МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Комп'ютерна інженерія»

Група: 2БКС-29

КВАЛІФІКАЦІЙНА РОБОТА

здобувача освіти денної форми навчання

БКС.29.06.000.КРБ

ГОЛОВАЧЕНКА ЛЕОНІДА ДМИТРОВИЧА

м. Одеса
2025 р.

АНОТАЦІЯ

Метою даної роботи є розробка ігрового проєкту на платформі Unity з акцентом на побудову повноцінної системи керування персонажем та створення ефективного левелдизайну. Особливу увагу приділено не лише технічному втіленню базових механік руху, а й архітектурі, що дозволяє масштабувати логіку керування для різних типів пристроїв та платформ.

У процесі дослідження вивчено закономірності взаємодії між системами введення, руху та анімації в контексті двовимірних платформерів. Розглянуто типові підходи до реалізації інтерфейсів введення, аналізовано способи оптимізації обробки інпутів для досягнення точного й передбачуваного управління в ігровому середовищі.

Отримані кількісні результати стосуються продуктивності системи, часу реакції на дії гравця, плавності анімацій та стабільності роботи на різних типах пристроїв. Проведено тестування реалізованої системи в кількох контрольних середовищах, що дозволило виявити сильні сторони обраного підходу та потенційні напрями вдосконалення.

Створено прототип гри з інтегрованою системою абстрактного введення, динамічним левелдизайном та базовим набором ігрових механік, серед яких — переміщення, стрибок, ривок і взаємодія з навколишнім середовищем. Усі компоненти побудовано з дотриманням принципів модульності та повторного використання коду, що забезпечує гнучкість подальшого розвитку проєкту.

Розглянуто питання з охорони праці та техніки безпеки в контексті роботи з комп’ютерною технікою та програмним забезпеченням. Зокрема, проаналізовано вплив тривалої роботи за комп’ютером на здоров’я користувача, описано базові вимоги до ергономіки робочого місця, освітлення, вентиляції, а також заходи для профілактики професійних захворювань, пов’язаних з тривалим сидячим способом праці.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-19**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левелдизайну»**

Проектний матеріал складається з пояснювальної записки на 75 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах)

Виконавець _____ (Головаченко Л.Д.)

Керівник проекту _____ (Іванова Л.В.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділенням _____ (Краснокутська К.Г.)

Захист «15» 06 2025 р. Протокол ЕК № 1

Оцінка ЕК 5 (відмінно) / 96

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"28" 05 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Головаченка Леоніда Дмитровича*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левелдизайну*

затверджена наказом по коледжу від "14" 11 20 р. № 246

2. Термін здачі студентом кваліфікаційної роботи *15.06.2025*

3. Вихідні дані до роботи *Концепт-документ ігрового проекту*
Функціональні вимоги до програмного забезпечення
Нефункціональні вимоги до програмного забезпечення
Вимоги до дизайну та інтерфейсу користувача

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналітичний огляд існуючих рішень. 2. Обґрунтування вибору технологій для створення ігрових проектів. 3. Архітектура ігрового проекту. 4. Побудова алгоритму роботи скриптів ігрової логіки. 5. Розробка ігрової механіки та левелдизайну у 3D-просторі. 6. Імплементація системи керування персонажем та взаємодії з ігровим середовищем. 7. Охорона праці. 8. Висновки. 9. Список використаних джерел інформації.

5. Перелік графічного матеріалу (слайдів мультимедійної презентації)

1. *Порівняльний аналіз існуючих рішень.*
2. *Технології створення ігрових проектів*
3. *Архітектура ігрового проекту.*
4. *Макет ігрового простору*
5. *Скріншоти /відео роботи сайту*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Іванова Л.В.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник роботи *Іванова Л.В.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз технічного завдання та пошук джерел інформації*	*4.05.2025*	*виконав*
2.	*Формування функціональних вимог до ПЗ*	*8.05.2025*	*виконав*
3.	*Визначення технічного завдання на розробку*	*10.05.2025*	*виконав*
4.	*Вибір технологій для реалізації гри*	*12.05.2025*	*виконав*
5.	*Розробка архітектури гри*	*15.05.2025*	*виконав*
6.	*Розробка алгоритмів роботи гри*	*18.05.2025*	*виконав*
7.	*Розробка елементів дизайну*	*22.05.2025*	*виконав*
8.	*Розробка елементів інтерфейсу*	*24.05.2025*	*виконав*
9.	*Створення макету гри*	*26.05.2025*	*виконав*
10.	*Створення прототипу*	*28.05.2025*	*виконав*
11.	*Тестування ПЗ*	*31.05.2025*	*виконав*
12.	*Тестування виконання вимог користувача*	*2.06.2025*	*виконав*
13.	*Перевірка роботи на плагіат*	*6.06.2025*	*виконав*
14.	*Підготовка до малого захисту*	*10.06.2025*	*виконав*
15.	*Розробка питань з охорони праці*	*12.062025*	*виконав*
16.	*Підготовка роботи до захисту та тестування ПЗ*	*15.06.2025*	*виконав*
17.	*Оформлення слайдів мультимедійної презентації*	*18.06.2025*	*виконав*

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 29. 06 000. 00 КРБ	Випускна робота		
А4			БКС 29. 06 000. 00 КРБ ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 29. 06 000. 00 КРБ		
Розробив		Головаченко Л.Д.		18.06	Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левелдизайну	Літ.	Аркуш 4 / Аркушів 75
Перевірив		Іванова Л.В.		18.06		Н Д П	
Н. Контр.		Петрашова В.І.		18.06		ВСП "ОТФК ОНТУ" гр. 2БКС-29	
Затверд.		Іванова Л.В.		18.06			

ЗМІСТ

ВСТУП

Індустрія цифрових розваг стрімко трансформується під впливом інноваційних технологій, що охоплюють комп'ютерну графіку, фізичне моделювання, штучний інтелект, мережеву взаємодію та обчислювальні платформи в реальному часі. У цьому контексті відеоігри вже давно вийшли за межі простого дозвілля, набувши значення повноцінного культурного феномена, засобу художнього самовираження, навчального інструмента і навіть соціального комунікатора. Розробка повноцінного ігрового продукту потребує міждисциплінарного підходу, що поєднує елементи програмування, дизайну, математики, фізики, психології сприйняття й аналітики користувацького досвіду.

Платформери як жанр посідають особливе місце в історії відеоігор: від класичних двовимірних аркад до сучасних тривимірних інтерпретацій із відкритим світом і розгалуженими механіками. Простота основного принципу - стрибки, переміщення, уникнення перешкод - робить цей жанр особливо зручним для експериментування з геймплейними елементами, поведінкою об'єктів та сценаріями взаємодії. Розробка платформера дозволяє водночас зосередитися на базових компонентах ігрового рушія, як система керування персонажем, фізика руху, логіка колізій, та водночас впроваджувати нові форми ігрової динаміки через інноваційні механіки.

Метою даної кваліфікаційної роботи є створення інтерактивного ігрового прототипу у середовищі Unity, заснованого на платформеній механіці з елементами вертикального переміщення, що передбачає високий рівень контролю персонажа та точне фізичне моделювання. Проект включає реалізацію цілісного ігрового середовища з активними й пасивними об'єктами, а також модулювання ігрової логіки, яка реагує на дії гравця в реальному часі. Розробка охоплює створення 3D-моделей, налаштування колайдерів, роботу з компонентами Rigidbody, написання скриптів на мові C#, а також проектування рівнів із урахуванням принципів ігрової навігації, ритму та складності.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Опис предметної області

Розробка вертикальних платформерів формується в межах значно ширшої предметної області, ніж просто створення двовимірних чи тривимірних ігор. Це окрема галузь у структурі цифрового геймдизайну, що поєднує знання з кінематики, комп'ютерної графіки, гравітаційного моделювання, психології сприйняття, штучного інтелекту та інтерактивного кіновиробництва. Така гра є не просто програмним продуктом - вона перетворюється на змодельований простір, в якому гравець активно і безпосередньо взаємодіє з фізичним середовищем, адаптованим до конкретних сценаріїв стрибків, падінь, пересування по вертикальних структурах і уникнення перешкод, що вимагає точного, динамічного і реактивного управління.

У цій предметній області питання реалістичності та інтуїтивності рухів є надзвичайно важливим. Щоб дати гравцеві відчуття контролю, розробники повинні моделювати рухи персонажів, використовуючи інерцію, прискорення, затримки реакції, контекстно-чутливу адаптивну анімацію (наприклад, різні анімації для коротких і довгих стрибків), а також багаторівневу систему зіткнень. Все це вимагає глибокого розуміння ігрових рушіїв, таких як Unity, які надають інструменти для моделювання фізики, роботи з компонентною архітектурою об'єктів та реалізації складної поведінкової логіки. Особливо складними є ситуації, коли рух персонажа потрібно координувати з мінливою геометрією оточення - рухомими платформами, нестабільними поверхнями або областями зі зміненою гравітацією, які часто використовуються в сучасних іграх.

Не менш важливим компонентом тематичної області є дизайн рівнів, який у цьому випадку виконує не тільки декоративну або навігаційну функцію, але й виступає основним джерелом виклику для гравця. Рівні повинні бути структуровані таким чином, щоб постійно змінювати темп гри, створювати ефекти вертикального прогресу або втрати, поєднувати відкриті простори з вузькими коридорами, змушувати гравця імпровізувати або розраховувати точні

траєкторії руху. Саме через просторову організацію рівня передається основна динаміка гри. Наприклад, вузька полиця над прірвою автоматично змінює настрій і вимагає обережного підходу, тоді як широкі відкриті платформи створюють відчуття безпеки. Все це інтегровано в єдину систему, яка повинна відповідати внутрішній логіці гри, уникаючи випадкових або недопрацьованих елементів, які можуть порушити відчуття занурення.

Тема гравітаційної та просторової навігації також дуже важлива в цій тематичній області. У типовій платформерній грі рух переважно горизонтальний, тоді як вертикальний прогрес передбачає багаторівневу вертикальну навігацію з частими змінами напрямку. Це вимагає не тільки технічної реалізації з боку розробника, але й ретельного балансування: занадто складний сегмент може викликати розчарування, а занадто легкий — втрату інтересу. Баланс досягається шляхом поступового збільшення складності просторових завдань, введення нових механізмів на основі знайомих дій та збільшення ризиків при збереженні відчуття контролю. Цей баланс є центральним для цієї тематичної області, оскільки він визначає рівень залучення гравців та здатність гри утримувати увагу протягом тривалого часу.

Важливо звернути увагу на роль камери в іграх такого типу. У вертикальних платформерах камера повинна бути достатньо гнучкою, щоб гравець міг бачити майбутні дії, оцінювати відстань стрибка і передбачати безпечні точки приземлення, але в той же час вона не повинна викликати дезорієнтацію. Найчастіше використовується кінематична камера, яка має певну інерцію, слідує за гравцем із затримкою або змінює масштаб відповідно до контексту. Саме завдяки грамотному розміщенню камери гравець може відчувати себе частиною ігрового світу, а не просто стороннім спостерігачем.

Анімація персонажів, взаємодія з поверхнями, фізична логіка та розміщення об'єктів у просторі — все це, в рамках даної тематики, є частиною єдиної системи, в якій важливе кожне налаштування. З точки зору розробки, навіть незначна зміна прискорення або часу затримки перед стрибком може повністю змінити відчуття від гри. Ось чому розробка платформера з елементами

вертикального руху вимагає не тільки розуміння технологічної основи, але й глибокої чутливості до поведінки гравця, його реакцій, очікувань і меж терпіння. Все це становить предмет дослідження — взаємодія технічних, дизайнерських і психологічних компонентів у контексті побудови контрольованого ігрового середовища, яке заохочує активну участь, дослідження та вдосконалення навичок.

1.2 Аналітичний огляд існуючих рішень

Огляд існуючих рішень у сфері платформерів з вертикальним просуванням дозволяє виявити основні закономірності у побудові ігрової логіки, механік взаємодії, дизайну рівнів і загального темпу геймплею. Такі проекти, як Only Up!, Jump King, Getting Over It with Bennett Foddy або Celeste, хоча й реалізовані у різних стилях, мають спільні структурні риси, які дозволяють виділити ефективні підходи до конструювання складного, але привабливого користувацького досвіду.

Проект Only Up!, створений на Unreal Engine, став популярним завдяки гранично простій механіці вертикального пересування та акценту на безперервному русі без точок збереження. Гравець повинен обережно просуватись вгору, уникаючи падінь, що миттєво повертають на значну відстань назад. Відсутність автоматичного збереження тут не є недоліком, а навпаки - ключовою частиною дизайну, що формує напруження, вимагає концентрації та дарує унікальне емоційне переживання. Проект використовує просту та чітко вибудувану систему колізій і великі сегментовані локації з поступовим наростанням складності, що дозволяє керувати геймплейним ритмом.

Структура рівнів у Only Up! реалізована за принципом вертикальної експедиції, у якій кожна нова ділянка вимагає від гравця адаптації до змінених умов пересування. У межах однієї ігрової сесії доводиться взаємодіяти з різноманітними платформами, механізмами, рухомими елементами та фізичними перешкодами, що значно ускладнює орієнтацію в просторі та посилює потребу в точності рухів. Візуальні контрасти між зонами - від індустріальних локацій до стилізованих абстрактних об’єктів - створюють ефект сюрреалістичної подорожі, в якій кожен сегмент не лише додає нових механічних викликів, а й формує нові

естетичні враження. Просторові рішення побудовані так, щоби підтримувати постійне відчуття прогресу, попри можливість падінь, які часто мають глибоко фруструючий, але водночас мобілізуючий характер.

Рисунок 1.1. Геймплей гри OnlyUp!

Ключовим компонентом взаємодії гравця з навколишнім середовищем є точне налаштування фізики руху персонажа. Інерція, вага, тривалість і висота стрибка налаштовані так, щоби досягти балансу між чутливим управлінням та складністю контролю. Цей баланс особливо важливий у грі, де будь-яка похибка призводить до втрати прогресу. Водночас саме таке налаштування створює ефект навчання через досвід - повторення одних і тих самих ділянок дозволяє гравцеві вдосконалювати навички, розвивати моторну пам’ять і поступово опановувати ігровий простір як цілісну структуру. Завдяки цьому Only Up! зберігає високу реіграбельність, навіть за відсутності класичних ігрових винагород або сюжетних стимулів.

