщенні кульки між семпла кривої труби і оновленні його позиції і повороту кожен кадр.

4.2 Реалізація інтерфейсу застосунку

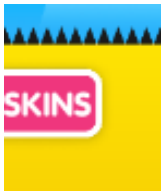

Рисунок 5 – Вид кнопки, що відкриває магазин

Принцип роботи магазину у грі Superball можна описати на основі набору класів, що описують його структуру. Клас

SuperballMoneySkinController відповідає за керування скінами та їх купівлю в магазині. Метод GetSkinPrice повертає ціну скіна залежно кількості вже відкритих скінів. Метод ActivateSkin активує обраний скін і викликає подію SkinActivated, сповіщаючи, що скін був активований. Методи Subscribe та Unsubscribe дозволяють підписуватися та відписуватися від події активації скіна. Клас BallSkinView відображає зовнішній вигляд м'ячика на основі вибраного скіна. Вид магазину до покупки нового скіна представлений на рис. 6.

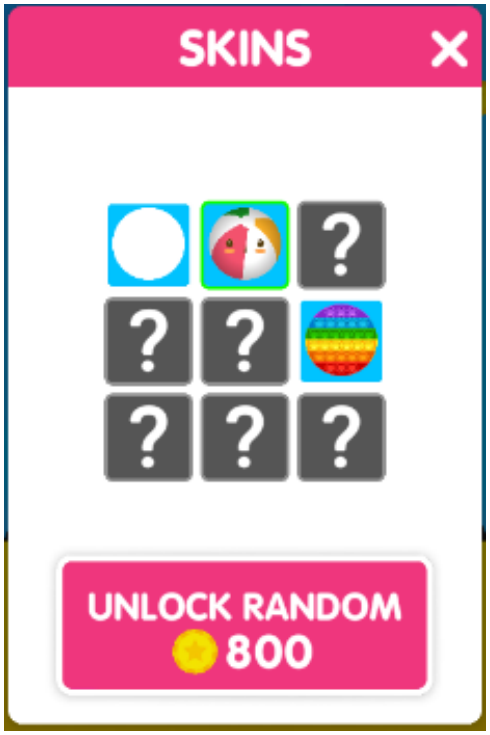

Рисунок 6 – Вид магазину до покупки нового вигляду

Метод SetSkin встановлює спрайт м'ячика відповідно до вибраного скіна (див.рис. 7).

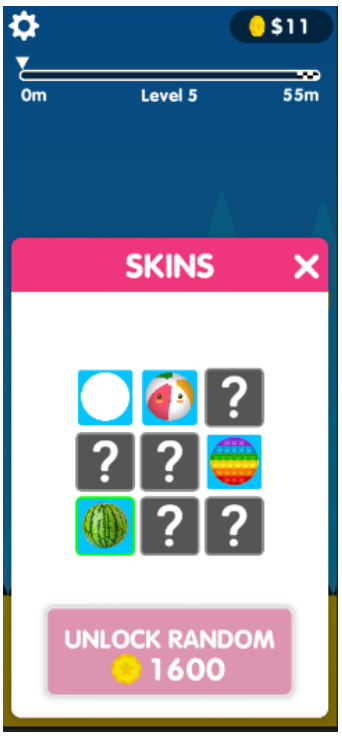

Рисунок 7 – Вид магазину після покупки нового вигляду

Клас SuperballSelectorView відповідає за відображення списку доступних скінів в ігровому магазині. Статичне поле ActiveSkins зберігає список активних скінів, які відображатимуться в магазині. Метод AddSkin додає новий скін до списку активних скінів. Метод CleanUp видаляє зі списку активних скінів усі нульові посилання на об'єкти скінів. Метод Update оновлює відображення списку скінів у магазині під час зміни активного скіна. UpdateSkins оновлює активні скіни в магазині на основі поточного активного скіна.

