МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускної кафедри

Аліна САВЧЕНКО

"____" _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВРА"**

Тема: " Технологія моделювання взаємодії об'єктів у ігровому середовищі Unity"

Виконавець: студент групи УС-413Б Дерда Іван Володимирович

Керівник: проф. каф. Воронін Альберт Миколайович

Нормоконтролер: Олександр ШЕВЧЕНКО.

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології, 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ

Завідувач випускної кафедри

Аліна САВЧЕНКО

"___" _____ 2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи (проєкту)

Дерди Івана Володимировича

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Технологія моделювання взаємодії об'єктів у ігровому середовищі Unity» затверджена наказом ректора №517/ст. від 05.04.2024.
2. **Термін виконання роботи:** 06.05.2024 – 10.06.2024.
3. **Вихідні дані до роботи:** 3D моделі об'єктів, текстури та матеріали об'єктів, готові анімації, документація середовища Unity, редактор коду Visual Studio.
4. **Зміст пояснювальної записки :** Аналіз мов програмування (C#, C++, Візуальне програмування) та їхніх переваг/недоліків для розробки в Unity. Порівняння технологій та фреймворків, таких як Unity, Unreal Engine. Огляд наявних ігрових продуктів та технологій моделювання взаємодії об'єктів у них
5. **Перелік обов'язкового графічного матеріалу:** скіншоти інтерфейсу, структура дерева анімацій ігрового застосунку, слайди презентації MS PowerPoint.

6. Календарний план-графік

№ з/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Аналіз та дослідження предметної області	06.05.2024р. – 11.05.2024р.	
2	Опрацювання літературних джерел, що відносяться до теми дипломного проекту	12.05.2024р. – 16.05.2024р.	
3	Аналіз наявних ігрових продуктів та технологій моделювання взаємодії об'єктів у них	17.05.2024р. – 18.05.2024р.	
4	Розробка структури дипломного проекту відповідно до обраної теми	19.05.2024р. – 20.05.2024р.	
5	Розробка функціональної частини проекту та інтерфейсу користувача	21.05.2024р. – 26.05.2024р.	
6	Оформлення пояснювальної записки	27.05.2024р. – 29.05.2024р.	
7	Підготовка презентації та доповіді	30.05.2024р. – 06.05.2024р.	

7. **Дата видачі завдання:** 06. травня .2024 р.

Керівник дипломного проекту Альберт ВОРОНІН
(підпис керівника)

Завдання прийняв до виконання Іван ДЕРДА
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Технологія моделювання взаємодії об’єктів у ігровому середовищі Unity» представлена на 54 сторінках, містить 35 рисунків, 8 наукових джерел.

Мета дипломного проекту: Розробити багатокористувацький ігровий застосунок з реалістичною взаємодією об’єктів, їх реакцією на зіткнення, удари та поштовхи з використанням технології моделювання взаємодії об’єктів у ігровому середовищі Unity.

Об'єкт дослідження: процес моделювання взаємодії об’єктів.

Предмет дослідження: методи симуляції взаємодії об’єктів в середовищі розробки Unity.

Метод дослідження: Аналіз літературних джерела, порівняльний аналіз аналогів, наявних ігрових продуктів та технологій моделювання взаємодії об'єктів у них та реалізація реалістичної взаємодії об’єктів у ігровому середовищі Unity.

Наукова новизна полягає в розробці ігрового застосунку на основі актуальних методів моделювання взаємодії об’єктів в двигуну Unity, з врахуванням сучасних тенденцій та вимоги індустрії. Розроблений додаток оптимізовано для максимальної продуктивності та з можливостю гри в багатокористувацькому режимі.

Результат проєкту: Розроблено ігровий застосунок з підтримкою мультиплеєру в жанрі королівської битви та перегонів, в якому реалізовано реалістичну взаємодію між об’єктами та гравцями. Тематика гри взята з шоу “WIPEOUT”.

МОДЕЛЮВАННЯ, UNITY, ПРОЦЕС ВЗАЄМОДІЇ ОБ’ЄКТІВ, РЕАЛІСТИЧНА ВЗАЄМОДІЯ, МОДЕОІ ОБ’ЄКТІВ, 3D ГРАФІКА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 7

ВСТУП..... 8

РОЗДІЛ 1. АКТУАЛЬНІСТЬ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ РОЗРОБКИ 9

1.1. Аналіз предметної області..... 9

1.2. Огляд наявних ігрових продуктів..... 15

1.3. Формування основних завдань та вимог 16

1.4. Висновки до розділу 18

РОЗДІЛ 2. АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ 19

2.1. Аналіз мов програмування (C#, C++, Візуальне програмування) 19

2.2. Порівняння фреймворків, таких як Unity, Unreal Engine..... 20

2.3. Визначення основних інструментів, необхідних для реалізації продукту(Unity Engine, Blender 3D, Adobe Photoshop)..... 22

2.4. Методи тестування та налагодження..... 22

2.5. Висновки до розділу 24

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОТОТИПУ СИСТЕМИ 25

3.1. Архітектура системи..... 25

3.2. Реалізація моделювання взаємодії об'єктів 30

3.3. Реалізація багатокористувацької складової 43

3.4. Тестування та налагодження..... 47

3.4. Висновки до розділу 51

ВИСНОВКИ..... 53

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ..... 55

ДОДАТКИ..... 56

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UE	Unreal Engine Двигун для розробки ігор.
AAA	Термін для класу високо бюджетних комп'ютерних ігор.
VR	Ілюзія дійсності, створювана за допомогою комп'ютерних систем, які забезпечують зорові, звукові та інші відчуття.
AR	Є результатом введення будь-яких сенсорних даних в поле зору зметою тдоповнення інформації про навколишнє середовище і зміни сприйняття навколишнього середовища.
Coll	скорочення компоненту Collider колайдер.
Rb	скорочення компоненту Rigidbody.

ВСТУП

На сьогоднішній день ігри відіграють важливу роль в нашому житті. Раніше ігровий процес був доступний лише на стаціонарних комп’ютерах, але завдяки стрімкому розвитку технологій в них можна грати не лише на комп’ютері дома але й на своєму мобільному телефоні, коли нам потрібно скоротати час або на відпочинку а то й навіть у дорозі. Реалізація реалістичної взаємодії об’єктів є не аби яким важливим завданням, яке має великий вплив на привернення уваги гравців. Тому в сучасних продуктах на відміну від старих присутня максимальна взаємодія з інтерактивним середовищем ігрового світу. Графічна складова ігор теж не стоїть на місці, в AAA чи VR проектах іноді важко відрізнити ігровий процес від реального життя. Усі ігри були зроблені завдяки цілим командам, не тільки розробниками. Команда складається з керівника проекту, розробників, дизайнерів, художників та клієнта, який замовив цей продукт.

Успішне виконання даного завдання потребує здійснення наступних кроків:

- огляд наявних ігрових продуктів та технологій моделювання взаємодії об'єктів у них;
- визначення основних інструментів та технологій, необхідних для реалізації продукту;
- проектування і реалізації основних функціональних можливостей системи.

Практичне значення отриманих результатів. Реалізований в процесі дипломного проектування прототип дозволяє користувачам наглядно побачити та відчути фізичну взаємодію об’єктів. Також мною було покращено свої технічні навички у програмуванні, дизайні та проектуванні ігор. Інноваційні підходи до вирішення проблем, пов'язаних з реалізацією фізичних взаємодій та мережевої синхронізації, можуть бути застосовані в інших проектах.

РОЗДІЛ 1
АКТУАЛЬНІСТЬ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ РОЗРОБКИ

1.1. Аналіз предметної області

В епоху стрімкого розвитку технологій та зростання популярності відеоігор, розробка якісного ігрового контенту стає все більш важливою. В даний момент ігрові магазини перенаповнені не якісними ігровими застосунками, які не аби як впливають на сприйняття ігор суспільством. Unity, як одна з найпопулярніших платформ для розробки ігор, має великий набір інструментів для реалізації фізичної взаємодії об’єктів у ігровому середовищі. Кваліфікаційна робота полягає в розробці ігрового застосунку на основі актуальних методів моделювання взаємодії об’єктів в двигуну Unity, з врахуванням сучасних тенденцій та вимоги індустрії. Розроблений додаток оптимізовано для максимальної продуктивності та з можливостю гри в багатокористувацькому режимі.

Unity є однією з найбільш популярних платформ для створення ігрових додатків, в якій є можливість роботи як з 2D так і з 3D графікою. Невід’ємною частиною розробки ігор є моделювання взаємодії об’єктів у ігровому середовищі. В цей процес входять фізичні взаємодії, анімації, логіка поведінки гри, що забезпечують реалістичну та захопливу атмосферу гри [1].

Фізичний рушій Unity дозволяє моделювати реалістичну взаємодію об’єктів за допомогою стандартних компонентів. Таких як:

- Rigidbodi. Додає об’єктам можливіть відчувати вплив фізичних сил.
- Colliders. Визначає фізичні рамки об’єкта для обробки зіткнень.

- Forces. Забезпечують вплив сил, таких як гравітація, тертя, пружність.
- Joints. Дозволяє імітувати шарніри чи суглоби для досягання реалістичної поведінки об’єктів.

Реалізація взаємодії об’єктів включає такі аспекти як:

- Колізії та Тригери. Unity використовує колайдери для визначення зіткнень між об’єктами.
- Анімації. Анімації додають реалістичності руху об’єктів. Компонент Animator в Unity дозволяє створювати складні дерева анімацій для керування об’єктами.
- Скрипти. Скрипти на мові C# дозволяють задавати поведінку об’єктів та реакції на події, такі як натиснення на кнопку чи виконання певних умов [1].

Рушій Unity не буде обраховувати фізичну взаємодію об’єктів якщо до них не додати компоненти фізики: Rigidbody та Collider. Взагалі компоненти визначають поведінку об'єктів у грі. Вони є функціональною частиною кожного ігрового об'єктах.

Ігровий об'єкт – це контейнер для різних компонентів. За замовчуванням всі ігрові об'єкти мають компонент Transform. Тому що Трансформація визначає, де знаходиться ігровий об'єкт, як він обертається і масштабується. Без компонента Transform ігровому об'єкту не буде місця у світі.

Rigidbody. Компоненти Rigidbody дозволяють внутрішньо ігровим об'єктам взаємодіяти з фізикою. Для реалістичного руху об’єктів на які впливають сили (таб. 1.1). Всі ігрові об’єкти повинні мати компонент Rigidbody, щоб піддаватися гравітації, діяти відповідно до заданих за сценарієм сил або взаємодіяти з іншими об'єктами через фізичний двигун NVIDIA PhysX [1].

Через те, що компонент Rigidbody керує рухом об'єкта, на якому він закріплений, не потрібно намагатися керувати об'єктом з коду, змінюючи властивості компоненту Transform, такі як положення та обертання. Замість цього

потрібно застосовувати сили, щоб штовхати чи обертати об'єкт і дозволити ігровому двигуну обчислити результати операцій.

Коли об’єкт з Rigidbody рухається із дуже малою швидкістю то фізичний двигун припускає, що об’єкт зупинився і перебуває в стані спокою. У цьому випадку об'єкт не буде рухатися знову, поки його не потурбують доклавши силу чи штувхнувши. Це все робиться для оптимізацій для того, щоб об'єкт не споживав ресурси процесора до тих пір, поки його знову не приведуть в рух.

Властивості Компоненту Rigidbody

Таблиця 1.1

Властивість	**Функція**
Mass	Маса об'єкта в кілограмах.
Drag	Який опір повітря чиниться на об'єкт під час його руху під дією цих сил. 0 означає, що опору немає, а нескінченність відразу зупиняє рух об'єкта.
Angular Drag	Який опір повітря чиниться на об'єкт під час його обертання під дією обертальної сили. 0 означає відсутність опору. Зауважте, що ви не можете зупинити обертання об'єкта, встановивши його кутове перетягування на нескінченність.
Use Gravity	При включенні на об'єкт діє сила тяжіння.
Is Kinematic	При включенні об'єкт не буде управлятися фізичним движком, а може управлятися тільки його перетворенням. Корисно під час переміщення платформ або якщо вам потрібно анімувати твердий тіло, якому призначено шарнірне з'єднання.

Продовження таблиці 1.1

Interpolate	Можна вибрати один з варіантів: • Інтерполяція не застосовується. • Згладжування трансформацій ґрунтується на перетворенні з попереднього кадру. • Згладжування трансформацій ґрунтується на наближеному перетворенні наступного кадру.
Collision Detectoin	Використовується для запобігання проникненню об'єктів, що швидко рухаються, на інші об'єкти без виявлення зіткнень.
Freeze Position	Вибірково зупиняє рух твердого тіла по осях X, Y і Z.
Freeze Rotation	Зупиняє обертання твердого тіла навколо локальних осей X, Y та Z вибірково.

Colliders. Компоненти Colliders визначають фізичну форму об'єкта гри з метою фізичних зіткнень. Колайдер, який є невидимим, не обов'язково має бути точно такої ж форми, як і GameObject. Приблизне наближення сітки часто є більш ефективним і непомітним в ігровому процесі [1].

Складні колайдери наближаються до форми ігрового об'єкта, зберігаючи при цьому низькі витрати на процесор. Щоб отримати додаткову гнучкість, можна додати додаткові колайдери до дочірніх ігрових об'єктів. Наприклад, ви можете обертати блоки відносно локальних осей батьківського об'єкта гри. Коли ви створюєте складений колайдер слід використовувати лише один компонент Rigidbody, розміщений у кореневому об'єкті GameObject у ієрархії .

Примітивні колайдери некоректно працюють зі зсувними перетвореннями. Якщо ви використовуєте комбінацію обертань і неоднорідних масштабів в ієрархії «Трансформування» так, що отримана фігура більше не є примітивною фігурою, примітивний колайдер не зможе представити її правильно.