Проект Jump King, реалізований у стилі піксельної 2D-графіки, зосереджується виключно на єдиній механіці - стрибку з фіксованою траєкторією, який потрібно точно розрахувати. Уся ігрова динаміка базується на довжині натискання кнопки, і ця механіка не змінюється протягом усієї гри, що створює

ілюзію простоти. Водночас рівні сконструйовані таким чином, що вимагають дедалі точнішого виконання рухів, і кожна помилка має значну ціну. Така структура підкреслює глибину, якої можна досягти навіть у межах однієї механіки, за умови детально продуманого левелдизайну.

Рисунок 1.2. Геймплей гри Jump King.

Особливість Jump King полягає у відсутності прямого контролю над напрямом руху персонажа в повітрі, що змушує гравця максимально зосереджуватись на фазі підготовки до стрибка. Необхідність ретельно оцінювати кожну поверхню, визначати силу натискання і прогнозувати результат руху формує геймплей як послідовність обережних рішень, де кожна дія має стратегічне значення. Водночас відсутність музичного супроводу підсилює атмосферу ізоляції й фокусує увагу на звукових ефектах, які стають частиною геймплейного ритму. Стрибок, удар об поверхню, звук падіння - усе це трансформується в тактильний досвід, що фіксується на рівні моторної пам’яті.

Рівнева архітектура Jump King побудована за принципом вертикального нарощення складності з обмеженим зоровим полем, що ускладнює передбачення структури наступних платформ. Це змушує гравця розвивати навички навчання через спроби й помилки, запам’ятовувати конфігурацію середовища та вивчати поведінку персонажа в різних ситуаціях. Такий підхід до дизайну не лише

стимулює уважність, а й створює характерну для жанру «precision platformer» напругу. У результаті гра не пропонує системи винагород у класичному розумінні, проте кожна подолана ділянка стає самостійним досягненням, яке має внутрішню цінність і формує відчуття поступу навіть без додаткових стимулів.

Гра Celeste, хоча й не є вертикальним платформером у вузькому сенсі, пропонує багатий набір рухових можливостей персонажа - подвійний стрибок, ривки в повітрі, захоплення за стіни - і вбудовує їх у короткі, але складні секції рівнів. Унікальність Celeste полягає у високій точності керування, емоційно насиченому наративі та ідеально збалансованій кривій складності. Тут важливу роль відіграє не лише рівень майстерності гравця, а й естетичне сприйняття простору: візуальна стилістика та анімації подають інформацію про безпечні поверхні, потенційні маршрути та ризики, що дозволяє будувати стратегії руху на ходу. Завдяки високому рівню полірування усіх компонентів проекту Celeste став еталонним зразком дизайну керованості в інді-платформерах.

Рівнева структура Celeste складається з невеликих, чітко відмежованих кімнат, кожна з яких презентує окремий геймплейний виклик і вимагає засвоєння певної рухової техніки або комбінації дій. Такий підхід дозволяє гравцеві концентруватись на конкретному завданні, поступово нарощуючи складність через зміни конфігурації перешкод і варіативність рухомих елементів. Системи збереження й миттєвого відродження після невдачі формують ритм навчання без покарання за помилку, що забезпечує комфортне середовище для експериментів. Цей дизайн підтримує відчуття прогресу навіть у межах складних ділянок, де кожна смерть розглядається як крок до вдосконалення навичок, а не як фрустраційна перешкода.

Анімаційна мова Celeste функціонує не лише як візуальний стиль, а як комунікативний інструмент, що допомагає гравцеві миттєво зчитувати стан персонажа, відгук на керування й умови навколишнього середовища. Наприклад, колір персонажа змінюється залежно від того, чи був уже використаний повітряний ривок, а дрібні зміни в рухах або позиції сигналізують про зчеплення зі стіною чи втрату імпульсу. Це створює тісний зв'язок між візуальними ефектами

та геймплейною механікою, який дозволяє гравцю інтуїтивно приймати рішення в реальному часі. Завдяки такому рівню злагодженості між контролем, фізикою та візуальною підтримкою Celeste не лише пропонує високий рівень точності, а й викликає відчуття гармонії між гравцем і грою, що є рідкісним досягненням у жанрі платформерів.

Рисунок 1.3. Геймплей гри Celeste

Getting Over It with Bennett Foddy демонструє радикально інший підхід. Замість традиційних кнопкових механік тут використовується мишкове керування молотом, через який герой взаємодіє з навколишнім середовищем. Це створює ефект повної фізичної неконтрольованості, що, втім, і є джерелом напруги й цікавості. Незважаючи на те, що графіка тут доволі примітивна, механіка спонукає до повторення, експериментів і формує глибоко індивідуальний стиль проходження. Структура рівня є неперервною, без чітко виокремлених сегментів, що стимулює почуття прогресу, незважаючи на численні відкотування назад.

Ключовий акцент Getting Over It - це абсолютна залежність від моторики руки гравця, що трансформує процес пересування у своєрідну взаємодію з віртуальною фізикою без звичних правил чи очікуваних обмежень. Завдяки відсутності стандартного навчання або поступового введення механік, гра одразу

занурює в середовище, де будь-який рух має наслідки, а відчуття контролю здобувається виключно шляхом багаторазових спроб. Сама форма управління, що нагадує маніпуляцію реальним інструментом, викликає надзвичайно тілесне, майже тактильне сприйняття геймплею, де провали сприймаються не як цифрові помилки, а як особисті прорахунки. У такий спосіб створюється парадоксальна ситуація: чим примітивніший інтерфейс, тим глибше й унікальніше гравець залучається в процес.

Рисунок 1.4. Геймплей гри Getting Over It with Bennett Foddy

Неперервна структура рівня не лише підсилює відчуття реального простору, а й підкреслює філософську складову гри, у якій падіння не є кінцем, а лише черговою фазою циклу. Тут відсутній будь-який зовнішній стимул для продовження, крім внутрішньої мотивації подолати виклик. Тексти-наратії, озвучені самим Беннетом Фодді, підкреслюють цей настрій, коментуючи невдачі, фрустрацію й завзятість гравця в іронічному, а іноді й співчутливому тоні. Цей елемент перетворює гру на інтимне переживання, де технічна складність переплітається з особистісним досвідом. У такій конфігурації дизайн рівня стає не просто серією перешкод, а цілісною метафорою психологічного випробування, що вимагає не лише технічної точності, а й витримки, терпіння та саморефлексії.

Усі розглянуті ігри демонструють різні підходи до вирішення завдань вертикального просування та створення викликів для гравця. Вони також ілюструють, як фундаментальні механіки - стрибки, контроль над інерцією, захоплення за платформи, балансування - можуть трансформуватись у повноцінну геймплейну структуру за умов ретельної реалізації рівнів, візуальної комунікації та налаштування фізичних параметрів. Кожен із прикладів, незважаючи на технічні або художні обмеження, формує унікальну динаміку гри через інтеграцію механік, дизайну середовища та інтерфейсних рішень.

На основі проведеного огляду проектів Only Up!, Jump King, Celeste та Getting Over It with Bennett Foddy простежується низка дизайнерських і технічних рішень, які мають високий потенціал для подальшого використання, адаптації або переосмислення в межах розробки власного ігрового проекту. Особливий інтерес становить принцип поступового вертикального просування без традиційних збережень, що забезпечує безперервний геймплей з високим рівнем напруги та внутрішньої мотивації. Подібна структура дозволяє сформувати відчуття справжньої подорожі, де кожна помилка не лише повертає назад, а й спонукає до переосмислення попередньої стратегії проходження.

Ретельне налаштування фізики переміщення гравця, включно з інерцією, вагою, тривалістю стрибка, а також обмеженням контролю під час польоту, є критичним компонентом, який визначає ігрову динаміку. Цей підхід буде адаптовано у вигляді високоточної фізичної моделі руху, орієнтованої на виклик, але водночас такої, що підтримує відчуття справедливості. Особлива увага приділятиметься варіативності поверхонь, а також використанню фрагментованих сегментів, які поступово підвищують складність і вимагають від гравця вдосконалення моторних навичок.

Ідеї фізичного залучення гравця, які продемонстровані в Getting Over It, також вплинуть на загальну концепцію: передбачено створення керування, яке вимагає постійної ручної точності, обережності й розуміння фізичних наслідків кожного руху. Проте у межах даного проекту буде збережено більш класичну систему контролю, проте з акцентом на складність і моторне навчання через

повторення.

Левелдизайн матиме вертикальну спрямованість із відсутністю проміжних збережень, але з можливістю умовного візуального зонування рівнів, як це реалізовано в Only Up!, що допоможе підтримувати психологічне відчуття прогресу. Кожна зона буде відрізнятись не лише візуально, а й механічно, що дозволить сформувати багаторівневий ігровий досвід з поступовим введенням нових викликів. У такий спосіб ігровий проект буде не просто поєднанням механік, а інтегрованою системою, яка черпає найефективніші рішення з розглянутих зразків і трансформує їх у цілісну авторську структуру.

1.3 Обґрунтування вибору технологій для створення ігрових проектів

Unity у поєднанні з Visual Studio Code становлять потужний інструментальний комплекс для створення сучасних ігрових проектів, що базуються на високих вимогах до інтерактивності, візуальної привабливості й продуктивності. Такий вибір зумовлений рядом технологічних, архітектурних та функціональних переваг, що забезпечують ефективний повний цикл розробки - від початкового прототипування до релізу готового продукту.

Ігровий рушій Unity є одним із найпоширеніших серед незалежних розробників та інді-студій завдяки своїй мультиплатформеній природі, широкому інструментарію, а також високому рівню абстракції, що дає змогу сконцентруватися безпосередньо на творчих аспектах проекту. Unity підтримує реалізацію двовимірних і тривимірних середовищ, має вбудовану фізику, системи частинок, анімаційний контролер, редактор матеріалів, підтримку шейдерів та механізми побудови світла. Особливо важливою є наявність інтегрованого компонентного підходу до побудови ігрових об’єктів, що дозволяє реалізовувати складну поведінку через композицію незалежних скриптів. Це відкриває широкі можливості для модульного проектування, повторного використання коду та масштабування логіки без порушення архітектурної цілісності.

Unity має добре розвинену екосистему, до складу якої входить офіційний магазин активів Unity Asset Store, де доступні тисячі готових рішень - від моделей, анімацій, аудіо- та візуальних ефектів до повноцінних шаблонів ігрових систем. Це суттєво скорочує час на розробку технічних підсистем, дозволяючи зосередитись на унікальних аспектах гри. В контексті платформера з вертикальним просуванням це особливо важливо, оскільки дає змогу використовувати існуючі інструменти для створення фізично правдоподібного руху, керування камерою, обробки колізій тощо, адаптуючи їх під конкретні вимоги геймплею.

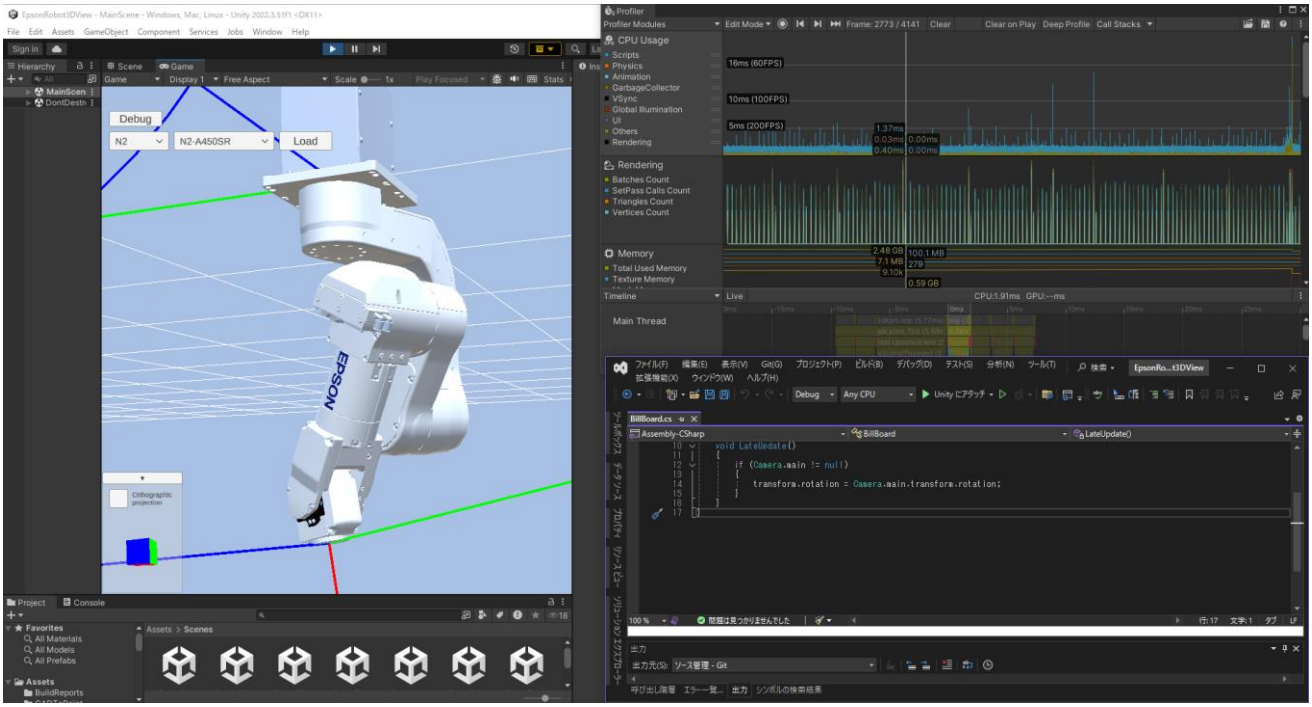


Рисунок 1.5. Зовнішній вигляд інструментарію Unity

Вибір Visual Studio Code як середовища для написання коду є раціональним у зв’язку з його легкістю, швидкодією та підтримкою широкого спектра розширень. Завдяки інтеграції з Unity через офіційні плагіни, Visual Studio Code надає доступ до функцій автодоповнення, дебагінгу, відстеження змін у версійному контролі та навігації по класах і методах. Його гнучка система налаштувань дозволяє створити комфортне середовище для розробника, незалежно від стилю написання коду чи розміру проекту.

Unity використовує C# як основну мову програмування, що є сучасною, об’єктно-орієнтованою та підтримує широкий спектр парадигм - від

імперативного до реактивного програмування. C# забезпечує високий рівень читабельності та підтримуваності коду, що є ключовим чинником у командних проектах і при роботі над довготривалими розробками. Завдяки використанню середовища .NET, розробник отримує доступ до великої кількості бібліотек і фреймворків, які можуть бути залучені для реалізації нестандартних задач, зокрема роботи з мережею, обробки даних, математичних обчислень тощо.