Алгоритм роботи магазину у грі Superball наступний:

- гравцю надається список доступних скінів, які можна придбати у магазині;
- кожен скін має власну унікальну ціну, яка визначається класом SuperballMoneySkinController;
- гравець може вибрати скін та придбати його, якщо у нього достатньо ігрової валюти;

- після придбання скіна, він стає активним і застосовується до м'яча у грі;
- магазин відображає список доступних скінів, їх ціни та статус (активний/неактивний) на основі класу SuperballSelectorView;
- при зміні активного скіна магазин оновлює список скінів і відображає новий вигляд.

Таким чином, гравець має можливість вибирати та купувати різні скіни для м'ячика в магазині, а також встановлювати активний скін, який відображатиметься у грі.

Меню паузи реалізовано за допомогою таких елементів:

- pauseMenuController – це контролер меню паузи, що відповідає за керування та відображення меню паузи;
- кнопка RestartLevelButton для перезапуску рівня меню паузи (при натисканні цієї кнопки відбувається перезапуск поточного рівня);
- перемикач вібрацій VibrationToggle у меню паузи, дозволяє гравцеві вмикати або вимикати вібрації у грі;
- перемикач обмеження FPS до 60 у меню паузи (при включенні цієї опції гра працюватиме з обмеженням FPS до 60 для більш плавного відображення);
- посилання на сторінку умов використання (при натисканні на це посилання відкривається сторінка з детальними умовами гри);
- посилання на сторінку політики конфіденційності (при натисканні на це посилання відкривається сторінка з детальною політикою конфіденційності гри);
- подання (візуальне відображення) меню паузи (включає всі елементи меню паузи, такі як кнопки, перемикачі та посилання);
- метод OnRestartLevelButtonClicked, викликаний при натисканні на кнопку перезапуску рівня меню паузи (у цьому методі відбувається логіка перезапуску рівня);

- метод OnVibrationToggleChanged, який викликається при зміні стану перемикача вібрацій у меню паузи (у цьому методі відбувається обробка зміни стану вібрацій, а саме увімкнення чи відключення);
- метод OnFPSToggleChanged, який викликається при зміні стану перемикача обмеження FPS в меню паузи (у цьому методі відбувається обробка зміни стану обмеження FPS, а саме включення чи відключення обмеження до 60);
- метод OnTermsOfUseLinkClicked – метод, який викликається при натисканні на посилання умов використання меню паузи.
- метод OnPrivacyPolicyLinkClicked, який викликається при натисканні на посилання політики конфіденційності в меню паузи.

Таким чином, опис реалізації меню паузи в розділі "Реалізація гри" включає опис контролера, елементів інтерфейсу, методів обробки подій та посилань на важливі документи, такі як умови використання та політика конфіденційності.

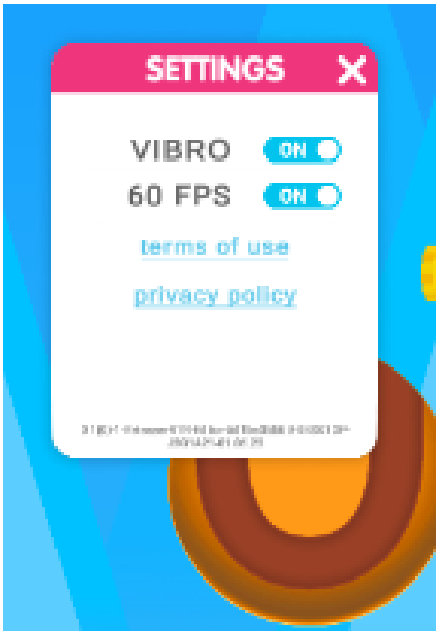

Рисунок 8 – Меню налаштувань

Реалізація меню програшу містить такі елементи:

– контролер меню програшу, що відповідає за керування та відображення меню;
– кнопка перезапуску (при натисканні цієї кнопки відбувається перезапуск поточного рівня);
– напис "Game Over" у меню програшу (відображається для передачі інформації про програш гравця);
– подання (візуальне відображення, яке включає елементи меню, такі як кнопки та написи);
– метод OnRestartButtonClicked, який викликається при натисканні на кнопку перезапуску в меню програшу (у цьому методі відбувається логіка перезапуску рівня).