Однак є деякі випадки, коли навіть складні колайдери недостатньо точні. У 3D можна використовувати Mesh Colliders, щоб точно відповідати формі сітки

GameObject. У 2D Polygon Collider 2D не ідеально відповідає формі спрайтової графіки, але ви можете вдосконалити форму до будь-якого рівня деталізації, який вам подобається.

Ці колайдери набагато більш створюють навантаження для процесора, ніж примітивні типи, тому використовувати їх потрібно помірно для підтримки хорошої продуктивності. Крім того, Mesh Collider не може зіткнутися з іншим Mesh Collider. У деяких випадках це можна обійти, позначивши Mesh Collider як Convex. Це генерує спрощену форму колайдера, яка схожа на оригінальну сітку, але з заповненими отворами.

Перевага цього полягає в тому, що Convex Mesh Collider може зіткнутися з іншими Mesh Collider, тому можна використовувати цю функцію, коли є рухомий персонаж з відповідною формою. Однак, хорошим правилом є використання Mesh Collider для геометрії сцен та наближення форми рухомих об'єктів гри за допомогою складних примітивних колайдерів .

Для створення не рухомих об’єктів на рівні таких як стіни чи підлога використовують лише Colliders без додавання компоненту Rigidbody. Такі об’єкти називають статичними і вони мають статичні колайдери. А Collider на ігровому об'єкті, що має ще й Rigidbody, відомі як динамічні. Статичні колайдери можуть взаємодіяти з динамічними колайдерами, але оскільки вони не мають RigidBody, вони не рухаються при зіткненні (таб. 1.2.).

Властивості Компоненту Collider

Таблиця 1.2

Влстивість	**Функція**
Is Trigger	Ввімкнений параметр використовується для запуску подій OnTrigger і ігнорується фізикию рушія.

Продовження таблиці 1.2

Влстивість	Функція
Material	Фізичний матеріал відповідає не за зовнішній вигляд а за симуляцію матеріалу з якого об’єкт зроблено, що забезпечити реалістичну взаємодію з іншими колайдерами.
Center	Відповідає за розташування колайдера в локальних координатах об'єкта.
Для Box Collider	
Size	Відповідає за розміри колайдера в просторі X,Y та Z.
Для Capsule Collider	
Radius	Відповідає за радіус(ширину) колайдера в локальному просторі об'єкта.
Height	Відповідає за висоту колайдера.
Direction	Відповідає за напрям куда буде направлена висота (X, Y, Z).
Для Sphere Collider	
Radius	Розмір колайдера.

На поведінку об’єктів також впливають і фізичні матеріали. Коли колайдери взаємодіють, їх поверхні повинні імітувати властивості матеріалу, з якого вони теоретично повинні бути виготовлені. Наприклад, шар льоду буде більш слизьким, тоді як гумовий м'яч забезпечить більше тертя і буде дуже пружним. Хоча форма колайдерів не деформується під час зіткнень, їх тертя та пружність можна налаштувати за допомогою Physics Materials. Ви можете регулювати параметри так, як вам хочеться, методом проб і помилок, але, наприклад, у крижаного матеріалу буде нульове тертя, в той час як гумовий матеріал буде мати високий індекс тертя і практично ідеальну пружність.

1.2. Огляд наявних ігрових продуктів

За до помогою ігрового двигуна Unity створено безліч різних ігор, але можна виділити декілька прикладів де фізика відіграє головну роль в проекті, через що вони і стали популярними серед користувачів. Список ігрових продуктів:

- Fall Guys;
- Human Fall Flat;
- Party Animals;
- Stick Fight: The Game.

Давайте подивимося на гру Fall Guys. Це багатокористувацька гра, яка поєднує в собі елементи платформера і королівської битви. В одній грі можуть брати участь до 60 гравців, керуючи забавними персонажами у формі бобів. Кожна гра складається з кількох раундів, у яких гравці повинні виконати певні завдання. Ті, хто не виконує умови гри, вибувають.

Міні-ігри можуть бути індивідуальними або колективними, що вимагають спільних зусиль гравців. Кількість команд може варіюватися від двох до чотирьох, а завдання можуть мати різний рівень перемог і поразок. Наприклад, гравцям може знадобитися дійти до фінішу першим або не вибути з рівня певного часу, уникаючи пасток або забивши кілька голів іншій команді.

У Fall Guys гравці повинні подолати різні перешкоди, щоб дістатися до фінішу. Багато з цих перешкод було взято з класичної японської гри Takeshi’s Castle. Щоразу, коли персонаж стикається з перешкодою, він або вона може втратити рівновагу, впасти або навіть бути збитим.

Humanity: Fall Flat — це гра квест і платформер. У Humans: Fall Flat немає сюжету, замість цього гравцям пропонується перевірити свої вміння розгадувати нестандартні головоломки. Складність головоломок походять від незручних і смішних елементів керування, які пропонує гра. У цій грі гравці повинні покладатися на свою кмітливість і фізику, щоб досліджувати навколишнє середовище і розгадувати фізичні головоломки, щоб уникнути цікавих і складних

рівнів. У грі є нестандартне управління персонажами, і це тому, що герої цілком фізичні і реагують на взаємодію кожною частиною тіла.

Party Animals — це змагальна гра файтинг, заснована на фізиці, де гравці грають за різних тварин, включаючи цуценят, кошенят, качок, кроликів, акул, динозаврів і навіть єдинорогів. Тварини можуть бити кулаками, підкидати, стрибати, штовхати ногами та бити головою один одного. Також вони здатні підібрати різноманітну зброю. Отримавши певну кількість шкоди, гравці тимчасово вибиваються. У більшості випадків вони відновлюються і повертаються в бій, якщо тільки їх не викидають з карти або в різні пастки.

Stick Fight - це змагальна гра файтинг, де ви граєте за фігурку чоловічка та використовуєте її, щоб битися з чотирма іншими гравцями на арені з перспективою з боку, де вам також потрібно долати перешкоди, використовуючи принцип платформера. Графіка заснована на ретро-грі, де фігурки паличок керуються фізикою тр'япкової ляльки і представлені кольоровим маркером. Для бою кулаки можна використовувати зі спеціальними комбо або різною зброєю. Ви можете грати проти інших гравців через Інтернет, через підключення до локальної мережі або на системі через режим розділеного екрана.

1.3. Формування основних завдань та вимог

Проаналізувавши аналогічні застосунки, розроблені на Unity, можна виділити спільні риси притаманні всім проєктам, а саме – фізична взаємодія героя з оточенням та подолання перешкод на його шляху.

Зважаючи на зазначені вище результати огляду та порівняльного аналізу існуючих аналогів, використання технології моделювання взаємодії об'єктів для розробки ігрового додатку є актуальною.

Метою кваліфікаційної роботи є розробка багатокористувацького ігрового застосунку з реалістичною взаємодією об'єктів, їх реакцією на зіткнення, удари та поштовхи з використанням технології моделювання взаємодії об'єктів у ігровому середовищі Unity.

Відповідно до поставлено мети в роботі потрібно вирішити такі задачі:

– дослідження технології та методів моделювання взаємодії об’єктів;

– створення 3D моделей для взаємодії;

– створення анімацій для об’єктів;

– реалізація інтерфейсу користувача;

– розробка технології для ігрового застосунку оптимізованого для максимальної продуктивності та з можливостю гри в багатокористувацькому режимі;

– тестування та налагодження систем.

Проаналізувавши вище надану інформацію можна сформувати наступні завдання та вимоги до моделювання взаємодії об'єктів:

- Реалістична фізика: забезпечення реалістичної поведінки об'єктів відповідно до принципів фізики, таких як гравітація, тертя, зіткнення тощо.
- Стійкість та оптимізація: забезпечення стабільної роботи гри навіть при великій кількості взаємодіючих об'єктів, а також оптимізація продуктивності для підтримки швидкості та ефективності гри.
- Колізії та зіткнення: коректне оброблення колізій між об'єктами та реалістичне відтворення їхніх зіткнень.
- Інтерактивність: можливість взаємодії гравця з об'єктами у грі, включаючи можливість пересування, обертання, взаємодію зі світлом, звуком тощо.
- Взаємодія з оточуючим середовищем: реалістична реакція об'єктів на зміни в середовищі, такі як зміна погоди, освітлення, руйнування тощо.
- Врахування контексту гри: моделювання взаємодії об'єктів відповідно до специфіки конкретної гри та її геймплею.
- Тестування та налагодження: проведення тестування системи взаємодії об'єктів для виявлення та виправлення можливих помилок та недоліків.

Розроблюваний проєкт має переваги, що дозволяють усунути недоліки інших ігрових застосунків, а саме:

- Просте та інтуїтивне керування;

Перевага: Інтуїтивно зрозуміла система керування, яка легко освоюється як новачками, так і досвідченими гравцями. Інтерфейс та механіки управління будуть налаштовані таким чином, щоб нові гравці одразу могли зрозуміти логіку керування.

- Збалансованість персонажів;

Перевага: Потрібно провести ретельне тестування і налаштування персонажів, щоб забезпечити їх рівноправність у грі.

- Простота графіки;

Перевага: Гра буде мати графіку з текстурами та анімаціями, що створює захоплюючий візуальний досвід. Використання сучасних графічних технологій дозволить досягти високого рівня естетичної привабливості.

- Стабільне мережеве з’єднання;

Перевага: Забезпечення стабільної та надійної мережевої інфраструктури, що мінімізує затримки і проблеми з підключенням. Використання сучасних серверних технологій та оптимізація мережевих протоколів забезпечать безперебійну гру.

- Розширені можливості кастомізації героя;

Перевага: Гра пропонуватиме широкий вибір налаштувань для персонажів, включаючи різноманітні костюми та кольори. Це дозволить гравцям створювати унікальних персонажів відповідно до своїх уподобань.

1.4. Висновки до розділу

У розділі було проведено аналіз предметної області та визначено задачі, які є актуальними в реалізації взаємодії об’єктів в ігрових середовищах.

В результаті огляду предметної області та аналізу вже створених ігрових продуктів було визначено перелік актуальних технологій та методів моделювання взаємодії об’єктів.

На основі отриманих даних було проведено постановку задач для розроблюваного програмного продукту та визначено функціональні вимоги.

РОЗДІЛ 2
АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1. Аналіз мов програмування (C#, C++, Візуальне програмування)

При розробці ігрових додатків важливим кроком є вибір мови програмування, яка дозволить максимально ефективно спростити процес розробки ігрової логіки. Найпопулярніші мови для розробки ігор це C#,C++ та візуальне програмування.

Таблиця 2.1

Критерії порівняння	**C#**	**C++**	**Візуальне програмування**
Простота та читабельність	Висока: легкий для вивчення	Середній: вимагає більше часу для вивчення	Висока: не потребує знань програмування
Продуктивність	Висока	Дуже висока	Середня
Управління пам’яттю	Автоматичне	Ручне	Не потрібне
Інтеграція з платформами	Відмінна	Відмінна	Відмінна
Бібліотеки та інструменти	Багато бібліотек, Хороша підтримка	Дуже багато бібліотек, відмінна підтримка	Залежить від середовища розробки
Гнучкість	Висока	Дуже висока	Обмежена

Продовження таблиці 2.1

Критерії порівняння	**C#**	**C++**	**Візуальне програмування**
Швидкість розробки	Швидка	Повільніша	Дуже швидка
Підтримка та розширюваність	Висока	Середня	Середня
Платформа залежність	Переважно .NET, хороша кросплатформеність з .NET Core	Портативна, доступна для більшость платформ	Залежить від середовища розробки
Застосування	Ігрова логіка, інтеграція з Unity	Розробка двигунів, Високопродуктивних систем	Прототипування, Менш складні ігрові проекти

Проаналізувавши таблицю 2.1 можна дійти до наступних висновків:

- C#. Рекомендується використовувати для більшості проектів на Unity, через її простоту, гарну інтеграцію та достатню продуктивність [2].
- C++. Ідеально підійде при розробці високопродуктивних систем та ігрових рушіїв, де важливо контролювати ресурси [2].
- Візуальне програмування найкраще підходить для швидкого створення прототипу, навчання чи не складних проектів, завдяки своїй зрозумілості та відсутності потреби розуміння програмування.

2.2. Порівняння фреймворків, таких як Unity, Unreal Engine

Одні з найпопулярніших середовищ розробки ігор є Unity та UE.

Таблиця 2.2

Критрії	Unity	UE
Мова програмування	C#, Візуальне програмування	C++, Візуальне програмування
Продуктивність	Висока	Дуже висока
Графічні можливості	Хороша	Відмінна
Інтерфейс користувача	Простий	Складний
Навчання	Легке	Складне
Кросплатформеність	Відмінна	Відмінна
Підтримка	Велика, орієнтована для новачків та професіоналів	Велика, орієнтована на професіоналів
Ціна	Безкоштовний з обчеженнями	Безкоштовний з обчеженнями
Масштабність проектів	Малі – середні	Ідеальний для великих
Фізика	Потужна	Дуже потужна
Бібліотеки та плагіни	Широкий вибір	Широкий вибір
Документація	Присутня	Присутня
Інструменти для розробки	Вбудовані	Вбудовані

Проаналізувавши таблицю 2.2 можна дійти висновку що:

- Unity : Переваги – Легкий для вивчення, зрозумілий інтерфейс, добре підходить до інді-розробок і малих студій, великі можливості.
  Недоліки – графіка на високому рівні, має обмеження у високопродуктивних проектах.
- UE : Переваги – Потужна графіка, висока продуктивність, ідеальний для великих проектів та компаній. Недоліки – складний для новачків, складність навчання через C++ та складний інтерфейс.

2.3. Визначення основних інструментів, необхідних для реалізації продукту(Unity Engine, Blender 3D, Adobe Photoshop)

В процесі виконання завдання окрім використання Unity для розробки продукту, ще потрібно використати Blender 3D та Adobe Photoshop.