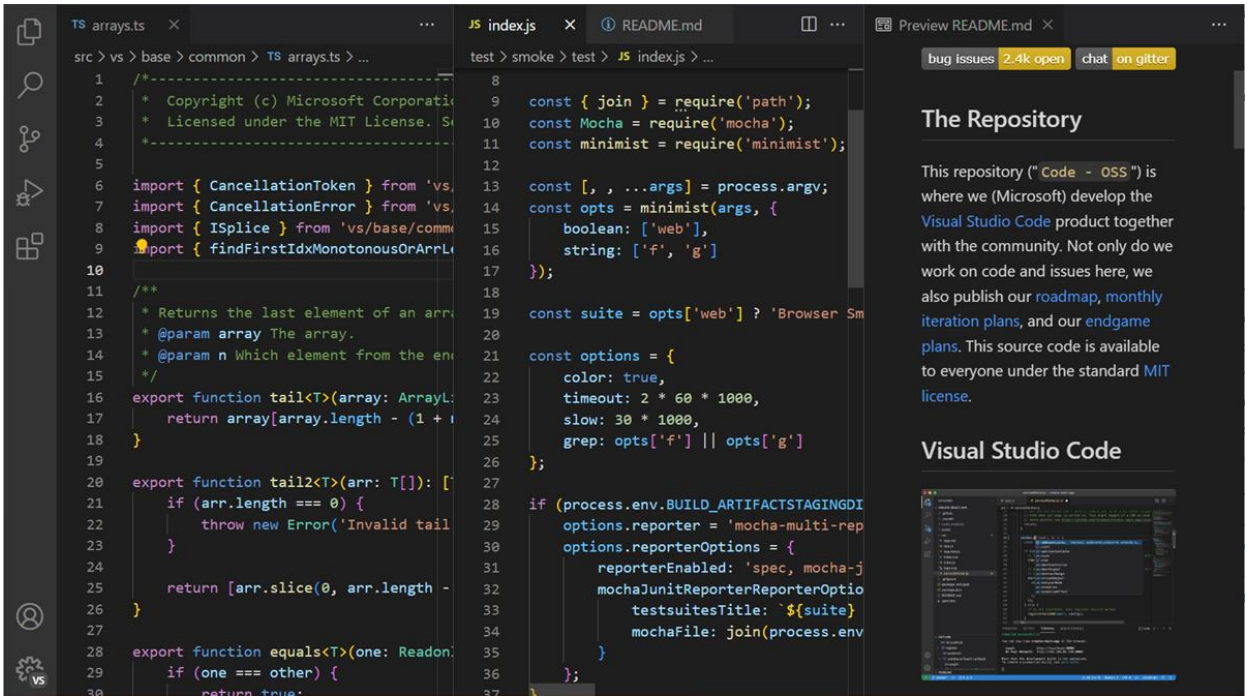


Рисунок 1.6. Зовнішній вигляд інструментарію Visual Studio Code

Така комбінація технологій є не лише технічно обґрунтованою, а й стратегічно доцільною в контексті проектів із високим рівнем динаміки, просторової навігації, деталізації середовища та складних анімаційно-фізичних взаємодій. Вона дозволяє втілити вертикально-орієнтовану механіку з точним контролем, складною геометрією рівнів і реактивним візуальним зворотним зв’язком, що критично важливо для створення захопливого ігрового досвіду у вертикально структурованому платформері.

Технічна стабільність платформи Unity, її постійну підтримку та розвиток з боку спільноти і самої компанії-розробника. Існування великої бази знань, офіційної документації, навчальних курсів і форумів дозволяє ефективно вирішувати типові проблеми й знаходити оптимальні підходи до реалізації конкретних функцій.

В реалізації ігрового процесу на рушії Unity заслуговує уваги система побудови фізики, яка дозволяє створювати реалістичні взаємодії між об’єктами в тривимірному середовищі. У випадку гри з вертикальним просуванням та паркурною динамікою, фізичний рушій відіграє ключову роль у створенні переконливої симуляції руху, гравітаційної інерції, а також точного відтворення зіткнень та обмежень. Використання Rigidbody, колайдерів різних типів, фізичних матеріалів і обробників подій дає змогу налаштувати інтуїтивно зрозумілу та водночас складну систему керування персонажем, що є критично важливим для платформерів, де гравець постійно перебуває в русі та здійснює точні переміщення по складній геометрії рівня.

До фізичної системи Unity, важливим компонентом є механізм анімації, заснований на Animator Controller. Він дозволяє створювати гнучкі переходи між станами анімацій, визначати умови їхнього запуску й синхронізувати візуальну поведінку з подіями геймплею. У грі з паркурними елементами правильна анімація стрибків, лазіння, балансування, падіння та інерційних рухів забезпечує відчуття фізичної присутності, знижує когнітивне навантаження та сприяє створенню ефекту повного занурення. Система подій анімації також дозволяє інтегрувати тригери до ігрової логіки, наприклад, запускати звукові ефекти, змінювати швидкість камери або активувати додаткові механіки, що посилює інтерактивність.

Паралельно з роботою над фізикою та анімацією, важливо враховувати організацію керування камерою. В іграх вертикального типу, де основне завдання полягає у підйомі по складній, небезпечній геометрії, камера повинна не лише передавати масштаб середовища, а й динамічно адаптуватися до контексту: віддалятись або наближатись до гравця, змінювати кут огляду, слідкувати за переміщенням із певною інерцією або фіксуватись у визначених зонах. Unity надає всі необхідні засоби для цього, зокрема систему Cinemachine, яка дозволяє налаштовувати камеру як реактивний об’єкт, що динамічно реагує на зміну ситуацій у грі, зберігаючи водночас плавність і контрольованість руху. Це критично важливо для створення напруги, контролю ритму гри та надання

гравцеві чіткої візуальної інформації про навколишнє середовище.

Використання Visual Studio Code як основного редактора для роботи з C#-кодом також створює сприятливі умови для підтримки чистої архітектури гри. Його інтеграція з системами контролю версій (наприклад, Git), підтримка форматування коду, система перевірки синтаксису та доступ до вбудованої документації дозволяють ефективно управляти складними ігровими системами. Для створення систем контролю стану, обробки подій, менеджменту ресурсів і оптимізації продуктивності критично важливим є чітке структурування коду, що значно полегшується завдяки розширенням, доступним у VSC.

Ця взаємодія між рушієм Unity та редактором Visual Studio Code формує цілісну розробницьку екосистему, що дозволяє реалізувати глибоко інтегровані механіки, складні логіки рівнів та продуману взаємодію гравця з навколишнім світом без втрати контрольованості чи стабільності коду. Усі ці технологічні компоненти створюють основу для побудови проекту, що поєднує технічну досконаліть з емоційним залученням гравця, відкриваючи можливість для подальшого розширення функціональності, нарощення складності рівнів, інтеграції нових типів взаємодії й оптимізації процесів розробки.

1.4 Побудова алгоритму роботи скриптів ігрової логіки

Побудова алгоритму роботи скриптів ігрової логіки є етапом, на якому відбувається формалізація поведінки ігрових об’єктів через послідовні, структуровані дії, що виконуються відповідно до певних правил, умов та подій. Саме тут абстрактні механіки, задумані в межах ігрового дизайну, набувають чіткої програмної форми, стаючи частиною взаємопов’язаної системи, у якій кожен компонент реагує на сигнали ззовні або змінює свій стан у відповідь на внутрішню логіку сцени. Завдяки побудові алгоритмів визначається точний порядок обробки введення, умов переходів між станами, пріоритетів дій, взаємодії з середовищем та запуску анімацій.

У цьому розділі описується, як розроблена система скриптів реалізує рух, керування, обробку подій, візуальний зворотний зв’язок і просторову адаптацію у

відповідь на зміну контексту гри. Розглядаються як окремі алгоритмічні блоки (наприклад, перевірка наявності землі чи запуск ривка), так і взаємодія між ними. Увага зосереджується на послідовності, з якою виконуються дії, умовах активації та способі обміну інформацією між різними скриптами, що дозволяє досягти керованої, передбачуваної й водночас адаптивної ігрової поведінки.

1.4.1 Логіка контролера руху персонажа

У структурі руху персонажа реалізовано механізм, що безперервно реагує на дії гравця, змінні умови навколишнього середовища та фізичні властивості простору. Усе функціонування системи базується на активному відстеженні поточного стану: чи є контакт із поверхнею, чи відбувається переміщення, чи виконуються додаткові маневри - стрибок, ривок, зміна напрямку або швидкості. У центрі всієї логіки перебуває система введення - вона формує сигнали, що визначають наміри користувача, після чого ці наміри трансформуються у конкретні дії відповідно до правил гри та ситуації, в якій опинився персонаж.

Переміщення виконується з урахуванням напрямку, який диктує камера: персонаж орієнтується у просторі відповідно до її погляду, і тому рух залишається інтуїтивно зрозумілим незалежно від кута огляду. Якщо персонаж перебуває на пологій поверхні, рух відбувається з плавним зчепленням; якщо ж нахил надмірний, система забороняє втримання, і виникає ефект зісковзування. Натискання кнопки стрибка активує перевірку: чи дозволено виконати цю дію у поточний момент. Якщо це первинний стрибок - достатньо контакту з землею; для другого - перевіряється наявність дозволу та те, чи ще не був використаний додатковий імпульс. Ривок діє незалежно від положення: це миттєвий рух уперед, який додається до загальної траєкторії та дозволяє реагувати на складні просторові виклики.

Під час взаємодії з динамічними елементами середовища, зокрема рухомими платформами, положення персонажа коригується в реальному часі. Це виключає ситуації розриву між візуальним становищем і фізичним розрахунком, забезпечуючи сталість і логіку просторової прив’язки. Усі зміщення, імпульси та корекції руху обчислюються у вигляді векторів, які передаються системі

переміщення, де застосовуються з урахуванням гравітації, опору й кутів нахилу. Завдяки такій побудові переміщення стає плавним, контрольованим і природним.

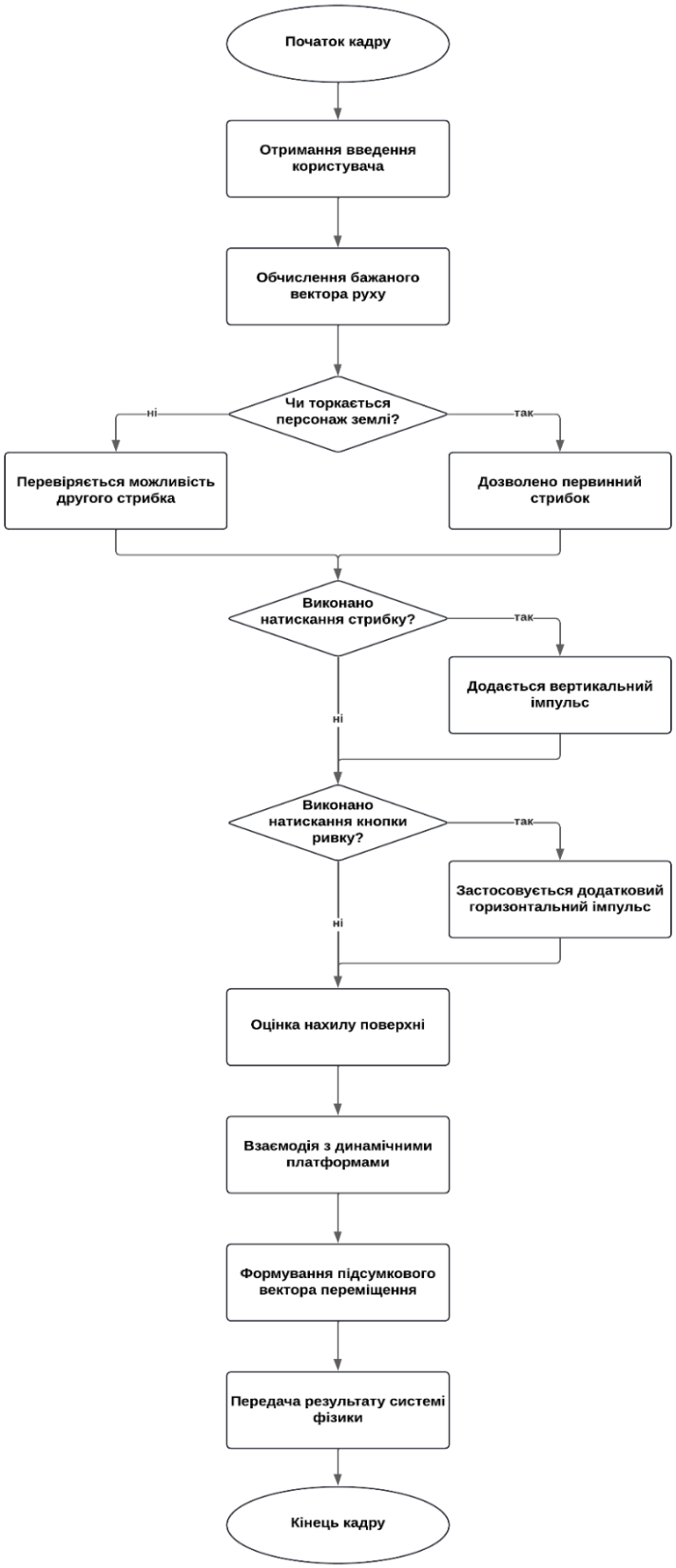


Рисунок 1.7. Функціональна схема контролера руху персонажа

1.4.2 Логіка роботи камери

Просторова поведінка камери у грі з видом згори організована так, щоби забезпечити гравцеві стабільний і зрозумілий контроль за ситуацією, незалежно від складності навігації. Вона не просто закріплюється над персонажем, а працює як жива система - здатна адаптуватися до змін, реагувати на рух, компенсувати швидкість і прогнозувати потребу в огляді наперед. Завдяки зміщенню по вектору руху, яке додається до основної позиції, камера ніби завжди дивиться трохи вперед, дозволяючи заздалегідь побачити потенційні загрози або варіанти дій. Це важливо у складних просторових структурах, де обмеження видимості можуть призводити до втрати контролю або помилкових рішень.

Під час переміщення персонажа камера зберігає задану відстань, але ніколи не перебуває в жорсткій прив'язці - навпаки, вона «пливе» за ним, вирівнюючи своє положення із запізненням, що створює ефект інерції. Така динаміка надає руху плавності, усуває ривки й робить зміну кадру менш помітною, що позитивно впливає на загальне враження від гри. Замість того, щоб бездумно повторювати координати, камера обчислює нову позицію як результат поєднання трьох векторів: положення персонажа, зміщення назад і зміщення вперед. Це дозволяє зберігати баланс між фіксованістю й рухливістю, уникати ефекту закритого простору або зайвої розфокусованості.