Таким чином, опис реалізації меню програшу у розділі "Реалізація гри" включає контролер, елементи інтерфейсу, метод обробки події натискання на кнопку рестарту та напис "Game Over", який інформує гравця про поразку.

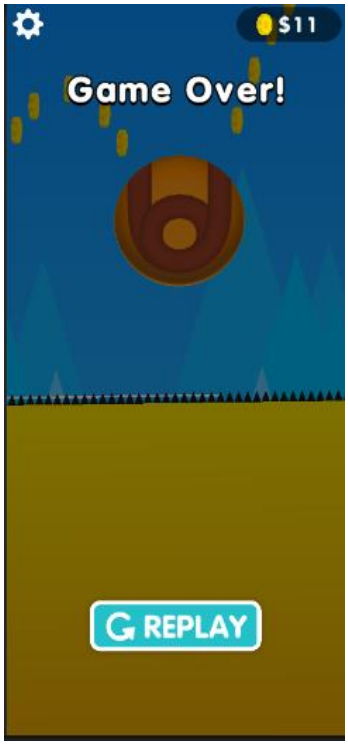

Рисунок 9 – Екран програшу

Реалізація меню проходження рівня містить такі компоненти:

- контролер меню перемоги, що відповідає за керування та відображення меню перемоги;
- кнопка "Поділитись" у меню перемоги (при натисканні на цю кнопку гравець може поділитися своїм успіхом на соціальних мережах або надіслати повідомлення);
- starHolders – три холдери для текстури зірок в меню перемоги (ці холдери заповнюються залежно від рівня проходження рівня. Наприклад, якщо гравець отримав максимальну кількість зірок, всі три холдери будуть заповнені);
- напис "Level X Completed" у меню перемоги (відображає інформацію про завершення рівня, де X - номер рівня);
- напис із кількістю зібраних монет у меню перемоги (відображає інформацію про те, скільки монет було зібрано на цьому рівні);
- кнопка "Наступний рівень" у меню перемоги (при натисканні на цю кнопку відбувається перехід до наступного рівня);
- метод OnShareButtonClicked, який викликається при натисканні на кнопку "Поділитися" в меню перемоги (у цьому методі відбувається логіка обробки події поділитися результатами гри);
- метод OnNextLevelButtonClicked, який викликається при натисканні на кнопку "Наступний рівень" у меню перемоги (у цьому методі відбувається логіка початку наступного рівня).

Таким чином, опис реалізації меню перемоги в розділі "Реалізація гри" включає контролер, елементи інтерфейсу (кнопки, холдери, написи), методи обробки подій натискання на кнопки та відображення інформації про перемогу гравця, такі як кількість зібраних монет та ступінь проходження рівня (заповнення зірок).

ЗІ уніфікує взаємодію з різними платформами та дозволяє легко адаптувати застосунок до різних пристроїв.

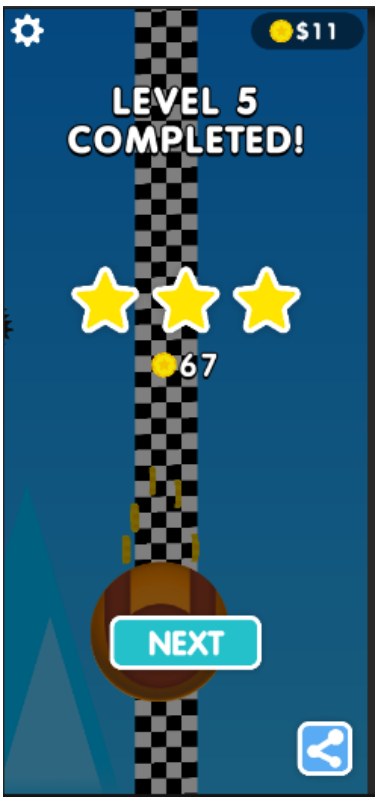

Рисунок 10 – Екран перемоги

4.3 Реалізація анімацій та візуальних ефектів у грі за допомогою бібліотеки DOTween та компонента ParticleSystem