Blender 3D це програмний продукт з відкритим доступом для створення 3Д моделей, анімацій, тектуренгу та редагування відео. За допомогою цього інструмента було створено 3D моделі для завдання [3].

Adobe Photoshop це програмний продукт призначений для створення та редагування графічних зображень. За допомогою цього інструмента було створено графіку для інтерфейсу користувача.

Visual Studio це середовище розробки, створене компанією Microsoft. Воно підтримує безліч мов програмування, включаючи C#, C++, Python, JavaScript. Visual Studio надає розробникам широкий набір інструментів для написання, редагування, налагодження та тестування коду, що робить процес розробки більш ефективним та зручним.

2.4. Методи тестування та налагодження

Процес тестування та налагодження ігрового додатку включа6 в себе кілька важливих етапів, які охоплювали як автоматичне, так і ручне тестування різних аспектів програмного забезпечення.

Автоматичне тестування. Автоматичне тестування було ключовим елементом процесу забезпечення якості програмного забезпечення. Використовуючи цей метод, вдалося виявити та виправити основні помилки на ранніх стадіях розробки. Основні види автоматичного тестування, що були застосовані:

- Модульне тестування: Перевірка окремих модулів або компонентів програми для забезпечення їх коректної роботи.

- Інтеграційне тестування: Тестування взаємодії між різними модулями, щоб впевнитися, що вони працюють разом без проблем.
- Регресійне тестування: Перевірка, що нові зміни або функції не порушують існуючий функціонал програми.

Автоматичне тестування забезпечить стабільність основних функцій програми та дозволило швидко знаходити помилки після внесення змін.

Ручне тестування. Ручне тестування доповнтить автоматичне тестування та буде спрямоване на перевірку зручності використання ігрового додатку з точки зору кінцевого користувача. Основні види ручного тестування включали:

- Функціональне тестування: Перевірка роботи всіх функцій програми згідно з технічними вимогами.
- Нефункціональне тестування: Перевірка аспектів продуктивності, зручності використання, надійності та безпеки.
- Тестування користувацького інтерфейсу: Перевірка зручності та інтуїтивності користувацького інтерфейсу, виявлення проблем, які можуть заважати користувачам.

Ручне тестування дозволить отримати зворотний зв'язок від користувачів та внесло важливі корективи в дизайн та функціональність програми.

Налагодження. Процес налагодження включатиме кілька етапів, спрямованих на забезпечення відповідності ігрового додатку очікуванням користувачів та технічним вимогам. Основні етапи налагодження включали:

- Ідентифікація та виправлення помилок: Після кожного етапу тестування, можуть виявитися помилки.
- Оптимізація продуктивності: Проведуться заходи для покращення продуктивності.

2.5. Висновки до розділу

У вище зазначеному розділі було проведено аналіз мов програмування та платформ для розробки ігор, а також обрано основні інструменти для реалізації завдання.

По-перше, було проведено огляд і порівняння поширених мов програмування, які є основними в розробці ігрових продуктів. На основі проведеного огляду і порівняльного аналізу було обрано мову C# для розроблюваного продукту, через її простоту, гарну інтеграцію та достатню продуктивність.

По-друге, було розглянуто найпопулярніші платформи, які використовують при розробці ігрових застосунків. На основі проведеного огляду та порівняльного аналізу різних платформ для розробки було обрано Unity Engine. Томущо, він егкий для вивчення, має зрозумілий інтерфейс, добре підходить до інді-розробок і малих студій, має великі можливості для розробки ігрових застосунків.

Базуючись на отриманих результатах було обрано допоміжні інструменти для розробки, а саме Blender 3D, Visual Studio та Adobe Photoshop. Обрані програми через їх простоту та зрозумілість інтерейсу для нових користувачів.

Процес тестування та налагодження включить у себе як автоматичне, так і ручне тестування різних аспектів ігрового додатку. Автоматичне тестування допоможе виявити та виправити основні помилки та забезпечило стабільність основних функцій. Ручне тестування дозволить перевірити зручність використання ігрового додатку з точки зору кінцевого користувача.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОТОТИПУ СИСТЕМИ

3.1. Архітектура системи

Система моделювання взаємодії об'єктів у ігровому середовищі Unity складається з кількох основних компонентів, кожен з яких відіграє ключову роль у забезпеченні функціонування та взаємодії об'єктів [1]. Ось опис цих компонентів та їхньої взаємодії:

- Сцена (Scene);

Опис: Сцена являє собою контейнер для всіх об'єктів гри, таких як персонажі, декорації, світло, камери та інші елементи.

Взаємодія: Всі інші компоненти розташовуються на сцені та взаємодіють між собою через неї.

- Об'єкти (GameObjects);

Опис: Основні елементи гри, які можуть мати різні форми, розміри та функції, такі як персонажі, предмети, елементи середовища тощо.

Взаємодія: Об'єкти взаємодіють один з одним через фізичні та логічні компоненти, визначені на них.

- Компоненти (Components);

Опис: Скрипти та модулі, які додаються до об'єктів для визначення їхніх властивостей та поведінки. Основні компоненти включають Transform, Renderer, Collider, Rigidbody, Scripts.

Взаємодія: Компоненти керують поведінкою об'єктів, дозволяючи їм рухатися, зіткатися, відображатися на екрані тощо. Наприклад, компоненти Collider і Rigidbody взаємодіють для обробки фізичних зіткнень.

- Скрипти (Scripts);

Опис: Код, написаний на C# або іншій мові програмування, який визначає логіку поведінки об'єктів.

Взаємодія: Скрипти використовуються для контролю анімацій, фізичних взаємодій, реакції на події користувача, штучного інтелекту та інших аспектів гри. Вони можуть звертатися до інших компонентів та змінювати їхні параметри.

- Фізичний рушій (Physics Engine);

Опис: Відповідає за моделювання фізичних явищ, таких як гравітація, зіткнення, тертя та інші.

Взаємодія: Використовується компонентами Collider та Rigidbody для реалістичної симуляції руху та зіткнень об'єктів у грі.

- Система анімації (Animation System);

Опис: Відповідає за анімацію об'єктів та персонажів.

Взаємодія: Анімації інтегруються зі скриптами та іншими компонентами, щоб забезпечити плавні та реалістичні рухи об'єктів. Наприклад, анімації можуть запускатися скриптами в залежності від дій гравця.

- Система вводу (Input System);

Опис: Відповідає за обробку вводу від користувача, включаючи клавіатуру, мишу, геймпади тощо.

Взаємодія: Скрипти використовують дані від системи вводу для керування об'єктами, змінювання їх положення, запуску анімацій та виконання інших дій у відповідь на команди користувача.

- Інтерфейс користувача (UI System);

Опис: Компоненти, що відповідають за відображення елементів інтерфейсу, таких як меню, індикатори здоров'я, повідомлення тощо.

Взаємодія: Інтерфейс взаємодіє зі скриптами для оновлення інформації, що відображається, та отримання введення від користувача через інтерфейс.

- Менеджер гри (Game Manager).

Опис: Відповідає за завантаження та розвантаження сцен, перехід між ними.

Взаємодія: Менеджер сцени дозволяє змінювати рівні, завантажувати нові об'єкти та зберігати прогрес гравця.

Взаємодія компонентів

Усі ці компоненти працюють разом для створення інтерактивного ігрового середовища. Ось приклад взаємодії компонентів у процесі гри:

- Гравець натискає клавішу далі Система вводу передає команду в скрипт керування персонажем. Приклад системи вводу для горизонтальної осі клавішами a – d представлено на рис. 3.1.

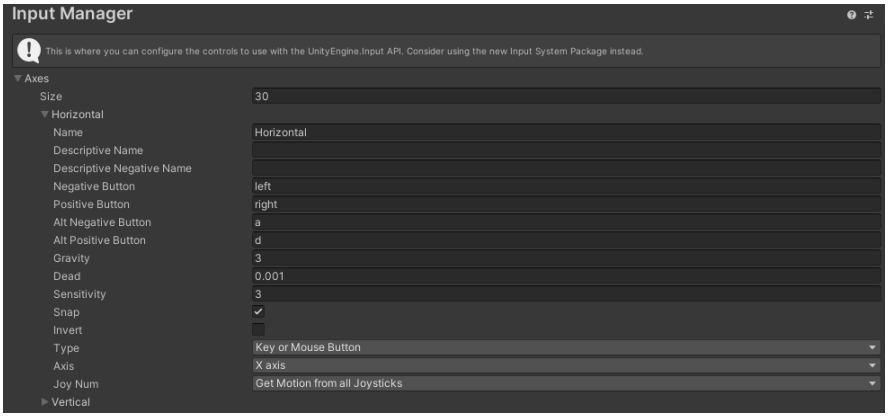


Рис. 3.1. Input Manager

- Скрипт керування персонажем обробляє введення та змінює параметри компонента Rigidbody для переміщення персонажа, який представлений на рис. 3.2.

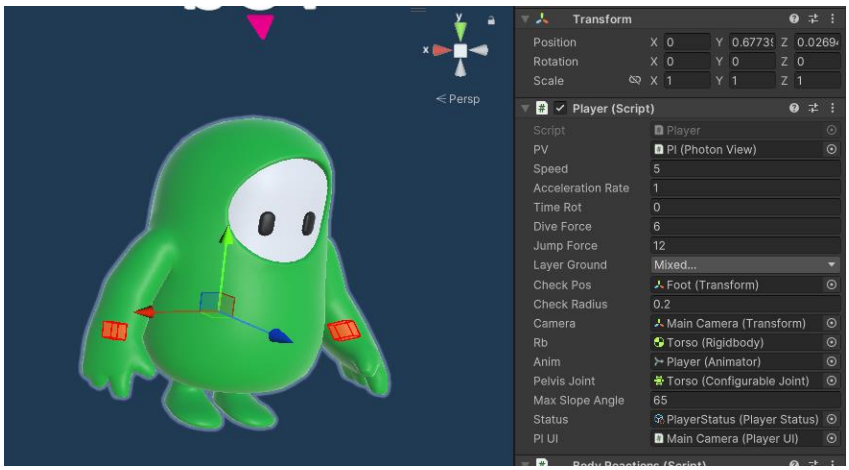


Рис. 3.2. Компонент керування персонажем

- Фізичний рушій обробляє рух персонажа та зіткнення з іншими об'єктами, використовуючи Collider та Rigidbody. На рис. 3.3 можна побачити принцип обробки зіткнення фізичним двигуном.

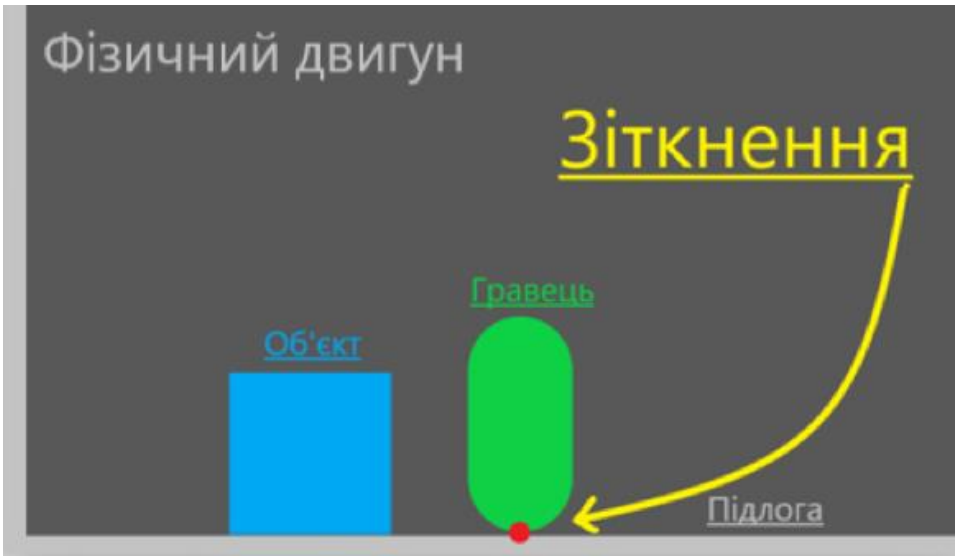


Рис. 3.3. Приклад визначення зіткнень фізичним двигуном

- Система анімації запускає відповідні анімації, наприклад, біг або стрибок, в залежності від дій персонажа.

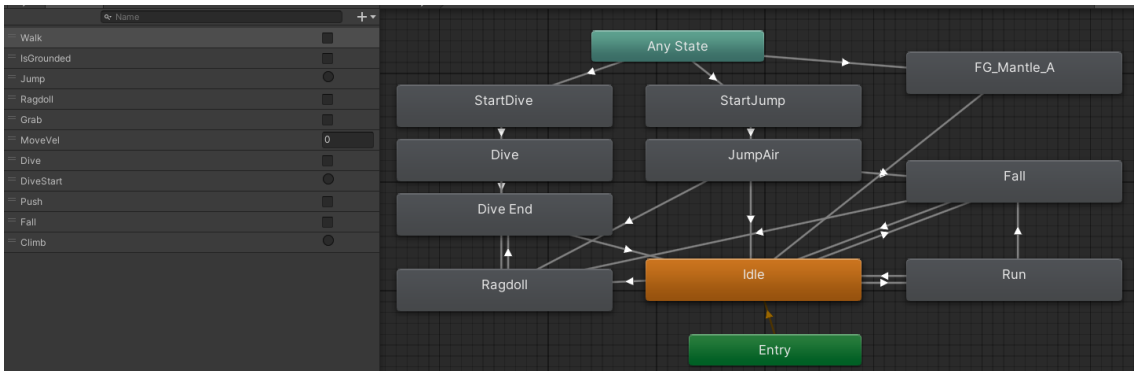


Рис. 3.4. Вікно компонента Animator з деревом анімацій для героя

На рис. 3.4. забрежено блоки та стрілки між ними. Кожен блок є одним з станів героя, а стрілки показують на те з якого стану в який герой може може переходити.