Окремий механізм обертання виконує функцію керованого перегляду сцени. Гравець може вручну змінювати кут огляду, впливаючи на композицію кадру, але без ризику порушити розташування персонажа чи логіку управління. Це дає змогу планувати дії у вертикальних чи багаторівневих структурах, особливо коли видимість обмежена статичними об'єктами або природною геометрією сцени. Обертання відбувається з постійною швидкістю, що дозволяє гравцеві поступово перелаштувати уявний простір і зберегти орієнтацію в ньому.

У моменти різкого маневрування або переходу між ділянками рівня камера не змінює позицію миттєво, а використовує інтерполяцію, яка згладжує рух і компенсує раптові стрибки або зміни напрямку. Це особливо важливо в проектах, орієнтованих на точний контроль і тактильне відчуття простору. Камера не

відволікає увагу, не створює візуального шуму, не змінює фокус без причини - вона діє як частина живого механізму, що перебуває у повній координації з діями гравця. Таким чином, візуальна навігація в грі стає не лише фоном, а повноцінним інструментом управління, що формує просторову логіку й емоційний ритм гри.

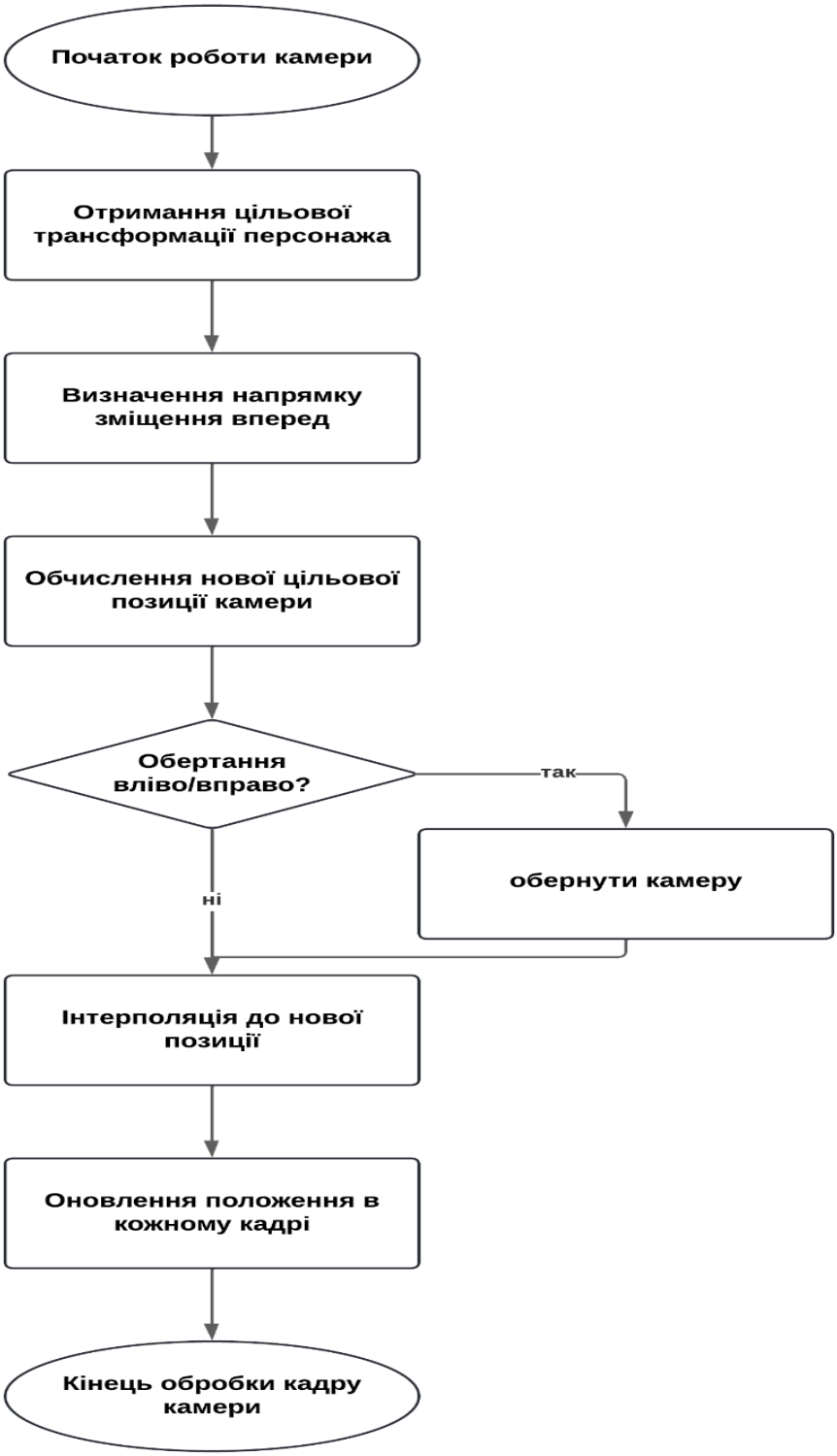


Рисунок 1.8. Функціональна схема роботи камери

1.4.3Логіка керування персонажем

У побудові логіки керування персонажем важливу роль відіграє не лише спосіб передавання команд, а й структура, яка визначає, як ці команди інтерпретуються, організовуються та перетворюються в дії. У запропонованій системі використовується абстрактна модель введення, що дозволяє повністю відокремити логіку руху від конкретного джерела введення. У центрі цієї архітектури - базовий клас, який виконує роль шаблону для будь-якого типу керування. Він визначає набір дій, які можуть бути реалізовані в грі, але не містить жодної реалізації сам по собі. Це означає, що жоден конкретний пристрій, жодна клавіша чи геймпадна кнопка не вбудовані в структуру наперед - натомість закладено механізм, що дає змогу створювати адаптивні модифікації для різних платформ або типів контролю.

Фактична реалізація керування покладається на похідні класи, що розшифровують абстрактні дії у конкретні інпут-сигнали. У випадку класу, орієнтованого на стандартну клавіатуру, кожна дія гравця співвідноситься з певною клавішею або напрямком осі. Натискання на кнопки, які відповідають за рух уперед або назад, вліво або вправо, перетворюється на числові значення, які далі обробляються скриптом руху. Окремо виділяються події, що мають чітку тригерну природу - такі як стрибок, ривок або скидання предмета. Вони фіксуються як одиничні події й запускають визначену логіку в контролері руху.

Перевага цієї системи полягає в її масштабованості: для того щоб змінити тип керування, не потрібно переписувати основну логіку гравця - достатньо реалізувати новий похідний клас, який забезпечить відповідь на ті самі запити. Завдяки цьому створюється гнучка інфраструктура, у якій один і той самий контролер може працювати і з клавіатурою, і з геймпадом, і з мобільним інтерфейсом, і навіть з алгоритмом, який імітує поведінку штучного гравця. В основі лежить принцип поліморфізму: кожен об'єкт має однакову зовнішню форму, але різну поведінку всередині. Це знижує складність підтримки, дозволяє легше тестувати й оновлювати систему без ризику порушити вже наявну логіку.

Рисунок 1.9. Функціональна схема роботи логіки керування персонажем

У структурі ігрового циклу ця система працює як проміжна ланка. В кожному кадрі скрипт руху персонажа запитує поточне значення осей руху, стан кнопки стрибка, ривка чи іншої дії. Він не "знає", що саме натиснув користувач і яким пристроєм це зроблено - його цікавить лише відповідь: чи активна дія, і в

якому напрямку. Це робить код руху універсальним, придатним до використання в будь-якому контексті, незалежно від конфігурації гравця або платформи. Завдяки чітко сформульованому інтерфейсу введення і правильному розділенню зон відповідальності, вдається зберегти архітектурну чистоту, підвищити стабільність роботи гри та забезпечити адаптивність до майбутніх змін.

Така організація введення забезпечує не лише технічну ефективність, а й дизайнерську гнучкість, створюючи умови, в яких контроль за персонажем може бути водночас точним, передбачуваним і адаптованим до різних сценаріїв гри.

1.5 Розробка ігрової логіки за допомогою скриптів

Розробка ігрової логіки за допомогою скриптів є ключовим етапом створення інтерактивного середовища, що визначає поведінку об’єктів, реакцію на дії гравця та загальну динаміку геймплею.

У рамках реалізації проекту з вертикальним просуванням, особлива увага приділяється точному відтворенню рухових характеристик персонажа. Створення логіки стрибків, прискорень. Скрипти повинні синхронізуватись із системами анімацій, щоб кожна дія мала не лише фізичне, а й візуальне підтвердження. Це досягається шляхом поєднання State Machine Behaviour у анімаційному контролері з подієвою логікою, прописаною в коді, що дозволяє формувати плавні переходи між станами: наприклад, з розгону в стрибок, із зависання - у падіння, з приземлення - у ковзання.

Реалізація геймплейних ситуацій у вертикальному платформері передбачає динамічну реакцію на складні послідовності дій гравця, де одна помилка може призвести до втрати значного прогресу. Для цього необхідне створення високонадійної системи перевірки умов і тригерів. Наприклад, скрипти, що обробляють зони небезпеки, перемикачі, платформу з обмеженим терміном дії чи реакцією на вагу, повинні мати точні умови активації та механізми виключення конфліктів. Завдяки використанню подій, делегатів і системи Tag/Layer Management в Unity забезпечується масштабована архітектура, здатна до подальшого ускладнення без потреби у переписуванні вже створених модулів.

1.5.1 Реалізація скрипту Inputs

Цей скрипт є абстрактним класом, який визначає інтерфейс взаємодії для системи введення в рамках ігрового проекту. Його структура вказує на те, що він призначений для наслідування іншими класами, які реалізовуватимуть конкретну поведінку, пов'язану з обробкою вводу від користувача. Завдяки ключовому слову abstract, кожен метод цього класу виступає як обов'язковий для реалізації в дочірніх компонентах, і не містить власної логіки - він лише задає сигнатуру, тобто формат того, як саме інші компоненти мають повертати відповідні значення.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public abstract class Inputs : MonoBehaviour
6  {
7      public abstract float GetHorizontal();
8
9      public abstract float GetVertical();
10
11     public abstract bool Jump();
12
13     public abstract bool Dash();
14 }
```

Рисунок 1.10. Скрипт інтерфейса взаємодії для системи введення

Функції GetHorizontal() та GetVertical() очікують повернення значень типу float, які відповідають аналоговим напрямкам руху персонажа або об'єкта. У випадку з GetHorizontal, йдеться про ліво-право, а з GetVertical - про вгору-вниз або вперед-назад залежно від розташування у просторі сцени. Ці методи зазвичай використовуються для керування переміщенням персонажа або подачею сигналів у фізичну систему руху, де значення можуть варіюватися від -1 до 1, що дає змогу враховувати як напрям, так і інтенсивність вводу.

Усі інші методи повертають булеві значення, що означає наявність або відсутність певної дії у конкретний момент часу. Метод Jump() сигналізує про намір виконати стрибок, Dash() - про виконання короткого імпульсного руху.

1.5.2 Реалізація скрипту PlayerInput

Цей скрипт відіграє роль модуля введення, що реалізує інтерфейс взаємодії гравця з керованим персонажем. Він знаходиться у просторі імен PlatformCharacterController і є спадкоємцем абстрактного класу Inputs, який, найімовірніше, задає базову структуру для управління рухом, стрибками та іншими діями. Основне завдання цього класу - забезпечити гнучке і централізоване зчитування вхідних даних з клавіатури або інших пристроїв керування.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  namespace PlatformCharacterController
6  {
7      [RequireComponent(typeof(MovementCharacterController))]
8      public class PlayerInput : Inputs
9      {
10         public override float GetHorizontal()
11         {
12             return Input.GetAxis("Horizontal");
13         }
14
15         public override float GetVertical()
16         {
17             return Input.GetAxis("Vertical");
18         }
19
20         public override bool Jump()
21         {
22             return Input.GetKeyDown(KeyCode.Space);
23         }
24
25         public override bool Dash()
26         {
27             return Input.GetKeyDown(KeyCode.F);
28         }
29     }
30 }
```

Рисунок 1.11. Скрипт модуля введення

У функціях GetHorizontal та GetVertical здійснюється звернення до стандартних осей Unity - "Horizontal" і "Vertical", які зазвичай прив’язані до клавіш A/D або стрілок вліво/вправо (для горизонтального руху), а також W/S або стрілок вгору/вниз (для вертикального). Це дозволяє забезпечити універсальне зчитування напрямку руху, який потім інтерпретується контролером персонажа як рух вперед, назад, ліворуч чи праворуч у координатній системі сцени або відносно камери.

Метод Jump() реагує на натискання клавіші Space, що інтерпретується як команда на стрибок. Оскільки використовується Input.GetKeyDown, ця команда активується лише один раз у момент натискання клавіші, що дозволяє уникнути багаторазового виклику дії під час тривалого натискання.

Аналогічно функціонує метод Dash(), який відповідає за ривок персонажа при натисканні клавіші F. Як і у випадку з Jump(), обраний метод зчитування клавіші дозволяє уникнути повторного виконання команди впродовж кількох кадрів. Це особливо важливо для механіки ривка, яка часто супроводжується коротким, але динамічним переміщенням або ефектом.

Цей скрипт не виконує ніяких дій безпосередньо у собі, а лише надає абстрактному класу Inputs конкретні реалізації методів, завдяки чому контролер персонажа може опитувати стани клавіш незалежно від конкретної реалізації джерела введення. Таким чином, у разі потреби, цю реалізацію можна легко замінити на іншу - наприклад, для гри на геймпаді або для мобільних пристроїв, - не змінюючи основну логіку контролера.

1.5.3 Реалізація скрипту MovementCharacterController

У цьому коді реалізовано основну ігрову механіку управління персонажем у 3D середовищі на платформі Unity, з використанням компоненту CharacterController. Клас MovementCharacterController забезпечує рух, стрибки, подвійні стибки, ривки, а також взаємодію з похилими поверхнями, платформами та іншими умовами, характерними для платформерів. Далі розглянуто детально, як працюють основні елементи скрипту.

У базових налаштуваннях переміщення, змінні RunningSpeed, Gravity, MaxDownYVelocity визначають швидкість гравця, силу гравітації та граничну швидкість падіння.