DOTween – це потужна бібліотека анімації для Unity, яка надає простий та гнучкий спосіб створення та управління анімаціями в ігровому проєкті. Вона полегшує процес створення плавних та інтерактивних анімацій, дозволяє контролювати властивості об'єктів, такі як позиція, розмір, обертання, прозорість, колір та інші.

Бібліотека включає багато елементів для керування поведінкою анімацій.

Tween уявляє собою основну одиницю анімації в DOTween. Об'єкт Tween являє собою анімацію для конкретної властивості об'єкта, наприклад, переміщення (Move), зміна розміру (Scale), зміна кольору (Color), зміна

прозорості (Alpha) та інші. DOTween надає широкий набір методів для створення Tween та встановлення їх параметрів;

Sequence являє собою послідовність анімацій, які можуть бути відтворені одна за одною. За допомогою Sequence можна комбінувати кілька Tween і задавати порядок виконання. Можна додавати Tween до Sequence з використанням методів, таких як Append, Prepend, Insert та інших;

Callback представляє функцію зворотного дзвінка, яка буде виконана після завершення анімації. Це може бути корисним для запуску інших дій або анімацій після завершення поточної анімації. Методи, такі як OnComplete, OnStart та інші, дозволяють додавати Callback до Tween або Sequence.

Ease надає безліч вбудованих типів плавності анімації (Ease), які можна використовувати для досягнення різних анімаційних ефектів. Деякі з них включають Ease.InQuad, Ease.OutCubic, Ease.InOutExpo та інші. Також можна створювати свої власні функції Ease.

DOTween пропонує різні методи управління анімацією, включаючи Play, Pause, Resume, Rewind і Kill. Метод Play використовується для запуску анімації, Pause – для припинення, Resume – для продовження, Rewind – для перемотування в початковий стан, а Kill – для примусового завершення анімації. DOTween надає методи для налаштування параметрів анімації, таких як тривалість (SetDuration), тип плавності (SetEase), затримка перед початком (SetDelay), кількість повторень (SetLoops) та інші. DOTween також дозволяє керувати об'єктами під час анімації. Можна змінювати їх властивості, наприклад, позицію, розмір, обертання, прозорість, колір тощо, у процесі відтворення анімації. DOTween надає безліч інших функцій та методів, які дозволяють створювати складні та інтерактивні анімації в ігровому проєкті. Бібліотека проста у використанні та має докладну документацію та приклади коду, що робить її популярним інструментом для розробки ігрових анімацій у Unity. Анімація коливань кулі на фініші виконується за допомогою:

```
transform.DOShakePosition(1f, 0.1f, 10, 90, false, false);
```

Компонент Particle System в Unity є потужним інструментом для створення та управління ефектами частинок в ігровому проєкті. Він дозволяє створювати різноманітні візуальні ефекти, такі як дим, вогонь, вибухи, пил, магічні іскри та багато іншого.

Нижче описані деякі основні компоненти та функції, які надає Particle System.

Компонент Emission відповідає за генерацію та випуск частинок у системі. Можна налаштувати швидкість випуску частинок, частоту випуску, життєвий цикл та інші параметри, щоб створити бажаний ефект. Також можна використовувати ключові кадри (keyframes) для динамічної зміни емісії у часі.

Компонент Shape визначає форму та розташування області, з якої генеруються частинки. Можна вибрати різні форми, такі як сфера, прямокутник, коло, конус та багато іншого. Крім того, можна налаштовувати розміри, орієнтацію та кути поширення частинок.