- Інтерфейс користувача оновлює інформацію про стан гри або інші показники на основі взаємодії персонажа з об'єктами.

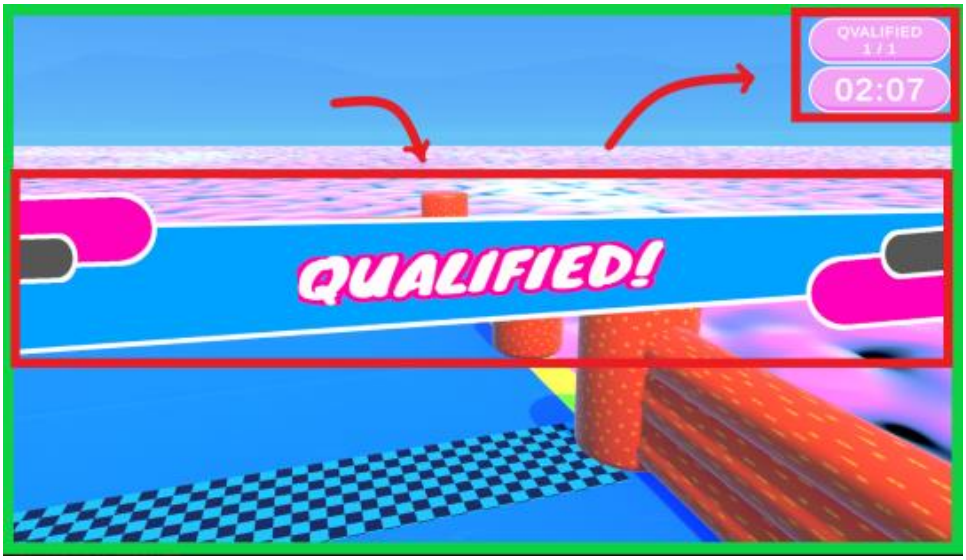


Рис. 3.5. Інтерфейс користувача підчас ігрової сесії

На рис. 3.5. показано інтерфейс користувача в процесі гри. В верхньому правому куті відображається поточна ціль і час на її виконання. По центру екрану можна спостерігати повідомлення які з’являються при виконанні умов рівня чи закінченню таймера.

- Менеджер сцени може змінити сцену, якщо персонаж досягає певного пункту, завантажуючи нові об'єкти та налаштування, набір сцен зображено на рис.3.6.

Рис. 3.6. Вікно з готовими сценами(рівнями) в ігровому додатку

Такий підхід забезпечує модульність та масштабованість системи, дозволяючи легко додавати нові функціональні можливості та компоненти.

3.2. Реалізація моделювання взаємодії об'єктів

Перший етап розробки передбачає створення простої системи, яка б дозволила об'єктам в грі взаємодіяти з навколишнім середовищем та один з одним, дотримуючись фізичних законів.

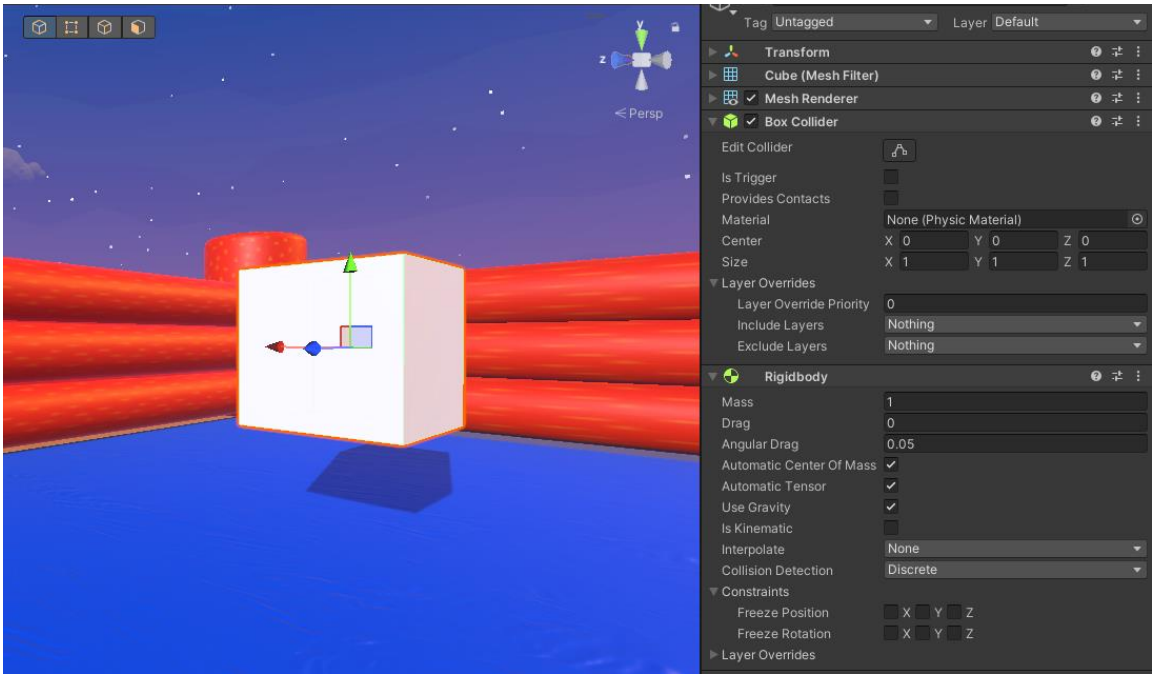


Рис. 3.7. Вікно інспектора для налаштування властивостей компонентів ігрових об’єктів

Додавши до куба компоненти Box Collider та Rigidbody на нього вже буде впливати гравітація та колізії з іншими об'єктами, це можна спостерігати на рис. 3.7. Для створення стаичних фізичних об'єктів таких як підлога чи стіни, використаємо лише компонент Collider. Це нам створить об'єкти які будуть взаємодіяти з іншими фізичини об'єктами але не реагувати на сили які будуть до них направленні [1]. На рис. 3.8 можна побачити налаштування компонента Mesh Collider для статичної стіни на ігровому рівні.

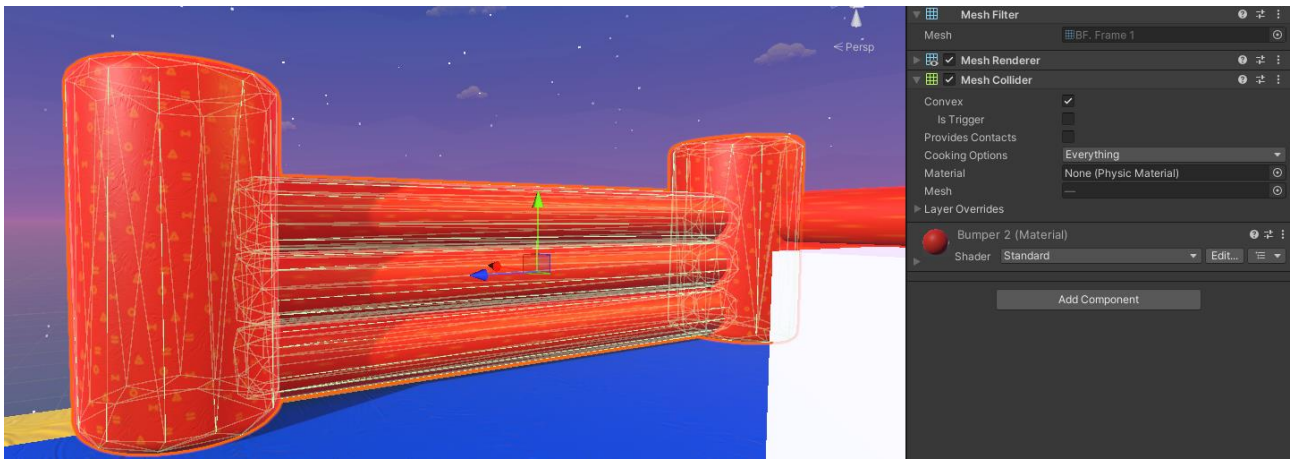


Рис. 3.8. Вікно інспектора для налаштування властивостей компонентів ігрових об’єктів

Жанром ігрового застосунку буде перегони та королівська битва, де гравцям потрібно буде дістатися до фінішу оминаючи всі перешкоди які виникатимуть на їх шляху.

Другий етап розробки передбачає реалізацію першкод та головного героя. Для перешкод було вибрано пастки з шоу Wipeout, в якому команда людей змгається за кубок. В прототипі реалізовано декілька видів перешкод, а саме:

- Колода що обертається;
- Платформа для обертання;
- Батут;
- Молот що обертається;
- Вентилятор;
- Айро труба;
- Бампер;
- Гармата.

Колода що обертається. Вона складає з себе колоду та вісь навколо якої вона обертається. Обертання виконується з певною швидкістю за годинниковою або проти годинникової стрілки. При зіткненню з героєм буде прораховано напрям в якому відкине героя із силою якою він зіткнеться з колодою (рис. 3.9) це добре демонструє.

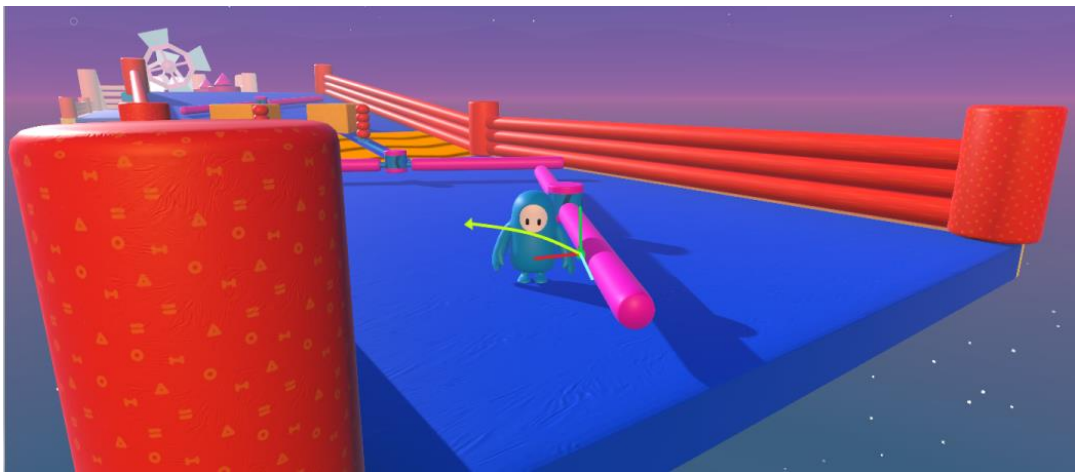


Рис. 3.9. Перешкода Колода що обертається

Молот що обертається. Це великий обертовий молот, який стоїть вертикально і замахується на гравців на високій швидкості, що демонструється на рис. 3.10.

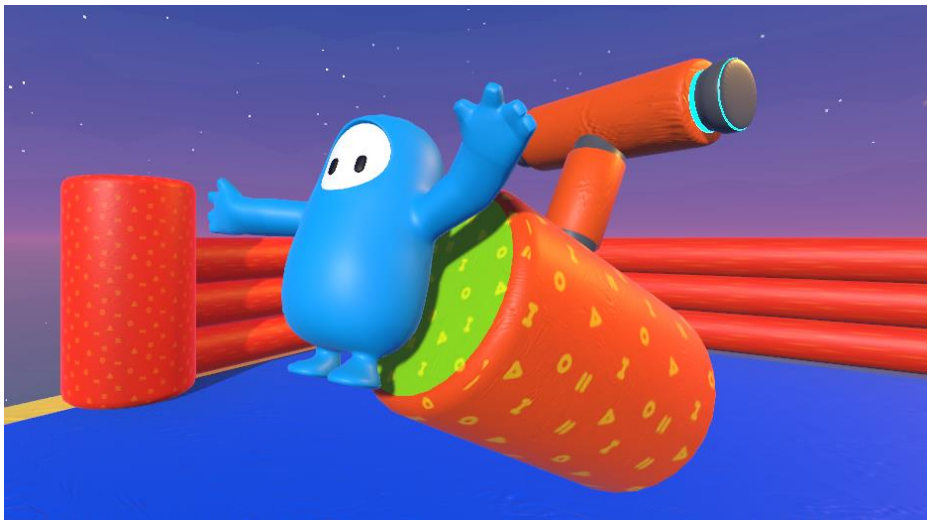


Рис. 3.10. Взаємодія героя з перешкодою Молот

Платформа що обертається. Це кругла платформа, яка обертається безперервно і швидко, змушуючи гравців викидатися через свою швидкість у напрямку стрілок.

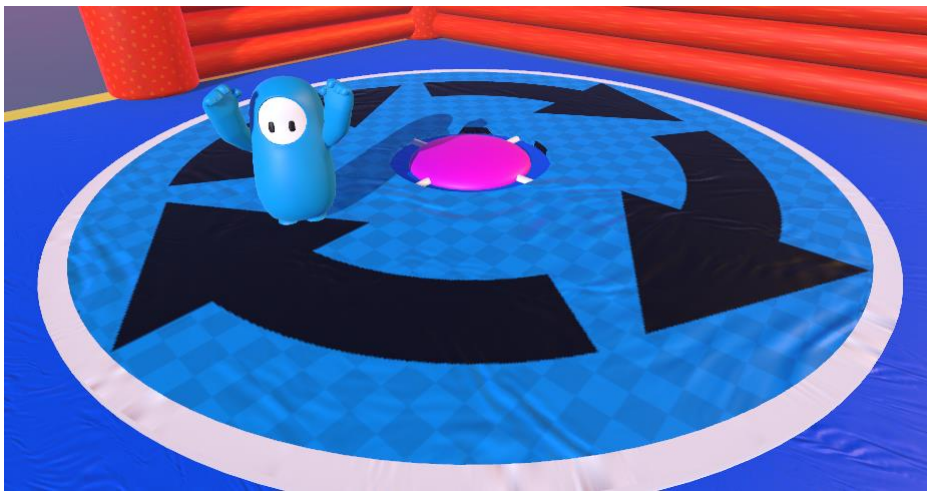


Рис. 3.11. Взаємодія гравця з платформою

Бампер. Це об'єкт з накладкою, схожою на повітряну кулю. Гравці, які торкаються його, відкидаються назад з певною силою, зазвичай достатньою, щоб скинути свій імпульс. На них також можна стрибати, при цьому він діє як батут штовхаючи гравців в одному напрямку з силою, спрямованою вгору, що дорівнює стрибку.