Метод FixedUpdate() щосекунди виконує обчислення напряму руху гравця, нормалізує його та застосовує пересування за допомогою CharacterController.Move() відповідно до рисунку 1.12.

```
private void FixedUpdate()
        {
            _forward = _cameraTransform.TransformDirection(Vector3.forward);
            _forward.y = 0f;
            _forward = _forward.normalized;
            _right = new Vector3(_forward.z, 0.0f, -_forward.x);

            _move = (_horizontal * _right + _vertical * _forward);
            _direction = (_horizontal * _right + _vertical * _forward);

            if (!_isCorrectGrounded && _isGrounded)
            {
                _move.x += (1f - _hitNormal.y) * _hitNormal.x * (1f - SlideFriction);
                _move.z += (1f - _hitNormal.y) * _hitNormal.z * (1f - SlideFriction);
            }

            _move.Normalize();

            _isCorrectGrounded = (Vector3.Angle(Vector3.up, _hitNormal) <= SlopeLimit);

            if (_direction != Vector3.zero)
            {
                transform.forward = _direction;
            }

            if (_velocity.y >= -MaxDownYVelocity)
            {
                _velocity.y += Gravity * Time.deltaTime;
            }

            _velocity.x /= 1 + DragForce.x * Time.deltaTime;
            _velocity.y /= 1 + DragForce.y * Time.deltaTime;
            _velocity.z /= 1 + DragForce.z * Time.deltaTime;
            if (_controller.enabled)
            {
                _controller.Move(_velocity * Time.deltaTime);
            }

            SetGroundedState();
        }
```

Рисунок 1.12. Функція обчислення руху гравця

У методі Update() зчитуються усі дії гравця через систему введення.

```
private void Update()
        {
            CheckGroundStatus();
            _horizontal = PlayerInputs.GetHorizontal();
            _vertical = PlayerInputs.GetVertical();
            _jump = PlayerInputs.Jump();
            _dash = PlayerInputs.Dash();

            //this invert controls
            if (_invertedControl)
            {
                _horizontal *= -1;
                _vertical *= -1;
                _jump = PlayerInputs.Dash();
                _dash = PlayerInputs.Jump();
            }

            if (_jump && !HoldingObject)
            {
                Jump(JumpHeight);
            }

            if (_dash && !HoldingObject)
            {
                Dash();
            }

            //dash cooldown
            if (DashCooldown > 0)
            {
                DashCooldown -= Time.fixedDeltaTime;
            }
            else
            {
                DashCooldown = 0;
            }
            //set running animation

            SetRunningAnimation((Math.Abs(_horizontal) > 0 || Math.Abs(_vertical) > 0));

            //platforms
            if (!CurrentActivePlatform || !CurrentActivePlatform.CompareTag("Platform")) return;
            if (CurrentActivePlatform)
            {
                var newGlobalPlatformPoint = CurrentActivePlatform.TransformPoint(_activeLocalPlatformPoint);
                _moveDirection = newGlobalPlatformPoint - _activeGlobalPlatformPoint;
                if (_moveDirection.magnitude > 0.01f)
                {
                    _controller.Move(_moveDirection);
                }

                if (!CurrentActivePlatform) return;

                // Support moving platform rotation
                var newGlobalPlatformRotation = CurrentActivePlatform.rotation * _activeLocalPlatformRotation;
                var rotationDiff = newGlobalPlatformRotation * Quaternion.Inverse(_activeGlobalPlatformRotation);
                // Prevent rotation of the local up vector
                rotationDiff = Quaternion.FromToRotation(rotationDiff * Vector3.up, Vector3.up) * rotationDiff;
                _characterTransform.rotation = rotationDiff * _characterTransform.rotation;
                _characterTransform.eulerAngles = new Vector3(0, _characterTransform.eulerAngles.y, 0);

                UpdateMovingPlatform();
            }
            else
            {
                if (!(_moveDirection.magnitude > 0.01f)) return;
                _moveDirection = Vector3.Lerp(_moveDirection, Vector3.zero, Time.deltaTime);
                _controller.Move(_moveDirection);
            }
        }
```

Рисунок 1.13. Метод зчитування дій граця

Метод Jump(float jumpHeight) виконується, коли гравець натискає кнопку стрибка. Якщо персонаж на землі - виконується звичайний стрибок. Якщо в повітрі - дозволяється один додатковий стрибок (подвійні стибки), якщо активовано CanDoubleJump.

```
public void Jump(float jumpHeight)
        {
            if (!CanJump || !CanControl)
            {
                return;
            }

            if (_isGrounded)
            {
                _hitNormal = Vector3.zero;
                SetJumpAnimation();
                _doubleJump = true;
                _velocity.y = 0;
                _velocity.y += Mathf.Sqrt(jumpHeight * -2f * Gravity);
                if (JumpEffect)
                {
                    Instantiate(JumpEffect, LowZonePosition.position, LowZonePosition.rotation);
                }
            }
            else if (CanDoubleJump && _doubleJump)
            {
                _doubleJump = false;
                _velocity.y = 0;
                _velocity.y += Mathf.Sqrt(jumpHeight * -2f * Gravity);
                if (JumpEffect)
                {
                    Instantiate(JumpEffect, LowZonePosition.position, LowZonePosition.rotation);
                }
            }
        }
```

Рисунок 1.14. Метод стрибків та подвійних стрибків

Це забезпечує фізично коректну висоту стрибка, виходячи з сили гравітації.

Метод Dash викликається при натисканні відповідної кнопки, якщо cooldown завершився Рисунок 1.15.

Ривок додає імпульс у напрямку вперед, використовуючи складну формулу, яка враховує силу опору (DragForce).

Крім того, запускається корутина Dashing() і виводиться ефект (DashEffect).

Якщо гравець стоїть на динамічній платформі (CurrentActivePlatform), координати персонажа оновлюються відносно платформи. Обчислюється зміщення і поворот, щоб гравець «їхав» разом із платформою:

Відбувається компенсація руху та повороту, щоб персонаж не «з'їжджав» з об'єкта під ногами відповідно до рисунку 1.16.

```
public void Dash()
        {
            if (!CanDash || DashCooldown > 0)
            {
                return;
            }

            DashCooldown = _dashCooldown;

            if (DashEffect)
            {
                Instantiate(DashEffect, transform.position, _characterTransform.rotation);
            }

            SetDashAnimation();
            StartCoroutine(Dashing(DashForce / 10));
            _velocity += Vector3.Scale(transform.forward,
                DashForce * new Vector3((Mathf.Log(1f / (Time.deltaTime * DragForce.x + 1)) / -Time.deltaTime),
                    0, (Mathf.Log(1f / (Time.deltaTime * DragForce.z + 1)) / -Time.deltaTime)));
        }
```

Рисунок 1.16. Метод ривка гравця.

```
_moveDirection = newGlobalPlatformPoint - _activeGlobalPlatformPoint;
                if (_moveDirection.magnitude > 0.01f)
                {
                    _controller.Move(_moveDirection);
                }
```

Рисунок 1.15. Метод переміщення по платформам

Якщо поверхня під персонажем нахилена більше, ніж дозволяє SlopeLimit, персонаж починає ковзати по ній. Це реалізується додаванням сили ковзання вбік відповідно до рисунку 1.17.

```
if (!_isCorrectGrounded && _isGrounded)
            {
                _move.x += (1f - _hitNormal.y) * _hitNormal.x * (1f - SlideFriction);
                _move.z += (1f - _hitNormal.y) * _hitNormal.z * (1f - SlideFriction);
            }
```

Рисунок 1.17. Взаємодія з поверхнею

Гравітація реалізована через зміну вертикальної складової вектора швидкості відповідно до рисунку 1.18.

```
if (_velocity.y >= -MaxDownYVelocity)
            {
                _velocity.y += Gravity * Time.deltaTime;
            }
```

Рисунок 1.18. Гравітація гравця

Цей скрипт є ядром контролера персонажа з багатьма можливостями для платформеру: керування, фізика, взаємодія з оточенням, ефекти, стани. Його побудовано досить масштабно, з урахуванням розширюваності для подальшого розвитку гри - можна додавати нові ефекти, змінювати механіки або розширювати умови руху персонажа.

1.5.4 Реалізація скрипту TopDownCamera

Скрипт TopDownCamera реалізує логіку керування камерою у вигляді зверху (top-down), яка слідує за цільовим об'єктом із певним зсувом і має можливість обертання навколо нього. Цей підхід дає змогу створити систему візуального супроводу гравця, яка не є повністю статичною, а реагує на дії користувача, зберігаючи при цьому чітку прив’язку до об’єкта на сцені. У контексті ігор з видом зверху або ізометричних сцен такий тип камери є одним із найчастіше застосовуваних, оскільки забезпечує стабільний огляд і водночас дозволяє гравцю впливати на ракурс.

На початку роботи камери в методі Start відбувається ініціалізація зміщення (_offset), що є векторною різницею між поточним положенням камери і позицією цільового об’єкта (Target). Це зміщення зберігається як опорне, щоб згодом, під час руху гравця, камера могла слідувати за ним, зберігаючи постійний просторовий інтервал. Таким чином, відстежується лише зміна позиції Target, а не абсолютна позиція в просторі, що дозволяє зменшити візуальні скачки і зробити слідування плавним.

У методі Update відбувається зчитування вводу з клавіатури. На основі натискань клавіш Q і E встановлюються відповідні булеві прапорці, які

сигналізують про бажання гравця обернути камеру ліворуч або праворуч. Ці значення не використовуються одразу, а лише зберігаються до наступного фізичного оновлення сцени.

```
using UnityEngine;
namespace PlatformCharacterController
{
    public class TopDownCamera : MonoBehaviour
    {
        public Transform Target;

        public float SeeForward;

        public float RotationSpeed = 3;

        // The speed with which the camera will be following.
        public float Smoothing = 5f;

        // The initial offset from the target.
        private Vector3 _offset;

        private bool _rotate;
        private bool _rotateToLeft;
        private bool _rotateToRight;

        private void Start()
        {
            _offset = transform.position - Target.position;
        }

        private void Update()
        {
            if (_mobile) return;
            _rotateToLeft = Input.GetKey(KeyCode.Q);
            _rotateToRight = Input.GetKey(KeyCode.E);
        }
        private void FixedUpdate()
        {
            if (_rotateToLeft)
            {
                TurnCameraToLeft();
            }

            if (_rotateToRight)
            {
                TurnCameraToRight();
            }

            var targetCamPos = Target.position + _offset + Target.forward * SeeForward;
            transform.position = Vector3.Lerp(transform.position, targetCamPos, Smoothing * Time.deltaTime);
        }
        public void TurnCameraToLeft()
        {
            transform.Rotate(Vector3.up * -RotationSpeed, Space.World);
        }
        public void TurnCameraToRight()
        {
            transform.Rotate(Vector3.up * RotationSpeed, Space.World);
        }
    }
}
```

Рисунок 1.19. Скрипт роботи камери

У FixedUpdate, який виконується синхронно з фізичним циклом Unity, відбувається перевірка встановлених раніше прапорців _rotateToLeft і _rotateToRight. Якщо один із них активний, викликається відповідний метод обертання: TurnCameraToLeft() або TurnCameraToRight(). У межах цих методів використовується функція transform.Rotate, яка обертає камеру навколо глобальної осі Y (Vector3.up) на кут, визначений параметром RotationSpeed. Обертання відбувається у глобальному просторі (Space.World), що дозволяє зберігати стабільну орієнтацію незалежно від локального положення камери.

Після цього обчислюється нова цільова позиція камери targetCamPos. Вона визначається як поточна позиція Target плюс раніше збережене зміщення _offset, доповнене напрямком Target.forward, помноженим на коефіцієнт SeeForward. Це дозволяє камері зміщуватись вперед у напрямку руху гравця, що часто необхідно для поліпшення огляду зони перед персонажем. Зміна позиції камери здійснюється за допомогою Vector3.Lerp, що забезпечує поступовий перехід між поточною і цільовою позицією із заданою швидкістю згладжування, визначеною параметром Smoothing.

1.6 Розробка додаткових механік та левелдизайну у 3D-просторі

У межах цієї частини проекту планується реалізувати набір систем, які поглиблюють ігровий досвід і дозволяють гравцеві активно взаємодіяти з навколишнім світом. Визначальну роль у цьому відіграють рухомі платформи, об’єкти з особливими властивостями, телепорти, змінні фізичні зони, а також інші елементи, що створюють виклики та урізноманітнюють ігровий процес.

Основна увага приділяється не тільки технічній реалізації цих механік, а й тому, як саме вони інтегруються в загальний ландшафт рівня. Просторова побудова середовища виконується з урахуванням можливостей персонажа, фізичних характеристик об'єктів і сценаріїв, які виникають у результаті взаємодії між ними. Наприклад, рухомі платформи дозволяють долати великі вертикальні або горизонтальні відстані, змінюючи геометрію доступних маршрутів. У свою

чергу, телепорти забезпечують швидке переміщення між зонами, відкриваючи нові ділянки мапи або створюючи ефекти петлі, що вимагає адаптивного мислення з боку гравця.

Скрипти для активних предметів додають можливість взаємодії зі світом. У процесі розробки таких механік важливо забезпечити зрозумілу логіку їхньої роботи та візуальну підказку, що дозволяє гравцю інтуїтивно здогадуватися про можливість взаємодії. Усі подібні елементи створюються з урахуванням рівня складності, поступовості ознайомлення та естетичного поєднання з основною темою гри.

Левелдизайн у тривимірному просторі орієнтується на вертикальну багаторівневість, змінність перспектив і глибину кадру. Це дозволяє створити відчуття масштабу, відкритості, але водночас і цілеспрямованості маршруту. Простір формується так, щоб гравець завжди мав кілька варіантів дій, але водночас чітке розуміння, куди варто рухатися далі. Створення кожної ділянки рівня супроводжується тестуванням на предмет геймплейної збалансованості: перевіряється, чи не виникає ситуацій, коли гравець опиняється у безвиході, або коли певна механіка стає занадто легкою для зловживання.

1.6.1 Реалізація модулю появи функціональних предметів

Скрипт було реалізовано як компонент, який прикріплюється до об’єкта на сцені, що виконує роль точки спавну. Було передбачено можливість задавати префаб предмета у відповідному полі Item у редакторі, що значно полегшує подальше конфігурування без потреби втручатися в код. Щоб уникнути одночасного існування кількох екземплярів одного предмета, було додано логіку, яка перевіряє, чи вже існує згенерований об’єкт - якщо так, новий предмет не створюється, доки попередній не буде знищений.