Компонент Lifetime визначає час життя кожної частки у системі. Можна налаштувати мінімальний та максимальний час життя, що дозволяє створювати ефекти з різною тривалістю частинок. Це дозволяє створювати ефекти, такі як частки, що зникають і мерехтять.

Компонент Renderer відповідає за візуалізацію частинок на екрані. Можна вибрати різні режими відображення, такі як точки, спрайти, меші і т. д. Також можна налаштовувати властивості рендерингу, такі як колір, текстури, прозорість та налаштування змішування.

Компонент Force дозволяє задавати зовнішні сили, які впливають частинки. Можна додати різні сили, такі як гравітація, вітер, турбулентність та інші, щоб змінити рух та поведінку частинок у системі.

Компонент Collision дозволяє часткам стикатися з іншими об'єктами в сцені. Можна налаштувати поведінку під час зіткнення, таке як відображення, злиття або знищення частинок. Це дозволяє створювати ефекти взаємодії частинок із довкіллям.

Sub-Emitters також підтримує використання подемітерів, що дозволяє створювати складні ієрархічні системи частинок. Підемітери можуть бути пов'язані з основним емітером та випускати додаткові частинки, створюючи ефекти вибухів, слідів, диму та інші складні ефекти. Particle System надає багатий набір налаштувань та параметрів, які дозволяють створювати різноманітні та унікальні ефекти частинок у ігровому проєкті.

Крім того, він інтегрується з іншими компонентами Unity, такими як анімації, колайдери, освітлення та інші, що дозволяє створювати складніші та інтерактивніші сцени. Реалізація променю світла та телепортації кулі в кінці рівня виконана за допомогою використання ефектів, наданих класом EffectManager:

```
EffectsManager.instance.PlayFinishBlow(_startPos);

public ParticleSystem PlayEffect(Vector3 position){
        if (list == null) list = new List<ParticleSystem>();
        var effect = list.Find(x => x!=null
        && !x.gameObject.activeSelf);
        if (!effect) {
            effect= GameObject.Instantiate(prefab, parent) ;
            list.Add(effect);
        }
        effect.transform.position = position;
        effect.gameObject.SetActive(true);
        return effect;
}
```

4.4 Опис алгоритму генерації рівня

Клас LevelGeneratorController відповідає за генерацію рівнів у грі та включає методи для створення різних об'єктів, таких як труби, перешкоди та монети. Він використовує компоненти даних (структури), визначені у просторі імен Superball, щоб зберігати інформацію про об'єкти рівня.

Принцип генерації рівнів ґрунтується на наступних кроках:

а) у методі Generate відбувається ініціалізація контролера генерації рівня GeneratorController та отримання списку вже існуючих труб pipes;

б) потім викликається метод SetFinish, який встановлює фінішну точку рівня, якщо це дозволено;

в) позраховується довжина рівня levelLength і застосовуються межі рівня і стелі з допомогою методу ApplyBounds і ApplyCeiling;

г) якщо GeneratorController потрібно згенерувати новий рівень то виконуються такі дії:

1) викликається метод ClearLevelObjectsData, який видаляє всі об'єкти рівня;

2) викликається метод SpawnPipes, який створює труби та перешкоди;

3) викликається метод SpawnCoins, який створює монети лише на рівні;

4) викликається метод SpawnSecondaryPipes, що створює додаткові труби.

д) встановлюється поточний рівень як згенерований за допомогою методу SetCurrentLevelAsGenerated.

Якщо в GeneratorController не потрібно генерувати новий рівень, відбувається завантаження вже згенерованого рівня за допомогою методу LoadLevel. Це включає створення труб, перешкод і монет, збережених у стані GeneratorController. Метод SpawnPipes відповідає за створення труб та перешкод. Він використовує різні параметри та налаштування рівня з об'єкта levelConfig для визначення розташування та характеристик об'єктів. У циклі відбувається створення труб з певними координатами та типами, а також можливе створення перешкод. Метод AddPipeRotator додає до труби обертання, якщо умови виконані. Метод LoadLevel завантажує створені труби, монети та перешкоди, використовуючи дані з GeneratorController. Інші методи класу LevelGeneratorController дозволяють отримати інформацію про об'єкти рівня, такі як список труб, монет та перешкод.