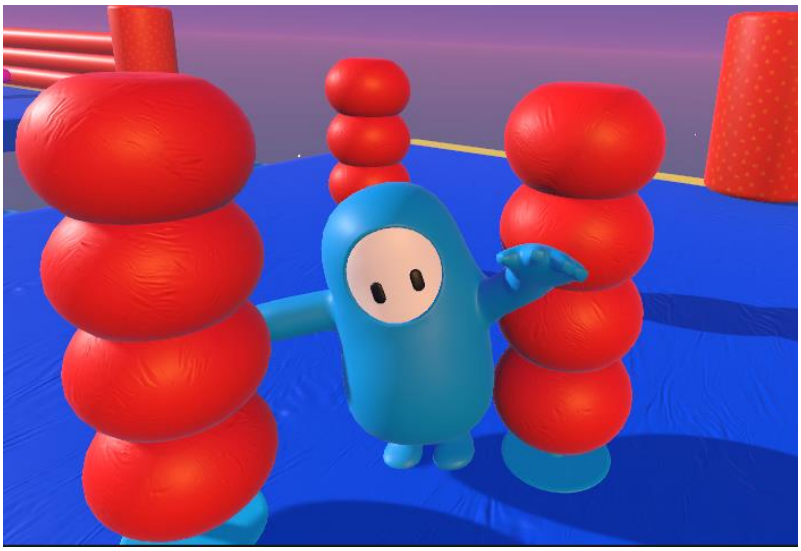


Рис. 3.12. Взаємодія гравця з перешкодою бампер

Батут. Цей Об’єкт який відкидає гравців які на нього стрибають з певною силою, може використовуватися для подолання високих перешкод чи прірв (рис. 3.13) це гарно демонструє.

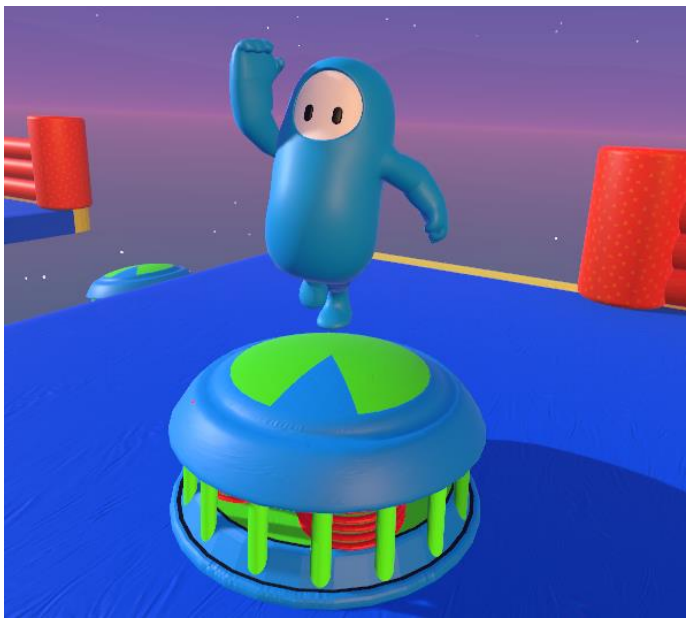


Рис. 3.13. Взаємодія гравця з Батутом

Вентилятори. Це вертикальні та горизонтальні вентилятори, які постійно дують в напрямку перед собою штовхаючи гравців, захоплених його гвинтом, може використовуватися для подолання високих перешкод чи прірв. На рис. 3.14. можна побачити принцип взаємодії.

Рис. 3.14. Перешкода горизонтальний вентилятор

Айро труба. Це велитенські трибу з повітрям зображені на рис. 3.15, які використовуються для швидкого переміщення між платформами.

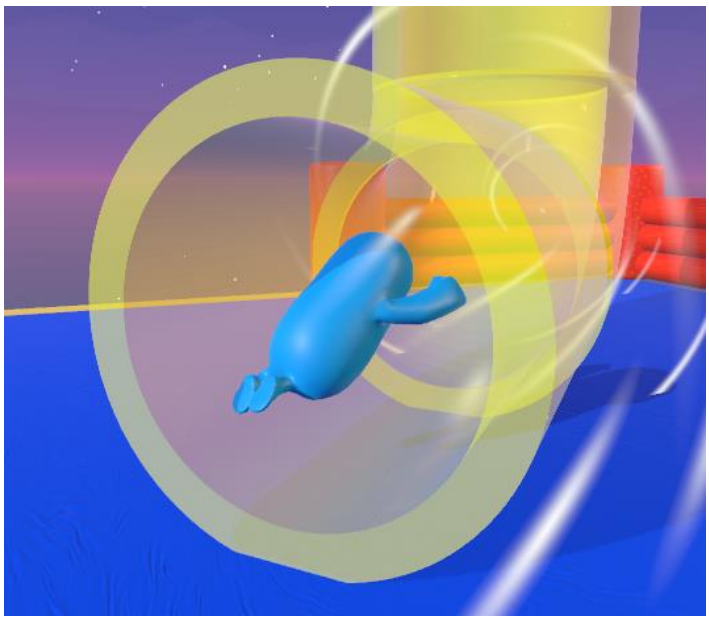


Рис. 3.15. Айро труба

Гармата. Це статична перешкода, яка постійно з випадковою періодичністю стріляє кулями по гравцях. Кулі в свою чергу через свою швидкість і розмір складають небезпеку для гравців (рис.3.16) це гарно демонструє.

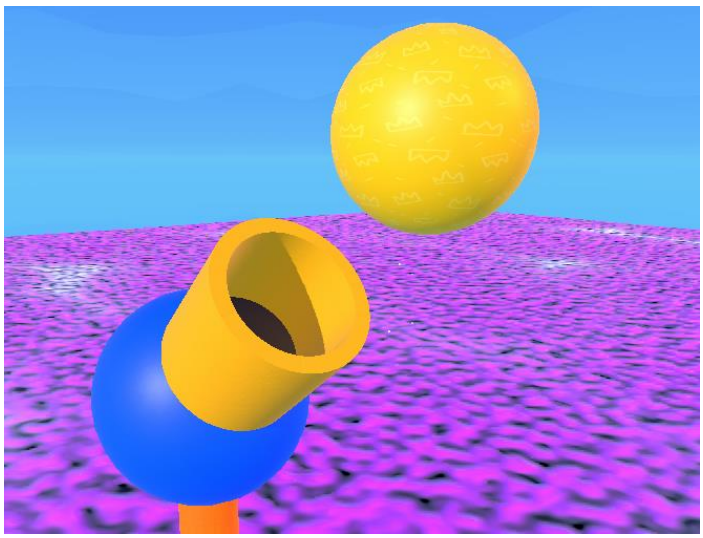


Рис. 3.16. Перешкода Гармата з ядром, що вона вистрілює

На третьому етапі для реалізації головного героя було використано не лише RigidBody та Capsule Collider для фізичної симуляції а й фізичне з'єднання частин тіла Configurable Joint, для реалізації реалістичних ефектів.

Рис. 3.17. Використання компоненту Configurable Joint на практиці

На рис. 3.17. можна переконатися що використання компоненту Configurable Joint робе взаємодію гравця з об’єктами реалістичнішою, голова персонажа тепер реагує на зіткнення і не проходить наскрізь об’єкти.

Для того щоб руки героя не проходили об’єкти навскрізь було використано методи фізики двигуна Unity такий як Physics.OverlapSphere який відмальовує невидиму сферу навколо об’єкта руки. Було реалізовано компонент для курування позиції руки якщо вона зіштовхується з об’єктами. А саме підчас кожного дотику визначається протилежний напрямок дотику і рука переміщується в цьому напрямку на радіус сфери яка перевіряє зіткнення, що продемонстровано на рис. 3.18. [5].

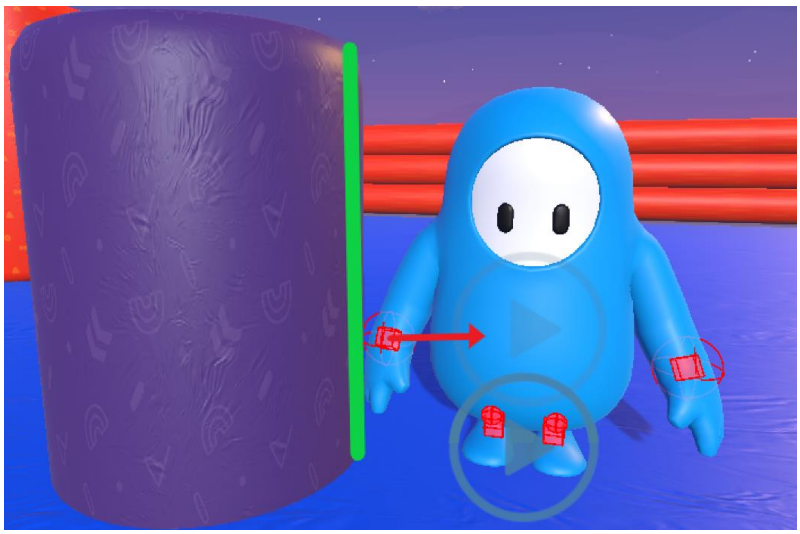


Рис. 3.18. Принцип дії компонента що запобігає проникненню рук героя в об’єкти

Четвертим етапом є анімації гроя. Всі рухи та реакції героя супроводжуються відповідними анімаціями. Які в свою чергу керуються через скрипти керування героєм.

Реалізація анімацій героя була зроблена не стандартним способом. Зазвичай в 3D іграх застосовують скелетну анімацію. Скелетна анімація це один з способів створити анімацію для моделей за допомогою набору кісток що симулює справжні кістки, що можна побачити на рис. 3.19.

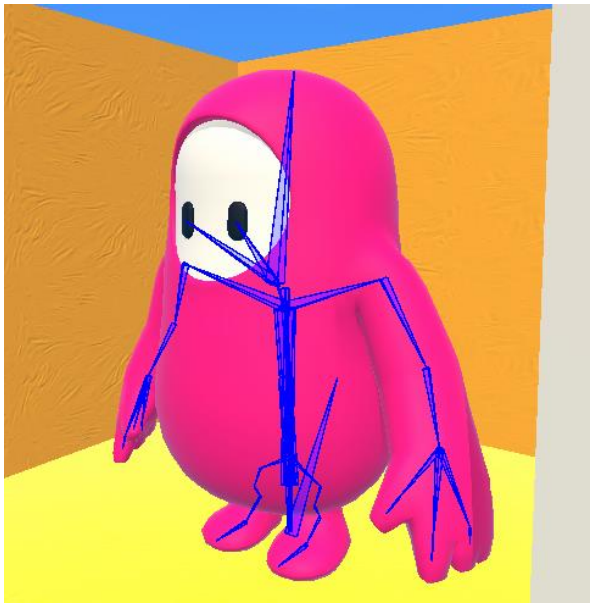


Рис. 3.19. Графічне відображення скелету 3D моделі персонажа

Ще один спосіб створення анімацій є Ragdoll анімації. Це спосіб використовується для реалізації фізичних анімацій для реалістичної поведінки не живих персонажів таких як трапова лялька. Також їх застосовують в шутерах при смерті гравців, щоб симулювати не живе тіло.

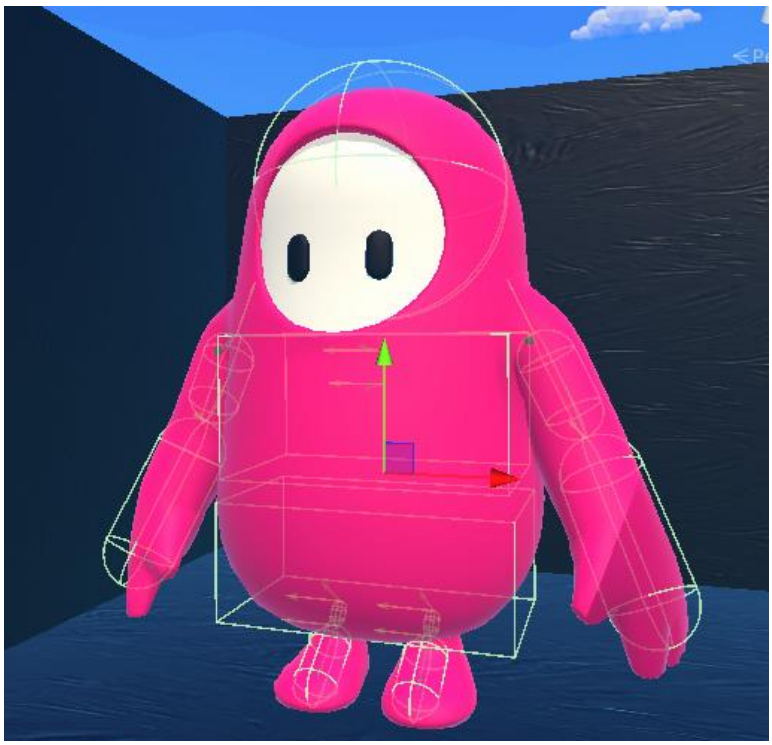


Рис. 3.20. Створений Ragdoll персонаж з компонентами Collider

Мій підхід до створення анімацій в тому щоб об’єднати ці два принципи для реалізації реалістичної поведінки персонажа. Цей підхід називається Active Ragdoll. Він є інноваційним способом створення анімацій для персонажів в іграх. Це поєднання скелетної анімації та Ragdoll для досягнення більш реалістичної та інтерактивної поведінки персонажів [6].