В основі реалізації лежить відлік часу через змінну _spawnTime, яка поступово зменшується у методі FixedUpdate() - він був обраний з урахуванням можливого подальшого розширення логіки з використанням фізичних компонентів, хоча на поточному етапі код не залежить від фізичного рушія. Щойно таймер досягає нуля, викликається метод SpawnTheItem(), у якому

безпосередньо створюється новий об’єкт. У разі відсутності вказаного префабу генерується попередження у консолі, що допомагає уникати помилок під час налагодження.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpawnItem : MonoBehaviour
{
    public GameObject Item;

    private GameObject _currentItem;

    private float _spawnTime = 4;

    private void FixedUpdate()
    {
        if (!_currentItem)
        {
            _spawnTime -= Time.deltaTime;
            if (_spawnTime <= 0)
            {
                _spawnTime = 4;
                SpawnTheItem();
            }
        }
    }

    public void SpawnTheItem()
    {
        if (_currentItem)
        {
            Destroy(_currentItem);
        }

        if (Item)
        {
            _currentItem = Instantiate(Item, transform.position, transform.rotation);
        }
        else
        {
            Debug.LogWarning("Please add a item to spawn.");
        }
    }
}
```

Рисунок 1.20. Скрипт появу предметів

Увагу було приділено тому, щоби згенерований об’єкт поводився природно у фізичному просторі гри. Саме тому передбачено, що сам префаб, який буде з’являтись, повинен містити компонент Rigidbody, що забезпечує інтеграцію з фізичним рушієм Unity. Також були протестовані сценарії, у яких предмет міг містити додаткові колайдери або скрипти для обробки взаємодії з гравцем. Це дозволило створити гнучку систему, яку можна масштабувати відповідно до потреб рівня чи сценарію.

У процесі інтеграції цієї системи з левелдизайном було передбачено можливість використання кількох таких точок на різних ділянках рівня. Таким чином вдалося створити відчуття живого середовища, де об’єкти з’являються після певної паузи, стимулюючи гравця взаємодіяти з простором повторно, а не один раз. Такий підхід виявився особливо ефективним у контексті створення квестових чи бонусних механік.

1.6.2 Реалізація модулю змінення характеристик персонажу

У системи взаємодії з активними предметами було реалізовано зміну ключових характеристик персонажа, що безпосередньо впливають на його здатність до навігації в просторі рівня. Зокрема, особлива увага була приділена змінним, пов'язаним зі швидкістю пересування та стрибками, які є центральними для забезпечення динамічного геймплею у платформері з вертикальним і горизонтальним пересуванням.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
namespace PlatformCharacterController
{
    public class Sprint : MonoBehaviour
    {
        [Tooltip("The speed value to add at the player speed.")]
        public float SpeedPlus = 5;
        [Tooltip("This is the time that the speed plus is active in the player.")]
        public float SprintTime = 4;
        public GameObject Effect;
        private bool _active = true;
        private void SprintPlayer(MovementCharacterController player)
        {
            _active = false;
            if (Effect)
            {
                Instantiate(Effect, transform.position, transform.rotation);
            }

            player.ChangeSpeedInTime(SpeedPlus, SprintTime);

            Destroy(gameObject);
        }
        private void OnTriggerEnter(Collider other)
        {
            if (other.CompareTag("Player") && _active)
            {
                SprintPlayer(other.GetComponent<MovementCharacterController>());
            }
        }
    }
}
```

Рисунок 1.21. Скрипт зміни швидкості

При активації певного предмета, персонаж отримує тимчасове посилення базової швидкості, що дозволяє йому швидше переміщатися по рівню, долати більші дистанції за менший проміжок часу та впевненіше реагувати на виклики середовища, такі як нестабільні платформи чи небезпечні перешкоди. Зміна швидкості реалізується програмно через доступ до відповідного параметра в контролері персонажа - зазвичай це публічна змінна типу float, яка визначає інтенсивність руху.

Окрім швидкості, у проекті була передбачена можливість модифікації поведінки стрибка, зокрема - додавання функціональності подвійного стрибка. Така механіка дозволяє гравцеві двічі натиснути кнопку стрибка, щоб виконати додаткове підняття у повітрі, навіть після того, як персонаж уже відірвався від землі. Цей ефект особливо актуальний при взаємодії з предметами, які надають тимчасовий доступ до важкодоступних ділянок рівня або створені як частина навігаційних викликів у вертикальному дизайні локації.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace PlatformCharacterController
{
    public class HandleDoubleJumpSkill : MonoBehaviour
    {
        [Header("Activate doblejump")] [Tooltip("Enable player doblejump if true, disable if false")]
        public bool ActivateDoubleJump;

        public bool DestroyIfActive;

        public GameObject Effect;

        private void OnTriggerEnter(Collider other)
        {
            if (!other.CompareTag("Player")) return;

            other.GetComponent<MovementCharacterController>().ActivateDeactivateDoubleJump(ActivateDoubleJump);

            if (Effect)
            {
                Instantiate(Effect, transform.position, transform.rotation);
            }

            if (DestroyIfActive)
            {
                Destroy(gameObject);
            }
        }
    }
}
```

Рисунок 1.22. Скрипт додавання подвійного стрибку

1.6.3 Реалізація модулю платформ та сфер

Було запроваджено систему просторового переміщення платформ і сфер, яка імітує ефект левітації або періодичного руху по заданій траєкторії. Цей підхід дозволяє оживити структуру рівня, зробити середовище більш реактивним та візуально привабливим, а також ускладнити платформінг за рахунок змінних положень опорних поверхонь.

Основу такої поведінки складає скрипт Oscillator, прикріплений до відповідного ігрового об'єкта. Його призначення - здійснювати осциляційний рух навколо початкової позиції. Механізм реалізується через перерахунок позиції об'єкта кожного кадру в методі FixedUpdate, що забезпечує стабільну фізичну симуляцію незалежно від зміни частоти кадрів. Поточна позиція визначається як сума початкової координати (m_StartPosition) та векторного зміщення, що коливається у часі за допомогою функції синуса.

```
using UnityEngine;

// Makes a transform oscillate relative to its start position
public class Oscillator : MonoBehaviour
{
    public float m_Amplitude = 1.0f;
    public float m_Period = 1.0f;
    public Vector3 m_Direction = Vector3.up;
    Vector3 m_StartPosition;

    void Start()
    {
        m_StartPosition = transform.position;
    }

    void FixedUpdate()
    {
        var pos = m_StartPosition + m_Direction * m_Amplitude * Mathf.Sin(2.0f * Mathf.PI * Time.time / m_Period);
        transform.position = pos;
    }
}
```

Рисунок 1.23. Скрипт переміщення об’єктів

Параметри m_Amplitude та m_Period задають, відповідно, амплітуду коливання - тобто, відстань, на яку платформа або сфера відхиляється від початкової точки, - та період повного циклу руху. Вектор m_Direction визначає напрямок коливання у просторі. Зміна цього вектора дозволяє налаштовувати рух вгору-вниз, ліворуч-праворуч або вперед-назад, а також по довільній діагоналі.

Таке рішення не потребує використання фізичного тіла (Rigidbody), оскільки зміна положення здійснюється напряму через transform.position. Завдяки цьому забезпечується повний контроль над рухом, відсутність інерції й передбачуваність, що особливо важливо для платформ, на які гравець має стрибати або з якими взаємодіяти в певному ритмі.

Об'єкти виступають не лише декоративним елементом, а й інструментом ігрового випробування. Левітуючі платформи можуть використовуватись для створення рухомих цілей, перешкод або логічних задач на таймінг. Сфери ж, як правило, виконують роль бонусних об’єктів або носіїв здібностей, і їхнє плавне переміщення у просторі додає ефект присутності, змушує гравця звертати на них увагу.

Імітація руху, близька до левітації, з м'якими і передбачуваними коливаннями - один із найефективніших прийомів для створення живого, змінного середовища у платформерах та 3D-іграх з елементами головоломки. За потреби, поведінку можна масштабувати, комбінуючи декілька векторів або синхронізуючи періоди декількох об’єктів для створення більш складних анімацій або динамічних викликів.

1.6.4 Реалізація модулю телепорту

У реалізації механіки безперервності геймплею та запобігання ситуаціям, коли гравець опиняється поза межами ігрової карти або потрапляє в недоступні ділянки простору, було впроваджено спеціальний скрипт Teleport. Його головне завдання - забезпечити автоматичне повернення персонажа на безпечну початкову позицію у разі потрапляння в зону, яка позначає межу ігрового світу або його "фоновий простір", зазвичай невидимий та не призначений для взаємодії.

Код працює за рахунок виявлення колізії з об'єктом, на якому він прикріплений. Такий об'єкт виконує роль тригер-зони, розташованої, як правило, під рівнем карти або по її краях. Як тільки гравець (об’єкт із тегом "Player") торкається цієї зони, спрацьовує метод OnTriggerEnter, який активує два паралельні процеси: тимчасове відключення контролю над персонажем і саму процедуру телепортації. Це реалізовано через використання корутин -

асинхронних функцій, які дозволяють відтермінувати певні дії на вказаний час без блокування основного потоку гри.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace PlatformCharacterController
{
    public class Teleport : MonoBehaviour
    {
        [Tooltip("Time to start teleporting the player.")]
        public float StartTeleport;

        [Tooltip("The player wait this time to can control the player again.")]
        public float TimeToControlPlayer;

        [Tooltip("Effect to start teleport.")] public GameObject TeleportEffect;

        public Transform TeleportPosition;

        private IEnumerator TeleportPlayer(Transform player)
        {
            yield return new WaitForSeconds(StartTeleport);
            if (TeleportEffect)
            {
                Instantiate(TeleportEffect, player.position, player.rotation);
            }

            player.position = TeleportPosition.position;
        }

        private void OnTriggerEnter(Collider other)
        {
            if (other.CompareTag("Player"))
            {
                StartCoroutine(other.GetComponent<MovementCharacterController>()
                    .DeactivatePlayerControlByTime(TimeToControlPlayer));
                StartCoroutine(TeleportPlayer(other.transform));
            }
        }
    }
}
```

Рисунок 1.24. Скрипт телепорту

Ключовим елементом є посилання на Transform TeleportPosition, яке задає координати місця, куди слід перемістити персонажа. Це зазвичай одна з контрольних точок - стартова позиція або інша зона, визначена дизайнером як безпечна для відновлення гравця. Перед телепортацією, з урахуванням заданої затримки (StartTeleport), може бути активований візуальний ефект (TeleportEffect), що сигналізує про перенесення й додає естетичної завершеності події.

Важливо, що на час телепортації відбувається деактивація можливості управління персонажем - це запобігає потенційним помилкам при зміні координат або повторному потраплянню до тригер-зони. Для цього використовується метод DeactivatePlayerControlByTime, що входить до системи контролю руху MovementCharacterController і тимчасово обмежує дії гравця.

1.6.5 Реалізація модулю персонажу гри

В розробці ігрового проекту для візуального втілення персонажа використовується готовий пакет із Unity Asset Store - Free Low Poly Human RPG Character(https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/free-low-poly-human-rpg-character-219979). Зазначений ассет містить повноцінну тривимірну модель гуманоїдного персонажа у стилі low poly, що ідеально відповідає загальній естетиці проекту та не перевантажує графічну сцену, що особливо важливо для стабільності продуктивності під час запуску на різних платформах.

Рисунок 1.25. Модель персонажу із Unity Asset Store

Модель підтримує стандартну гуманоїдну анімаційну структуру Unity (Mecanim), що дає змогу безпосередньо застосовувати до неї як кастомні, так і бібліотечні анімації, включно з ходьбою, бігом, стрибками, приземленням та

взаємодією з навколишнім середовищем. Візуальна простота та оптимізоване використання полігонів робить даного персонажа придатним як для розробки прототипу, так і для повноцінного ігрового процесу.

Персонаж, отриманий з цього ассету, був інтегрований у систему управління, що реалізована через спеціально створений скрипт MovementCharacterController. Це забезпечує повний контроль за його переміщенням у 3D-просторі, а також дозволяє використовувати додаткові можливості, зокрема активацію подвійного стрибка, телепортацію, взаємодію з предметами тощо.

Завдяки гнучкості структури ассету, також була проведена адаптація текстур і налаштування матеріалів під освітлення сцени. Це забезпечило візуальну гармонію між персонажем і середовищем, у якому він перебуває. Таким чином, Free Low Poly Human RPG Character виступає не лише як візуальна складова, але і як повноцінний елемент системи, що забезпечує ігрову функціональність та є важливою частиною загальної архітектури проекту.

1.6.6 Реалізація модулю матеріалів та текстур

У процесі візуального оформлення ігрового середовища нашого проекту було використано спеціалізований графічний ресурс із Unity Asset Store - KCISA Korean Traditional Smart Materials (https://assetstore.unity.com/packages/2d/textures-materials/tiles/kcisa-korean-traditional-smart-materials-vol-6-303901). Цей набір матеріалів та текстур відзначається високою якістю відображення традиційних архітектурних елементів корейської естетики, що робить його придатним для створення унікальної атмосфери в ігровому світі, з акцентом на фактурність поверхонь і декоративну стилізацію.

У межах реалізації нашого платформера ці текстури були інтегровані в оформлення ключових елементів рівня, включно з:

–ігровими платформами, які завдяки матеріалам із набору отримали візуальну глибину й деталізацію, що нагадує плитку або старовинні кам’яні кладки;

–колонами, які були стилізовані під архітектурні опори, із застосуванням

розширених нормальних карт і текстур високої роздільності для досягнення реалістичності візуального зчитування об'єму та поверхні;

–трикутними платформами, що слугують навігаційними або декоративними елементами, стилістично інтегрованими в загальну концепцію середовища завдяки текстурам з азійськими архітектурними візерунками;

– стіновими обмеженнями рівня, які були оформлені з використанням модифікованих матеріалів із цього пакета, щоб надати гравцеві чітке візуальне розуміння меж світу та водночас не порушити загальну естетику сцени.