4.5 Реалізація введення гравця за допомогою джойстика

Компонент "Joystick" є частиною ігрового інтерфейсу і є елементом управління, що дозволяє користувачеві взаємодіяти з ігровим персонажем або об'єктом через переміщення пальця на екрані або за допомогою клавіатури. "Joystick" відображається на екрані і складається з двох частин: основи (originView) та покажчика (pointerView). Основа являє собою фіксовану позицію, а покажчик переміщається в залежності від введення користувача.

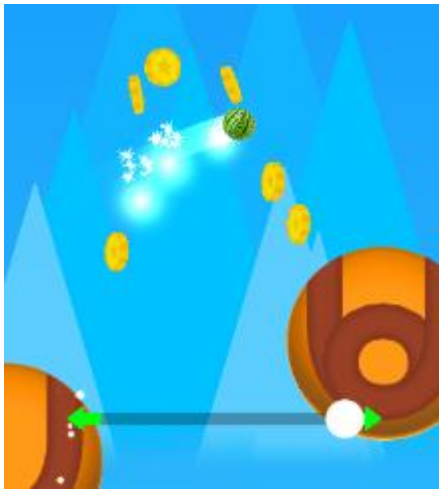

Рисунок 11 – Зовнішній вигляд джойстика

Користувач може переміщати вказівник, торкаючись екрана та перетягуючи його. При цьому основа залишається на своєму місці. Відстань, на яку можна переміщати вказівник, обмежена параметром maxDistance.

Також, джойстик реагує на введення користувача, як за допомогою сенсорного екрана, так і за допомогою клавіатури. При переміщенні покажчика сенсорним екраном або натискання відповідних клавіш на клавіатурі, генеруються події OnBeginDrag, OnDrag та OnEndDrag, які обробляються всередині компонента.

Він генерує події Dragged та DragEnded для оповіщення інших компонентів або скриптів про переміщення покажчика. Подія Dragged передає

вектор переміщення покажчика, а подія DragEnded вказує на завершення перетягування. Крім цього, джойстик має методи Show та Hide, які дозволяють програмно показувати чи приховувати елемент керування на екрані. Компонент також обробляє натискання клавіш на клавіатурі, дозволяючи користувачеві керувати переміщенням вказівника за допомогою стрілок або WASD. При натисканні відповідних клавіш генеруються події OnBeginDrag, OnDrag та OnEndDrag, а покажчик переміщується відповідно до напрямку руху. Загальна ідея використання компонента Joystick полягає в тому, щоб надати гравцеві зручний спосіб управління персонажем або об'єктом у грі. Користувач може переміщати вказівник по обмеженій області і, таким чином, керувати рухом об'єкта в ігровому світі.

4.6 Реалізація збереження варіацій зовнішнього вигляду та монет

Клас PlayerPrefs в Unity надає функціональність для збереження та завантаження даних гри. Він використовується для зберігання ключ-значення пар у постійній пам'яті пристрою. Клас PlayerPrefs дозволяє зберігати різні типи даних, такі як цілочисельні значення, значення з плаваючою комою та рядки. Щоб зберегти дані, необхідно вказати ключ та відповідне значення. Ключ використовується для ідентифікації збережених даних, і з нього можна звернутися до збереженого значення в майбутньому. При завантаженні даних ви вказуєте ключ і PlayerPrefs повертає збережене значення, пов'язане з цим ключем. Клас PlayerPrefs дозволяє перевіряти наявність збережених даних по ключу та видаляти збережені дані, якщо це необхідно [9].