Переваги Active Ragdoll підходу:

- **Реалістичність:** Active Ragdoll дозволяє персонажам реагувати на фізичні взаємодії в реальному часі. Наприклад, якщо персонаж піддається впливу сили (штовхання, падіння), його рухи будуть виглядати природно завдяки фізичним симуляціям.
- **Інтерактивність:** Персонажі можуть взаємодіяти з навколишнім середовищем більш природно. Наприклад, вони можуть падати і скочуватись по схилу, зберігаючи контроль над своїм тілом.
- **Гнучкість:** Поєднання традиційної скелетної анімації з Ragdoll дозволяє досягти балансу між передбачуваними анімаціями та динамічними реакціями на події в грі.

Рис. 3.21. Практичне порівняння видів анімування в 3D

Як видно на вище наданому рис. 3.20. реалізація за допомогою скелетної анімації ніяк не взаємодіє з кулями. Ragdoll взаємодіє з об’єктами але в цьому стані ми не можемо програвати анімацій. А на моїй реалізації активного Ragdoll голова персонажа реагує на зіткнення з кулею.

Вбудований компонент Unity для анімацій Animator дозволяє курувати анімаціями за допомогою зміни параметрів, а також налаштовувати дерево анімацій [1]. Дерево анімацій героя представлено на рис. 3.21.

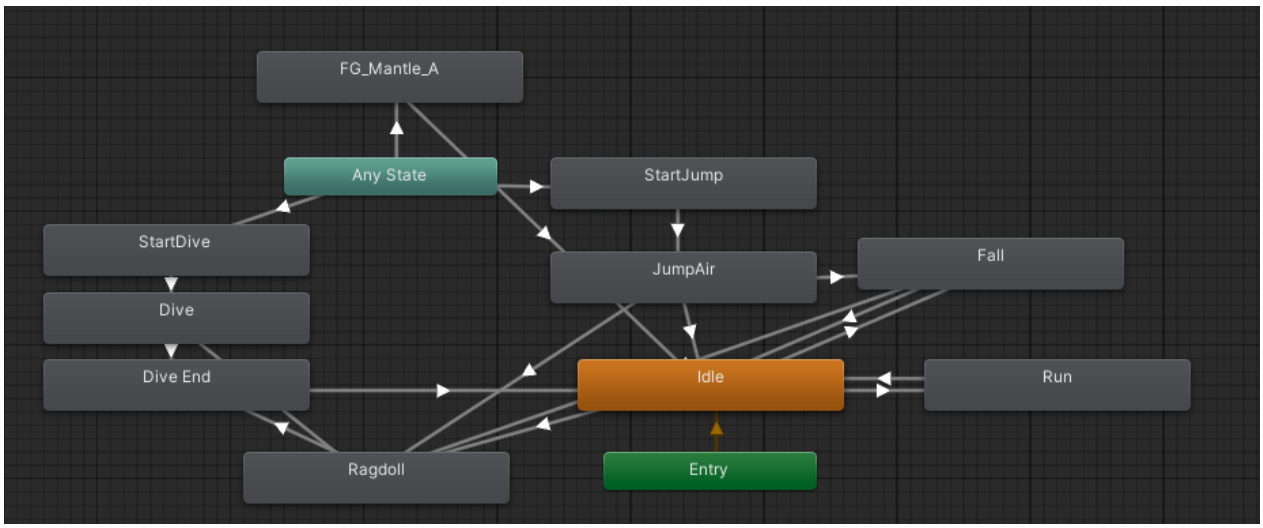
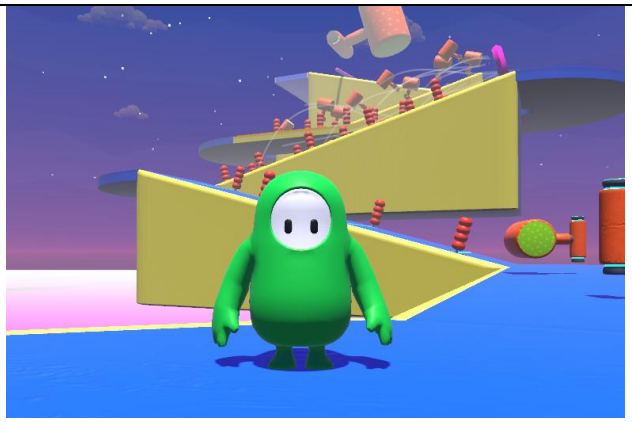
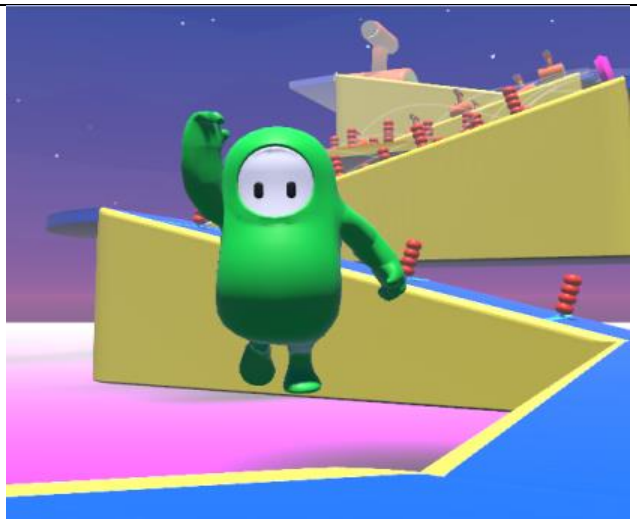


Рис. 3.22. Вікно компоненту Animator з блоками анімацій та переходами між ними

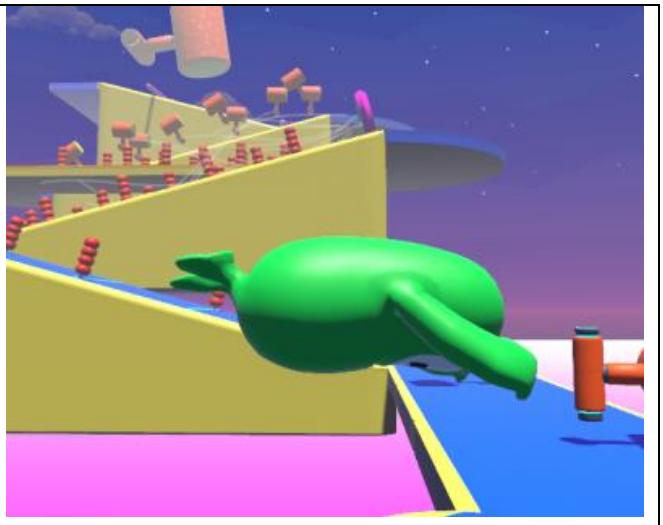
Де блоки це анімація а стрілки це доріжки переходів між ними з певними умовами.

Сам герой має кілька станів, представлених в таб. 3.1.

Таблиця 3.1

Стан спокою – герой перебуває в ньому коли він не рухається і він не зіштовхується з перешкодами.	
Стан стрибку - герой перебуває в ньому коли він стрибнув вгору і він не зіштовхується з перешкодами.	
Стан бігу - герой перебуває в ньому коли він рухається і він не зіштовхується з перешкодами.	

Продовження таблиці 3.1

Стан ниряння - герой перебуває в ньому коли натиснута клавіша ниряння, після чого герой стрибає вперед, під час цього стану він не може рухатися.	
Стан втрати рівноваги – настає кожного разу при контакті з перешкодою якщо імпульс дотику менший за імпульс який вводить героя в нокаут. Керувати героєм підчас цього стану не можна.	
Стан нокауту – настає кожного разу при контакті з перешкодою якщо імпульс дотику більший чи рівний за імпульс який вводить героя в нокаут. Керувати героєм підчас цього стану не можна. Герой лежить непритомно n кількість часу пропорційному силі удару перешкоди, але не більше 2с.	

Розширити стандартну взаємодію об’єктів можна за допомогою скриптів на мові C#, додаючи різних властивостей чи ефектів під час взаємодії [1]. Використовуючи такі методи Collider як:

Таблиця 3.2

Таблиця 3.2

OnCollisionEnter	Визивається коли об’єкт з collider/rigidbody доторкнувся до іншого об’єкта з collider/rigidbody.
OnCollisionExit	Визивається коли об’єкт з collider/rigidbody перестав доторкатися до іншого об’єкта з collider/rigidbody.
OnCollisionStay	Визивається коли об’єкт з collider/rigidbody доторкається протягом певного часу до іншого об’єкта з collider/rigidbody.
OnTriggerEnter	Визивається коли інший Collider other входить в зону тригера.
OnTriggerExit	Визивається коли інший Collider other виходить із зону тригера.
OnTriggerStay	Визивається коли інший Collider other перебуває в зоні тригера.

Або використовуючи такі методи Rigidbody як:

Таблиця 3.3

AddExplosionForce	Прикладає до твердого тіла силу, що імітує ефекти вибуху.
AddForce	Додає силу Rigidbody.
AddForceAtPosition	Прикладає силу в потрібному положенні. В результаті на об'єкт буде прикладено обертання і силу.
AddRelativeForce	Додає силу Rigidbody щодо його системи координат.
AddTorque	Додає обертання Rigidbody.

3.3. Реалізація багатокористувацької складової

Для реалізації багатокористувацької взаємодії в додатках на Unity є безліч інструментів таких як UNET, Photon PUN 2, Mirror, Bolt.

Таблиця 3.4

Характеристики	**PUN 2**	**UNET**	**Mirror**	**Bolt**
Легкість використання	Висока	Середня	Середня	Висока
Підтримка	Висока	Низька	Висока	Середня
Інтеграція з Unity	Активна	Припинено	Активна	Активна
Ціна	Базова версія	Безкоштовно	Безкоштовно	Безкоштовно
Стабільність	Висока	Середня	Висока	Висока
Підтримка різних платформ	Так	Так	Так	Так
Можливість налаштувань	Проста	Проста	Складна	Проста
Візуальне програмування	Ні	Ні	Ні	Так

Після детального аналізу було обрано інструмент Photon PUN 2.

Photon PUN 2 це пакет Unity для реалізації багатокористувацьких ігор. В ньому присутній гнучкі налаштування пошуку матчів, після чого гравців переносить до кімнат, де всі об’єкти можна синхронізувати через мережу. Швидкий і стабільний зв’язок забезпечується спеціальними виділеними серверами, через що клієнтам не потрібно підключатися один до одного. Photon PUN 2 базується на моделі клієнт-сервер, представлену на рис. 3.23. [7]. Ця модель включає такі компоненти як:

- Сервер;
- Клієнт;
- Зв’язок між клієнтами і сервером;

- Розподілення відповідальності;
- Обробка подій.

Сервер Photon має найбільший пріоритет над грою, що забезпечує йому мати контроль над станом гри та уникати виникнення спроб чітерства чи зміни стану гри. Також він обробляє всі події гри, такі як зміна стану об’єкта, дії користувачів чи інші події. Після чого він розсилає всі клієнтам зміни для забезпечення синхронізації ігрового процесу.

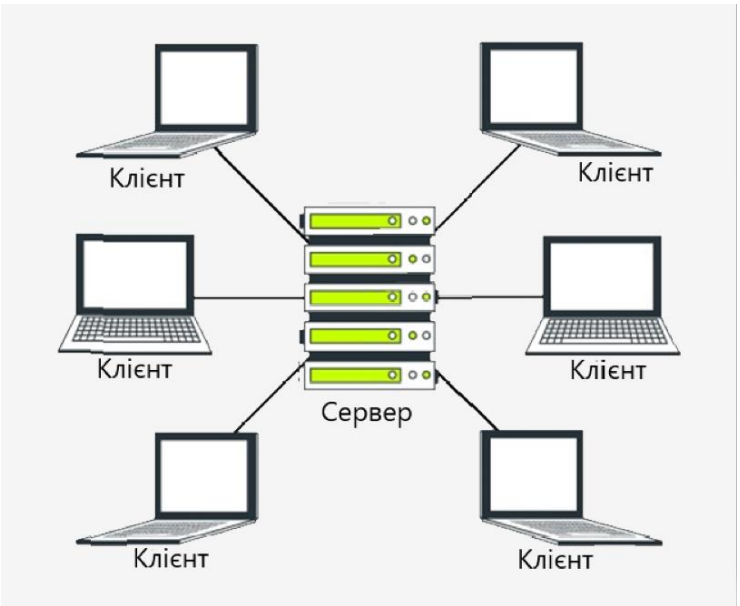


Рис. 3.23. Принцип моделі клієнт-сервер

Після встановлення пакету Unity Photon PUN 2 потрібно описати логіку пошуку матчів та підключення до кімнат. Було використано просту логіку пошуку матчів, а саме пошук всіх відкритих кімнат та приєднання до однієї з них.

Для керування ігровою логікою потрібно створити відповідний скрипт такий як GameManager, приклад коду наведено в додатку А. Він відповідає за контролем поточного раунду та збереженням даних користувачів. Для логіки появи гравця було створено PlayerManager який відповідає лише за створення локального гравця чи його знищення. Схема логіки створення гравців показано на рис. 3.24 [8].