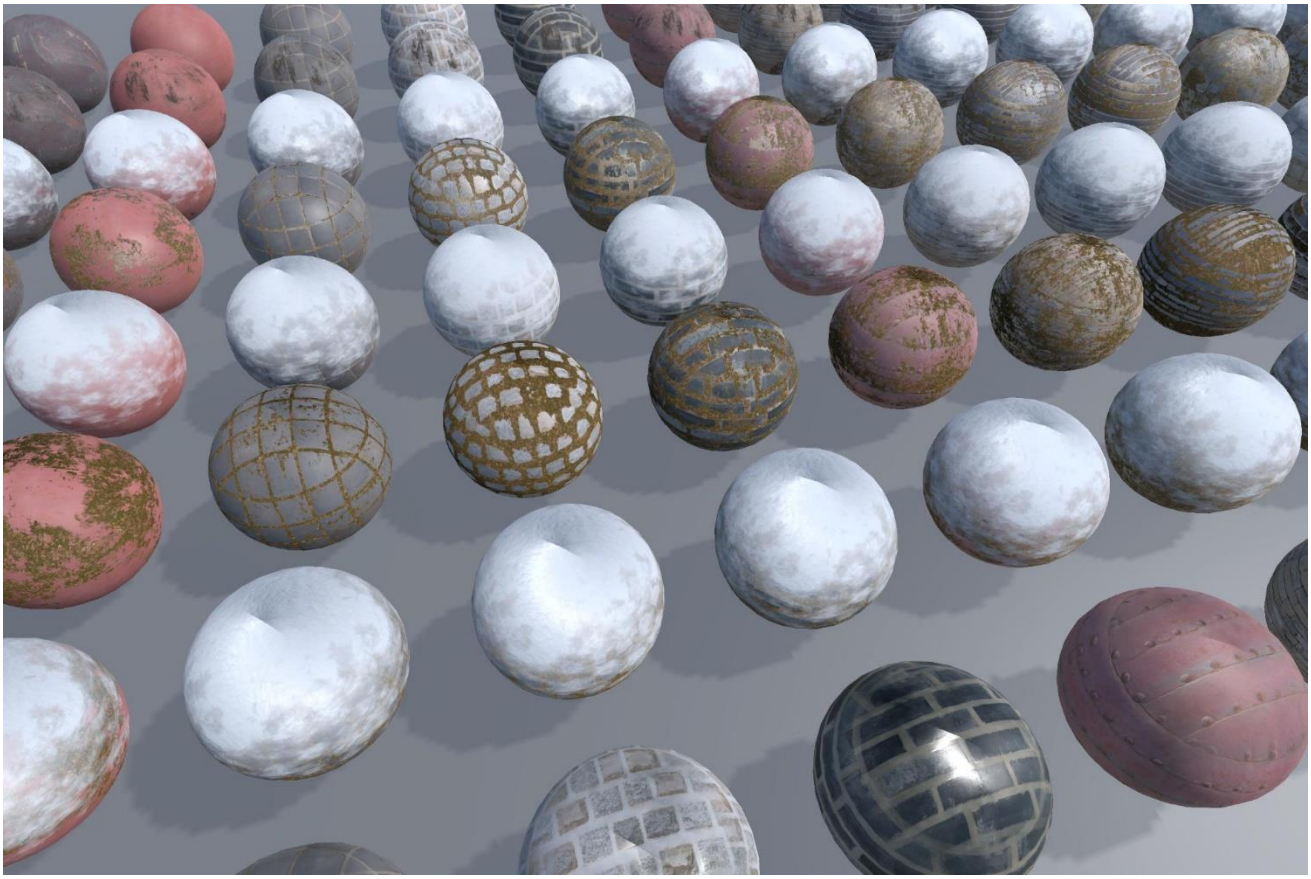


Рисунок 1.26. Текстури для проекту із Unity Asset Store

1.6.7 Реалізація модулю об’єктів гри

Для візуального оформлення ігрового світу було створено набір тривимірних об’єктів, які формують як функціональне середовище для переміщення персонажа, так і слугують важливими орієнтирами в просторі рівня. Кожен з об’єктів несе як естетичне, так і геймплейне навантаження, що підсилює атмосферу та забезпечує зрозумілість ігрової логіки.

Платформи є основними елементами ігрового рівня. Вони слугують опорами, по яких гравець пересувається, перестрибує та взаємодіє з оточенням. Більшість платформ реалізовано у вигляді прямокутних об’ємних форм із чітко окресленими краями, що добре зчитуються в перспективі. Деякі з них статичні, інші - анімовані за допомогою спеціального скрипта (Oscillator.cs), що забезпечує плавну осциляцію в просторі - вертикальну або горизонтальну. Це створює відчуття левітації або зависання в повітрі, що ускладнює та урізноманітнює пересування.

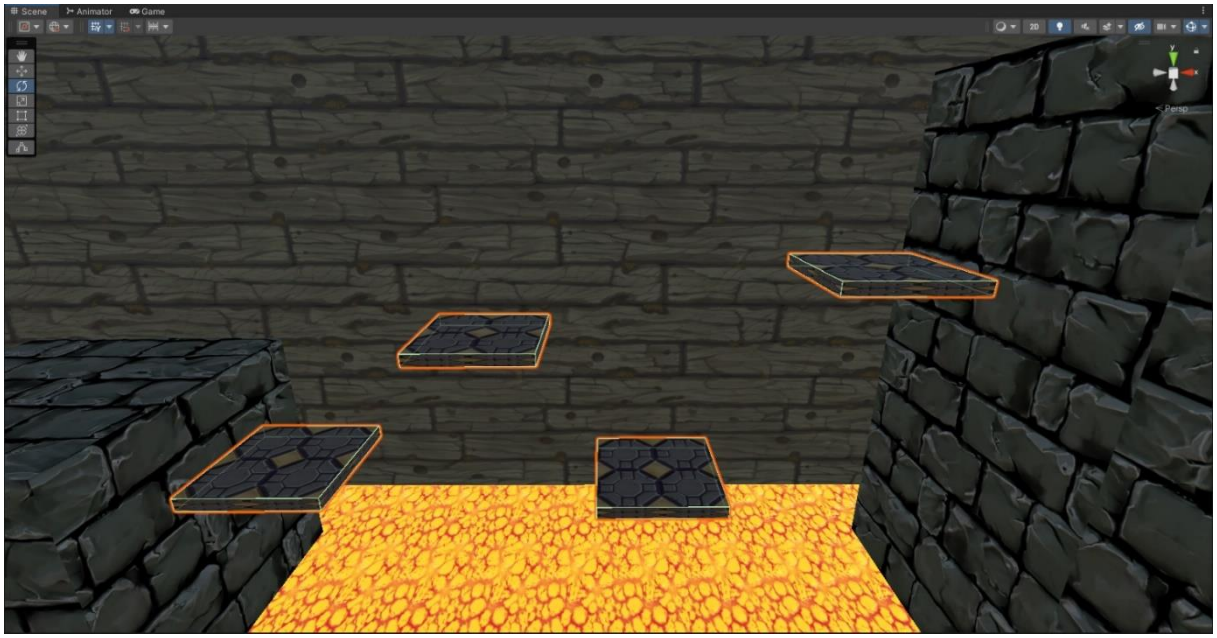


Рисунок 1.27. Моделі платформ у проекті

Сфери у грі використовуються як елементи, що виконують як декоративну, так і функціональну роль. Візуально вони мають гладку, симетричну форму, що контрастує з жорсткою геометрією платформ. Часто сфери або обертаються, або плавно переміщуються у просторі за допомогою того ж механізму осциляції, підсилюючи враження динамічного середовища. У деяких випадках сфери можуть виступати як активні об’єкти, що реагують на взаємодію гравця, як-от швидкість переміщення та можливість виконання подвійного стрибка відповідно до рисунку 1.28.

Углові платформи представлені у формі об’ємних трикутників або трапецій, що розміщуються під різними кутами в просторі. Їхня функція - створити змінну геометрію рівня: похилі траєкторії, ковзання або використання

як трамплінів. Візуально ці платформи відрізняються різкістю кутів і створюють ефект зламаного середовища, де традиційне горизонтальне пересування піддається виклику відповідно до рисунку 1.29.

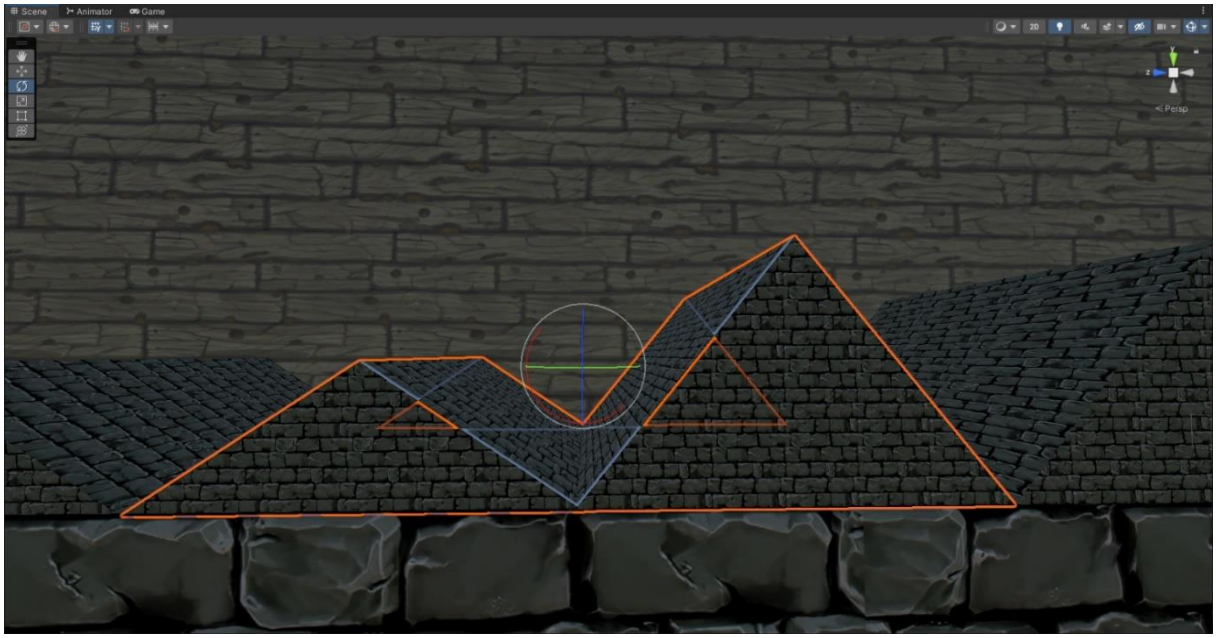


Рисунок 1.29. Модель трикутної перешкоди

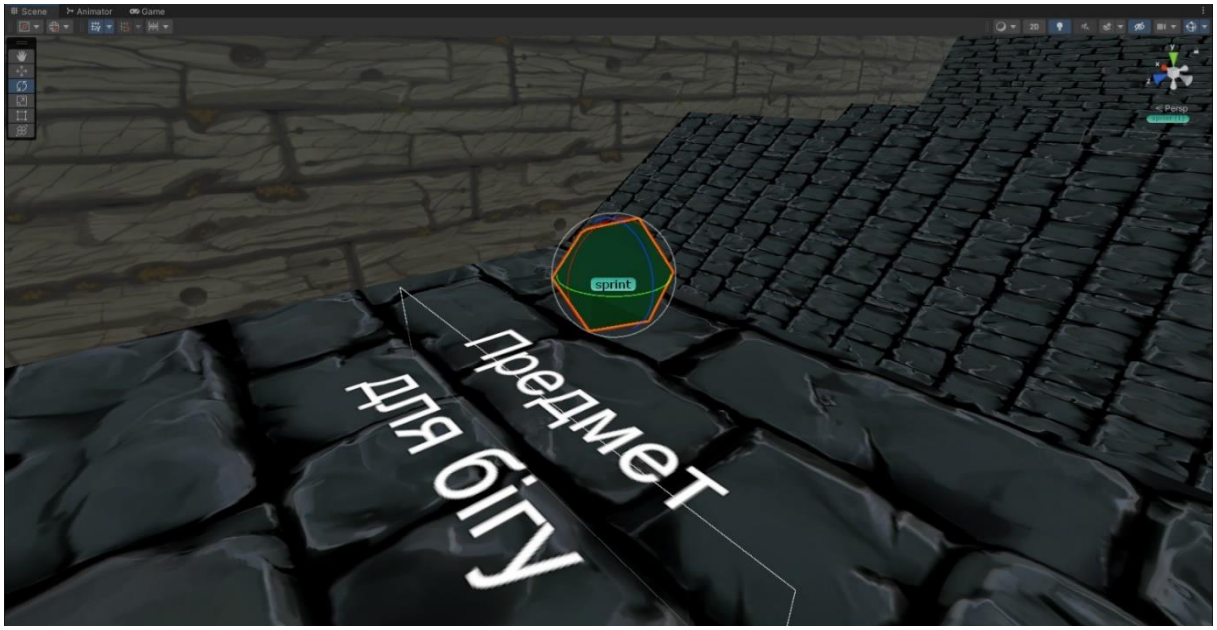


Рисунок 1.28. Модель інтерактивної сфери

Колони є вертикальними конструкціями, що можуть виконувати одразу кілька ролей: бути частиною архітектури, слугувати опорами для інших об’єктів або виступати як ізольовані майданчики для стрибків. Вони мають витягнуту форму, часто великої висоти, і використовуються як візуальні маркери вертикального прогресу або як перешкоди, що створюють складні маршрути для гравця. Колони додають грі глибини та вертикального виміру.

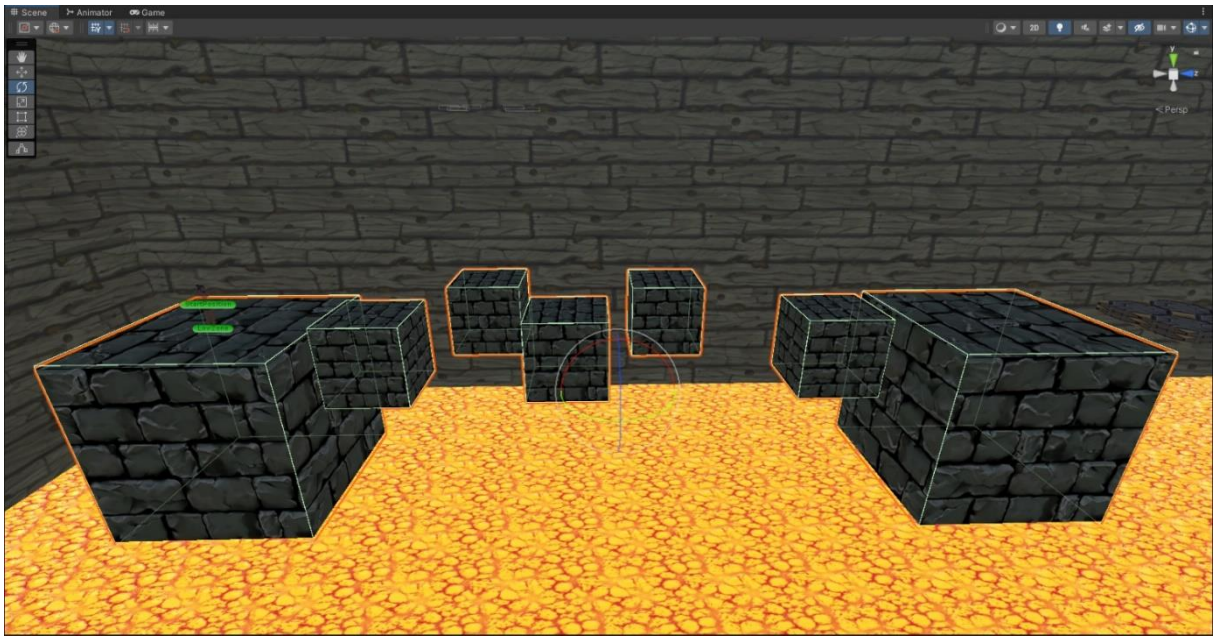


Рисунок 1.30. Модель колон для сцени

У візуальній структурі рівня окреме місце посідають лава та стіни, що виступають як обмежувальні й каральні елементи ігрового простору. Їхнє призначення - не лише формувати межі, але й активно впливати на гравця, змушуючи бути уважним до свого положення в просторі.