Завдяки використанню PlayerPrefs, ігровий прогрес, налаштування гри, досягнення та інші дані можуть бути збережені та завантажені в наступних сеансах гри, забезпечуючи збереження та відновлення стану гри для користувачів. Використання PlayerPrefs дозволяє спростити процес збереження та завантаження даних гри без необхідності вручну керувати файлами та структурами даних. Реалізація збереження монет виглядає так:

```
private void Start() {
  coinAmount = PlayerPrefs.GetInt("Coins");
  rb = GetComponent<Rigidbody2D>();
}

private void OnTriggerEnter2D(Collider2D collision) {
  coinAmount = PlayerPrefs.GetInt("Coins");
  coinAmount = ball.jumpCounter % 5 == 0 ? coinAmount += 3 : coinAmount += 1;
  PlayerPrefs.SetInt("Coins", coinAmount);
}
```

ВИСНОВКИ

У ході розробки крос-платформної програми були виконані основні завдання, які дозволили досягти поставленої мети та створити успішний продукт. Було використано сучасний стек технологій, який забезпечив ефективне та масштабоване рішення. На початку розробки було визначено основні ігрові механіки та концепцію гри.

У процесі аналізу вимог були визначені функціональні та нефункціональні характеристики програми, що відіграло ключову роль у визначенні його архітектури. Було обрано підхід крос-платформної розробки, який дозволив забезпечити сумісність з різними операційними системами та пристроями. Це дозволило розширити аудиторію програми та підвищити його конкурентоспроможність на ринку.

Одним із важливих аспектів розробки крос-платформної гри було забезпечення сумісності з різними операційними системами та пристроями. Це дозволило гравцям запускати гру на різних платформах та пристроях, розширюючи її аудиторію та забезпечуючи доступність для максимально великої кількості гравців.

Гра "Superball" має інтуїтивно зрозумілий інтерфейс і зручне управління, що забезпечує приємну і комфортну взаємодію гравців з грою. Простота та зрозумілість управління сприяють легкому освоєнню гри. У ході розробки були використані інструменти для створення інтерфейсу користувача, які забезпечують зручність та інтуїтивно зрозумілий досвід використання. Це включало розробку сучасного та естетично привабливого дизайну, а також оптимізацію інтерфейсу для різних екранів і дозволів.

Розробка гри здійснювалася з орієнтацією на оптимізацію продуктивності. Це дозволяє використовувати продукт без затримок або зниження якості роботи. Крім того, було реалізовано інтеграцію з різними зовнішніми сервісами та API, що дозволяє розширити функціональність програми та забезпечити більш корисний досвід використання для

користувачів. У ході тестування програми були виявлені та виправлені помилки та недоліки, що забезпечило високу стабільність та надійність роботи.

У результаті, гра "Superball" є захоплюючим і якісним ігровим продуктом, який може бути запущений на різних платформах. Розробка гри демонструє успішне застосування крос-платформного підходу, забезпечуючи доступність та задоволення потреб гравців.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Smith, J. Unity Game Development in 24 Hours. Indianapolis, IN: Sams Publishing, 2017. 464 с.
2. Harrison, H. Unity in Action: Multiplatform Game Development in C#. Shelter Island, NY: Manning Publications, 2018. 496 с.
3. Троелсен А., C# 8.0 та платформи .NET Core та .NET Standard. М.: Вільямс, 2019. 1040 с.
4. Албахарі Дж., Програмування C# 8 і .NET Core 3.0. М.: ДМК Прес, 2019. 928 с.
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA: Addison-Wesley Professional, 1994. 395 с.
6. Брейтенберг Ст, Алгоритми ігрового проєктування. М.: Вільямс, 2019. 320 с.
7. Martin, R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston, MA: Prentice Hall, 2017. 432 с.
8. Ferrone, H. Learning C# by Developing Games with Unity. Birmingham, UK: Packt Publishing, 2018. 828 с.
9. Sherrod, D. Mastering Unity 2D Game Development. Birmingham, UK: Packt Publishing, 2016. 398 с.