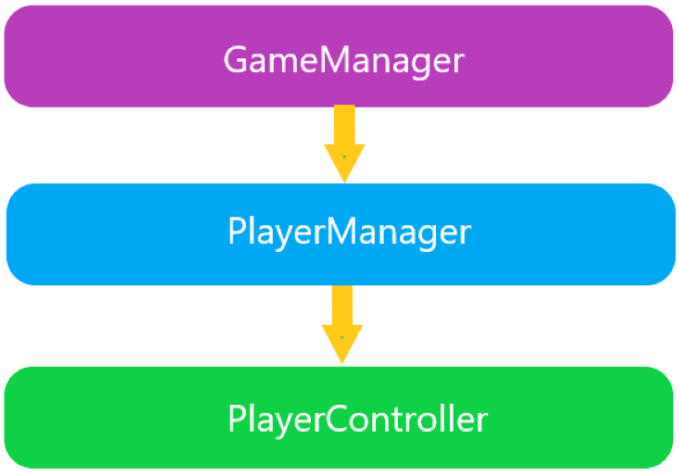


Рис. 3.24. Схема логіки створення гравців

Для синхронізації гравців між використовуються такі компоненти як:

- PhotonView;
- PhotonTransformView;
- PhotonAnimatorView.

PhotonView це ключовий компонент PUN 2 для забезпечення синхронізації об’єктів між різними клієнтами в грі. Він дозволяє синхронізувати стан об’єкта в реальному часі між усіма гравцями, що підключені до однієї кімнати. Кожен PhotonView маєсвій власний ID, що дозволяє легко керувати об’єктами які нам потрібні. Також він відстежує власника об’єкта, що забезпечує зміну стану об’єкта лише його власником а не іншими гравцями [7].

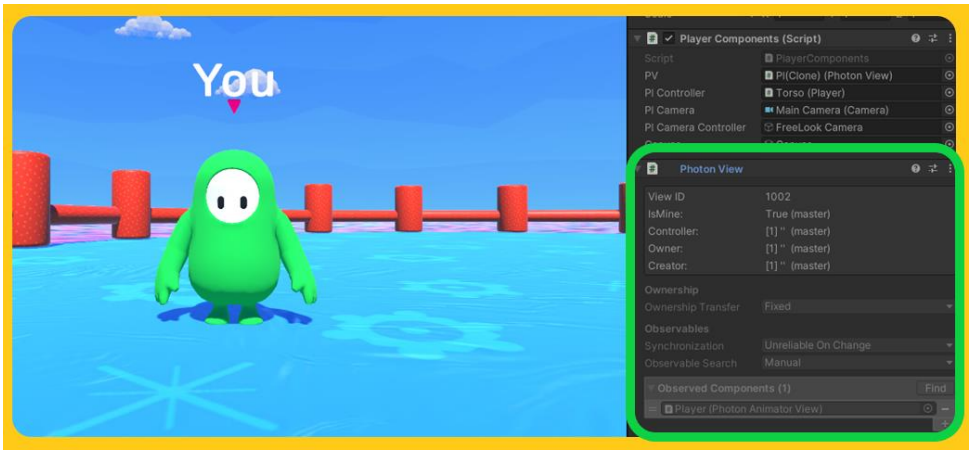


Рис. 3.25. Модель гравця з компонентом PhotonView

PhotonTransformView це потужний інструмент для синхронізації трансормацій об’єктів у онлайн грі, показано на рис. 3.26. Завдяки цьому компоненту можна автоматично синхронізувати позиції, обертання та розміри об’єктів, що і робить його важливим інструментом при розробці мультиплеєрних ігор.

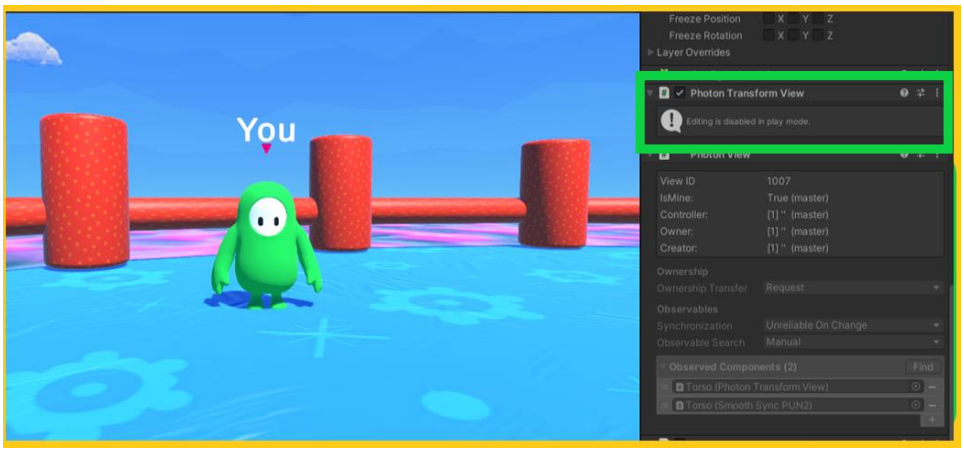


Рис. 3.26. Модель гравця з компонентом PhotonTransformView для синхронізації позиції

PhotonAnimatorView ще один з компонентів Photon PUN 2, який забезпечує синхронізацію анімацій між гравцями в мережівій грі. Він допомагає синхронізувати стан компонента Animator на об’єкті, забезпечуючи узгодженість анімацій і параметрів між усіма гравцями [7].

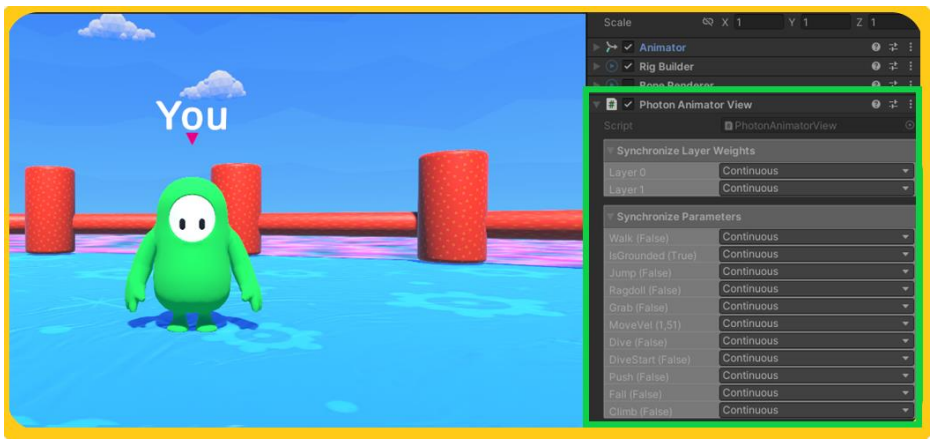


Рис. 3.26. Модель гравця з компонентом PhotonAnimatorView для синхронізації станів анімацій

3.4. Тестування та налагодження

Після програмної реалізації взаємодії об’єктів в середовищі Unity та реалізації багатокористувацького режиму потрібно протестувати та налагодити системи, щоб зменшити ризик виникнення помилок, чи не передбачуваної поведінки ігрового процесу.

Для початку потрібно налаштувати компонент гравців який відповідає за їх реакцію при зіткненнях і взаємодії. Після довгого тестування було обрано оптимальні показники для характеристик взаємодії, рис. 3.28, а саме мінімальні сили з якими гравця будуть відкидати перешкоди при зіткненні, значення сили з якою потрібно вдарити гравця щоб той втратив свідомість, значення сили з якою гравець стоїть в стані спокою та без свідомості .

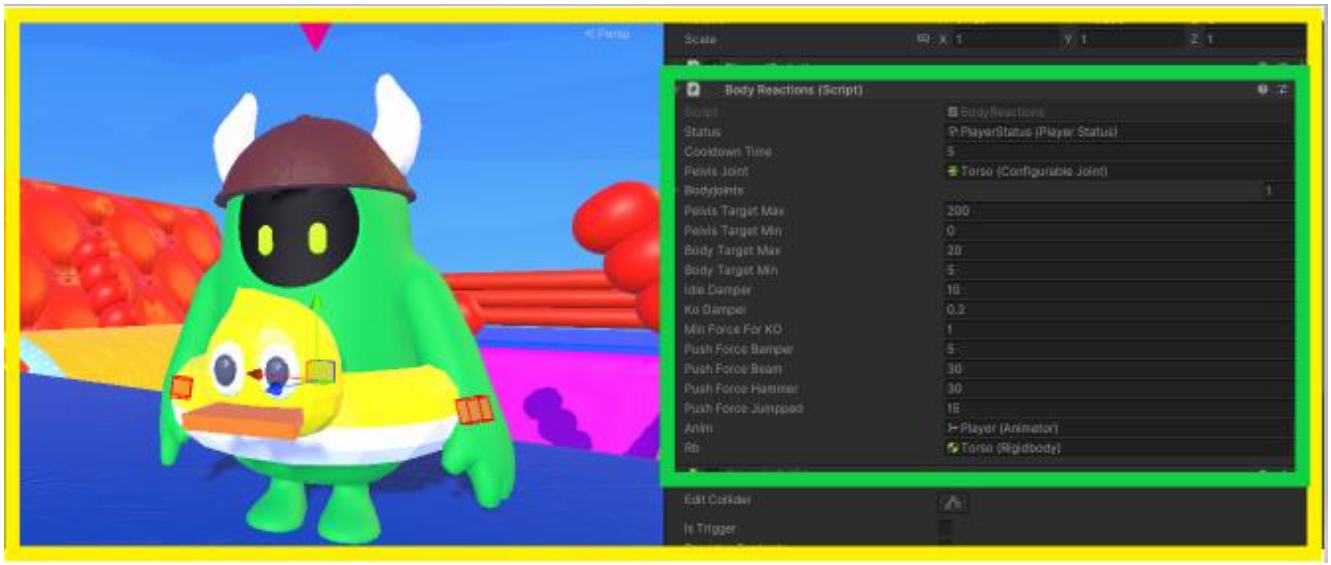


Рис. 3.28. Модель гравця з налаштованим компонентом реакції тіла

Наступним етапом тестування буде підключення клієнта до мережі та налаштування зовнішнього вигляду героя. Процес підключення до мережі супроводжується екраном завантаження.

Після успішного підключення гравці потрапляють в головне меню, де вони можуть розпочати гру чи налаштувати зовнішній вид свого героя. Зміна зовнішнього вигляду гравця повинна зберігатися (рис. 3.29).

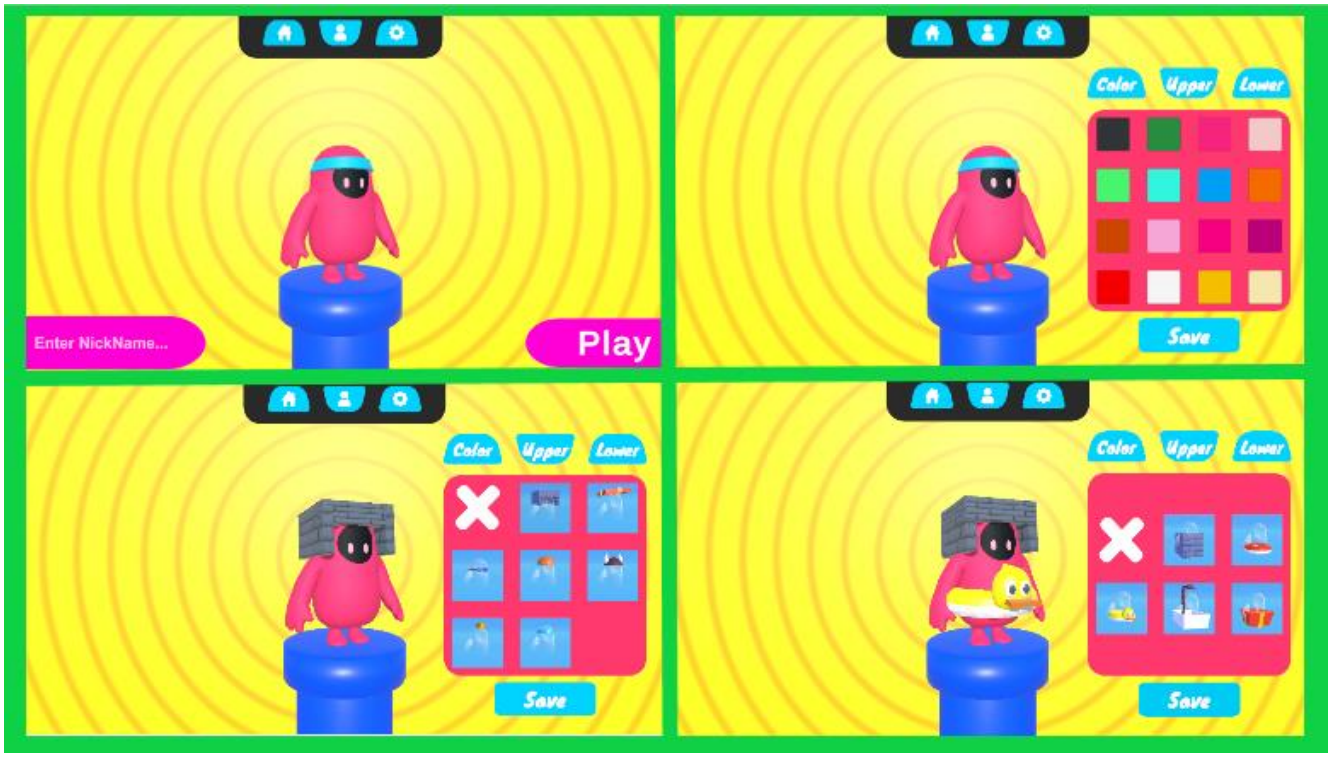


Рис. 3.29. Процес зміни зовнішнього вигляду головного персонажа

Наступним етапом тестування буде синхронізація зовнішнього вигляду гравців та правильне функціонування логіки матчів. Компонент Game Manager, рис. 3.30, відповідає за логіку матчів, які складаються з трьох раундів. Він також відповідає за збереження поточних даних гравців, приклад коду наведено в додатку А.

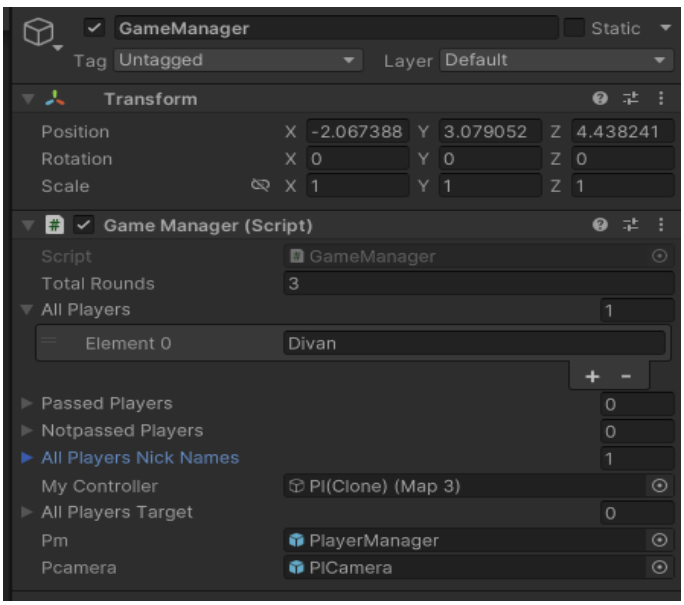


Рис. 3.30. Вигляд компоненту Game Manager в вікні інспектора

Після налаштування та тестування зовнішній вигляд гравців синхронізовано і правильно відображається для інших клієнтів гри. На рис. 3.31. це гарно продемонстровано.

Рис. 3.31. Процес гри з синхронізацією зовнішнього вигляду інших клієнтів

Коли таймер закінчується гравці отримують відповідні сповіщення, рис. 3.32, про їх прогрес на рівні, а саме пройшов чи ні.

Рис. 3.32. Сповіщення гравця про їх прогрес

Після чого Game Manager змінює сцену на рівень, де гравців що не виконали умову для проходження викидають з платформ, а ті хто пройшов залишаються грати далі (рис 3.33).

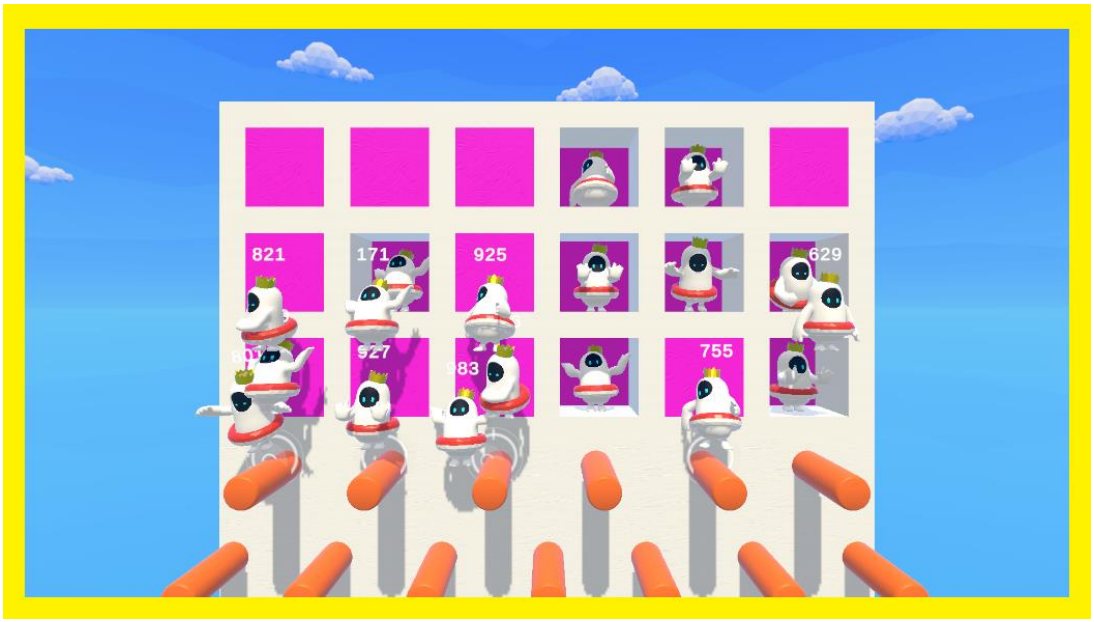


Рис. 3.33. Процес відбору гравців

Коли всі тири раунди завершаються то гравці можуть побачити найкращого гравця в цій грі, якщо такого немає то всі бачить свого персонажа, оскільки гра завершена можна перейти в головне меню для пошуку нової гри.

Рис. 3.34. Показ найкращого гравця

Також протестовано синхронізацію позиції та анімацій між клієнтами. Для цього проведено багатокористувацьку гру щоб можна було спостерігати також і за іншими гравцями. Після тривалого тестування всі переміщення та анімації гравців синхронізовано та налаштовано завдяки компонентам синхронізації Photon PUN 2.

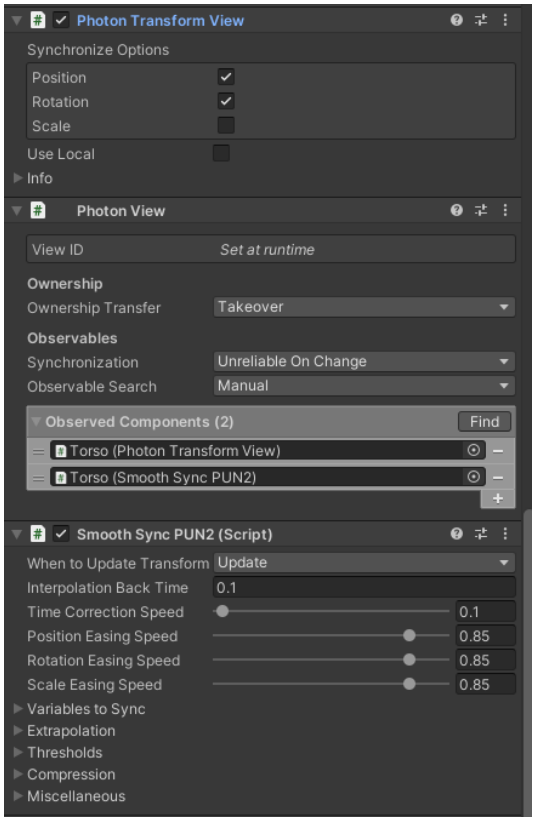


Рис. 3.35. Компоненти синхронізації Photon PUN 2

3.4. Висновки до розділу

У цьому розділі було розглянуто та реалізовано основні аспекти розробки ігрового застосунку, зокрема архітектура системи, моделювання взаємодії об'єктів, багатокористувацький режим, тестування і налагодження. Детальний аналіз і виконання кожного з цих аспектів дозволили створити функціональну технологію моделювання взаємодії об’єктів в ігровій системі, яка відповідає сучасним стандартам розробки.

Архітектура системи була спроектована таким чином, щоб забезпечити максимальну гнучкість та масштабованість. Основні компоненти системи, такі як

управління гравцями, ігрові об'єкти, взаємодія між клієнтами та сервером, були структуровані у модульний спосіб. Це дозволило легко додавати нові функціональні можливості та модифікувати існуючі без суттєвих змін у коді. Крім того, використання дизайну на основі компонентів значно полегшило процес тестування та налагодження.

Моделювання взаємодії об'єктів було реалізовано з використанням фізичного рушія Unity. Це забезпечило реалістичність та достовірність поведінки об'єктів у грі. Було створено різні типи взаємодій, такі як зіткнення, рух гравітація, що надало грі динамічності та інтерактивності.

Багатокористувацька складова була реалізована за допомогою Photon Pun 2, що дозволило створити надійне та масштабоване мережеве середовище. Були реалізовані основні функції, такі як з'єднання гравців, синхронізація станів об'єктів та обмін даними між клієнтами та сервером. Особлива увага приділялася мінімізації затримок та забезпеченню стабільної роботи ігрового додатку навіть при великій кількості одночасних підключень.

Процес тестування та налагодження включав у себе як автоматичне, так і ручне тестування різних аспектів ігрового додатку. Автоматичне тестування допомогло виявити та виправити основні помилки та забезпечило стабільність основних функцій. Ручне тестування дозволило перевірити зручність використання ігрового додатку з точки зору кінцевого користувача. Було проведено кілька етапів налагодження для забезпечення відповідності ігрового додатку очікуванням.

ВИСНОВКИ

В рамках кваліфікаційної роботи було досліджено та реалізовано всі необхідні етапи для створення повнофункціональної гри з багатокористувацькою підтримкою. Робота охопила аналіз предметної області, огляд існуючих ігрових продуктів, визначення основних завдань та вимог, а також технічну реалізацію проекту.

Було розглянуто поточні тенденції та потреби у сфері ігрової індустрії, визначено, що створення багатокористувацьких ігорових додатків є актуальним завданням через зростаючу популярність онлайн-ігор.

Проведений аналіз існуючих ігрових продуктів дозволив визначити сильні та слабкі сторони конкурентів. Це допомогло сформувати вимоги до майбутнього ігрового додатку та визначити унікальні аспекти, які могли б виділити її на ринку.

На основі аналізу було сформульовано основні завдання та вимоги до ігрового додатку. Визначено ключові функціональні можливості, такі як підтримка багатокористувацького режиму, реалістичне моделювання фізичних взаємодій та зручний інтерфейс користувача.

Було проведено детальний аналіз мов програмування, зокрема C# та C++, які є основними для розробки ігор у Unity та Unreal Engine відповідно. Було визначено переваги та недоліки кожної з мов.

Аналіз фреймворків дозволив визначити, що Unity є більш зручним для створення інді-ігор та проектів середнього масштабу, тоді як Unreal Engine підходить для більш масштабних та ресурсомістких проектів. Було визначено, що для розробки гри найбільш доцільно використовувати Unity Engine, Blender 3D для створення 3D-моделей та Visual Studio як основне середовище розробки.

Проведений аналіз дозволив обрати найбільш підходящі інструменти для розробки гри, що забезпечують оптимальне поєднання функціональності та зручності.

Архітектура системи була спроектована таким чином, щоб забезпечити гнучкість та масштабованість. Використання модульного підходу дозволило легко додавати нові функції та оптимізувати процес розробки.

Реалістичне моделювання взаємодії об'єктів було досягнуто завдяки використанню фізичного рушія Unity. Це дозволило створити достовірну та інтерактивну ігрову середу.

Використання Photon PUN 2 для реалізації мультиплеєрної складової забезпечило надійне та масштабоване мережеве середовище, що дозволяє гравцям взаємодіяти у режимі реального часу.

Процес тестування включав автоматичне та ручне тестування, що дозволило виявити та виправити помилки. Налагодження забезпечило стабільність роботи гри та відповідність вимогам.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity technologies | Unity documentation. URL: https://docs.unity3d.com/Manual/index.html (дата звернення 16.05.2024).
2. C# for c++ developers | microsoft . URL: https://learn.microsoft.com/en-us/previousversions/visualstudio/visualstudio2008/yyaad03b(v=vs.90)?redirectedfrom=msdn (дата звернення 17.05.2024).
3. Blender 3d training | blender studio . URL: https://studio.blender.org/training/ (дата звернення 17.05.2024).
4. Use visul studio tools for unity | microsoft learn . URL: https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/using-visual-studio-tools-for-unity?pivots=windows (дата звернення 17.05.2024).
5. Ik rigging and weapon system in unity | medium . URL: https://medium.com/@pkillman2000/ik-rigging-and-weapon-system-in-unity-3cb6be888de3 (дата звернення 17.05.2024).
6. How to make active ragdolls in unity | medium . URL: https://sergioabreu-g.medium.com/how-to-make-active-ragdolls-in-unity-35347dcb952d (дата звернення 17.05.2024).
7. Pun 2 introduction | photon engine . URL: https://doc.photonengine.com/pun/current/getting-started/pun-intro (дата звернення 17.05.2024).
8. Matchmaking guide | photon engine . URL: https://doc.photonengine.com/pun/current/getting-started/pun-intro (дата звернення 17.05.2024).

ДОДАТКИ

Додаток А

Код класу GameManager

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

public class GameManager : MonoBehaviourPunCallbacks
{
    public static GameManager instance;
    [SerializeField] private int totalRounds = 1;
    private int currentRound = 0;

    public List<string> allPlayers = new List<string>();
    public List<string> passedPlayers = new List<string>();
    public List<string> notpassedPlayers = new List<string>();
    public List<string> allPlayersNickNames = new List<string>();
    public GameObject myController;
    public List<GameObject> allPlayersTarget = new List<GameObject>();

    [SerializeField] private GameObject pm;
    [SerializeField] private GameObject pcamera;

    private int playersLoaded = 0;
    private PhotonView PV;

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
        PV=GetComponent<PhotonView>();
    }
```

```
        private void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }
    private void OnDisable()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.buildIndex != 2 && scene.buildIndex != 3 && scene.buildIndex != 4)
        {
            if (currentRound != 1)
            {
                allPlayers.Clear();
                PlayersSkinManager.instance.CrearSkins();
if (passedPlayers.Count > 0)
                {
                    allPlayers.AddRange(passedPlayers);
                    passedPlayers.Clear();
                }
                if (allPlayers.Contains(PhotonNetwork.LocalPlayer.NickName))
                {
                    PhotonNetwork.Instantiate(pm.name, Vector3.zero, Quaternion.identity);
                }
                else
                {
                    PhotonNetwork.Instantiate(pcamera.name, Vector3.zero,
Quaternion.identity);
                    UpdatePlayerTargets();
                }
            }
            else
            {
                PhotonNetwork.Instantiate(pm.name, Vector3.zero, Quaternion.identity);
            }
        }
    }
```