Лава розміщується в нижній частині сцени, на дні рівня, що візуально формує ігрову загрозу. Вона подається як яскравий, пульсуючий шар, що імітує розпечену магму або енергію, з ефектами світіння й анімацією. Потрапляння гравця в зону лави не призводить до звичної анімації смерті, а активує систему телепортації: персонаж миттєво повертається на визначену стартову позицію.

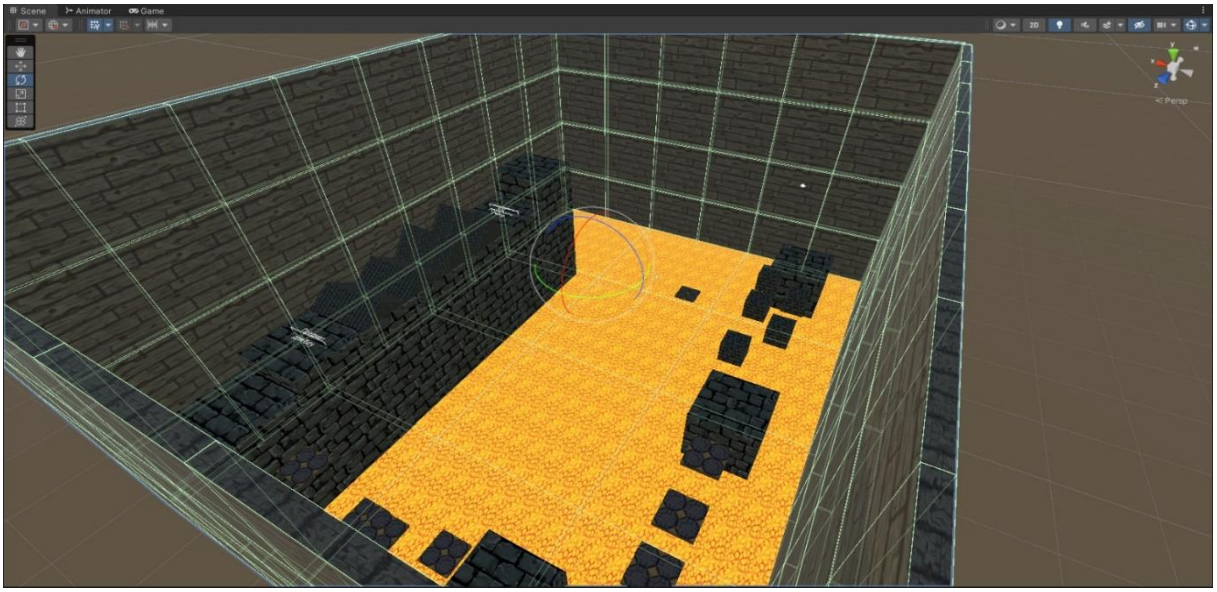


Рисунок 1.31. Зовнішній вигляд моделі стін та лави

Разом ці об’єкти - платформи, сфери, колони, похилі елементи, стіни та лава - формують комплексне, динамічне середовище, яке балансує між викликом, візуальною привабливістю та функціональністю. Взаємодія з кожним із них - це частина загальної системи, яка підпорядкована логіці переміщення, розвитку рівня та підтримання темпу гри. Такий підхід дозволяє гравцю залишатися в постійній динаміці, не втрачаючи орієнтації, і при цьому чітко усвідомлювати ризики, пов’язані з невдалими діями.

1.7 Імплементація системи керування персонажем та взаємодії з ігровим середовищем

У цьому розділі розглядається загальний процес додавання логіки до ігрових об'єктів за допомогою скриптів, що керують їхньою поведінкою. Йдеться про імплементацію ключових механік взаємодії персонажа з навколишнім середовищем, а також налаштування властивостей самих моделей для забезпечення коректної фізичної реакції та анімаційної відповіді. Скрипти додаються як до головного персонажа, так і до допоміжних об’єктів середовища, зокрема платформ, зон взаємодії та декоративних або геймплейних елементів. Це дозволяє моделювати рух, стрибки, обмеження простору, реакцію на колізії, телепортацію, а також інші дії, що виникають під час проходження рівнів. Усе це інтегрується безпосередньо через інспектор в редакторі Unity, де кожна модель отримує відповідні компоненти, які забезпечують потрібну функціональність у рамках загального геймплейного процесу.

У цьому етапі проекту до моделі гравця додається кілька ключових компонентів, які забезпечують його повноцінну інтеракцію з ігровим середовищем. Додається компонент Character Controller, що дозволяє обробляти зіткнення з об’єктами рівня, переміщення по поверхнях різного нахилу та керування фізикою без використання фізичного тіла. Його параметри налаштували таким чином, щоби забезпечити коректну взаємодію з платформами та уникнути зависань у колізіях: обмеження кута нахилу встановили на рівні 45 градусів, мінімальну відстань руху задали як 0.001, а радіус капсули - 0.45.

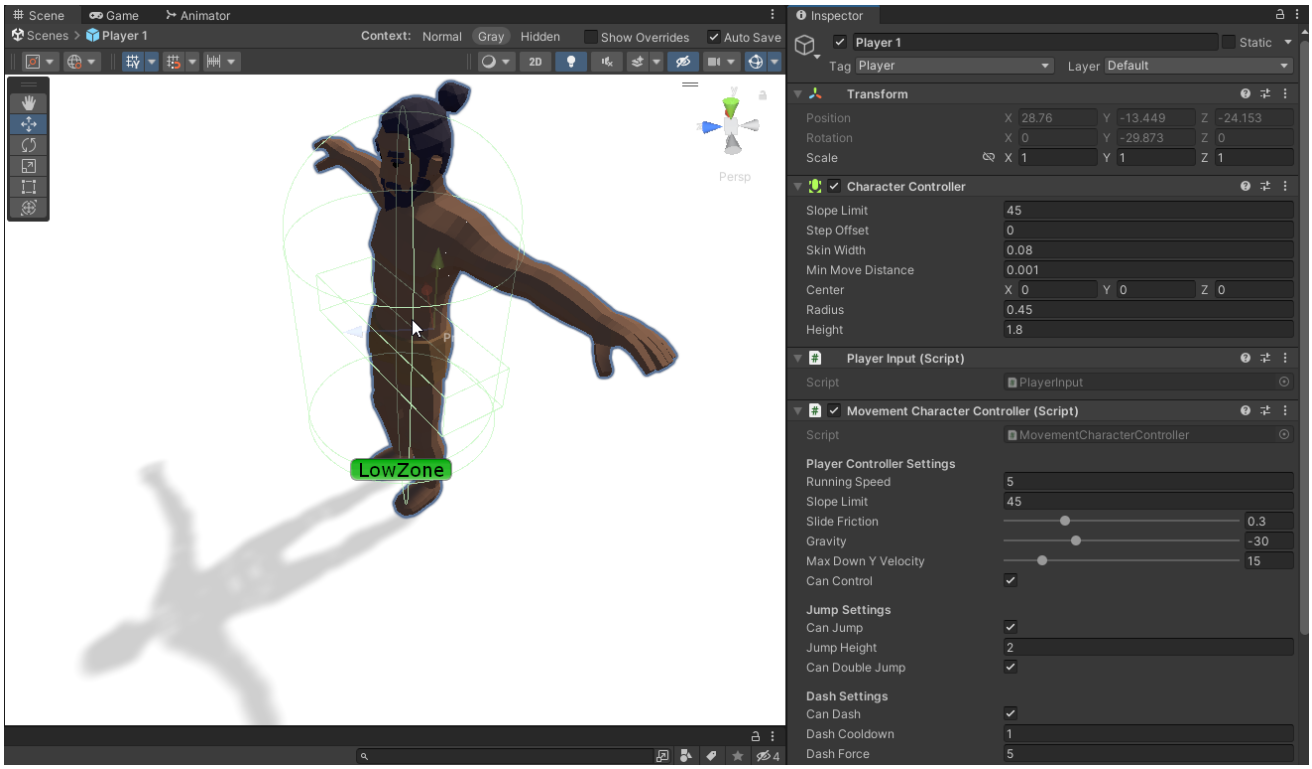


Рисунок 1.32. Налаштування об’єкта гравця на сцені

Було додано спеціальний скрипт, який реалізує систему керування гравцем. У налаштуваннях цього компонента визначено основні характеристики: швидкість бігу - 5, сила гравітації - –30, обмеження вертикальної швидкості - 15. Активація можливість стрибка, задавши висоту стрибка - 2, а також увімкнути функцію ривка (dash), встановивши силу ривка - 5 та затримку між ривками - 2.

Також до гравця додається компонент Player Input, який дозволяє приймати сигнали з клавіатури й передавати їх у систему контролю руху.

Було додано функціональну платформу, яка виконує роль основної опори для переміщення гравця в межах сцени. Об'єкт платформи має тег Platform, що дозволяє коректно ідентифікувати його серед інших об'єктів у сцені та забезпечити взаємодію зі скриптами персонажа. За геометричною основою платформи використано базову тривимірну фігуру типу Cube, яка відображається за допомогою Mesh Filter та Mesh Renderer.

Щоб забезпечити фізичну взаємодію платформи з гравцем, об’єкт містить компонент Box Collider, який відповідає за зіткнення. Колайдер не має активованого режиму Is Trigger, що означає - гравець буде фізично зіштовхуватись із платформою, а не проходити крізь неї. Також вказано розміри

та центр колайдера, що дозволяє точно підігнати його під геометрію об’єкта. Через Mesh Renderer платформа отримує візуальні властивості, а також активовано прийом тіней, що створює більш реалістичну освітленість сцени.

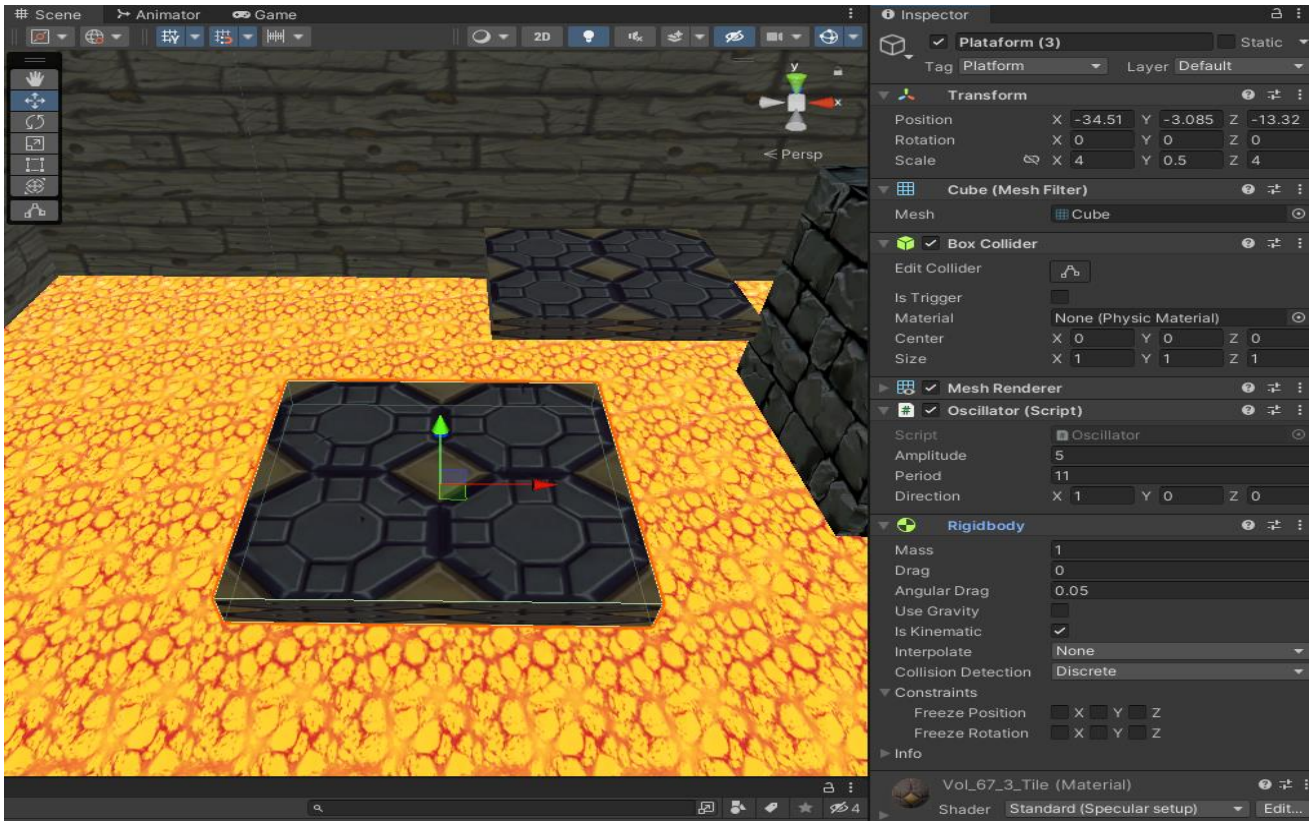


Рисунок 1.33. Налаштування об’єкта платформи на сцені

Особливістю цієї платформи є доданий скрипт типу Oscillator, що забезпечує її рух. Через параметри Amplitude та Period визначено амплітуду (5) та періодичність (10) руху, а через вектор напрямку (X = 1, Y = 0, Z = 0) встановлено горизонтальний зсув вздовж осі X.

Компонент Rigidbody, доданий до об’єкта, дозволяє використовувати фізику Unity. При цьому увімкнено режим Is Kinematic, що означає, що платформа не піддається впливу фізичних сил, однак може переміщуватись програмно. Усі ці параметри дозволяють створити інтерактивну платформу, яка органічно вписується в ігровий простір і забезпечує необхідний рівень виклику для гравця.

Для організації візуального сприйняття ігрової сцени було додано камеру з відповідними параметрами позиціонування та огляду. Основною камерою виступає об'єкт Main Camera, який відображає події у грі з перспективи третьої

особи або згори, залежно від налаштувань. Її положення у просторі визначається через компонент Transform, де зазначено координати позиції, обертання та масштаб, що дозволяє розмістити камеру над рівнем для повноцінного огляду ігрового середовища.

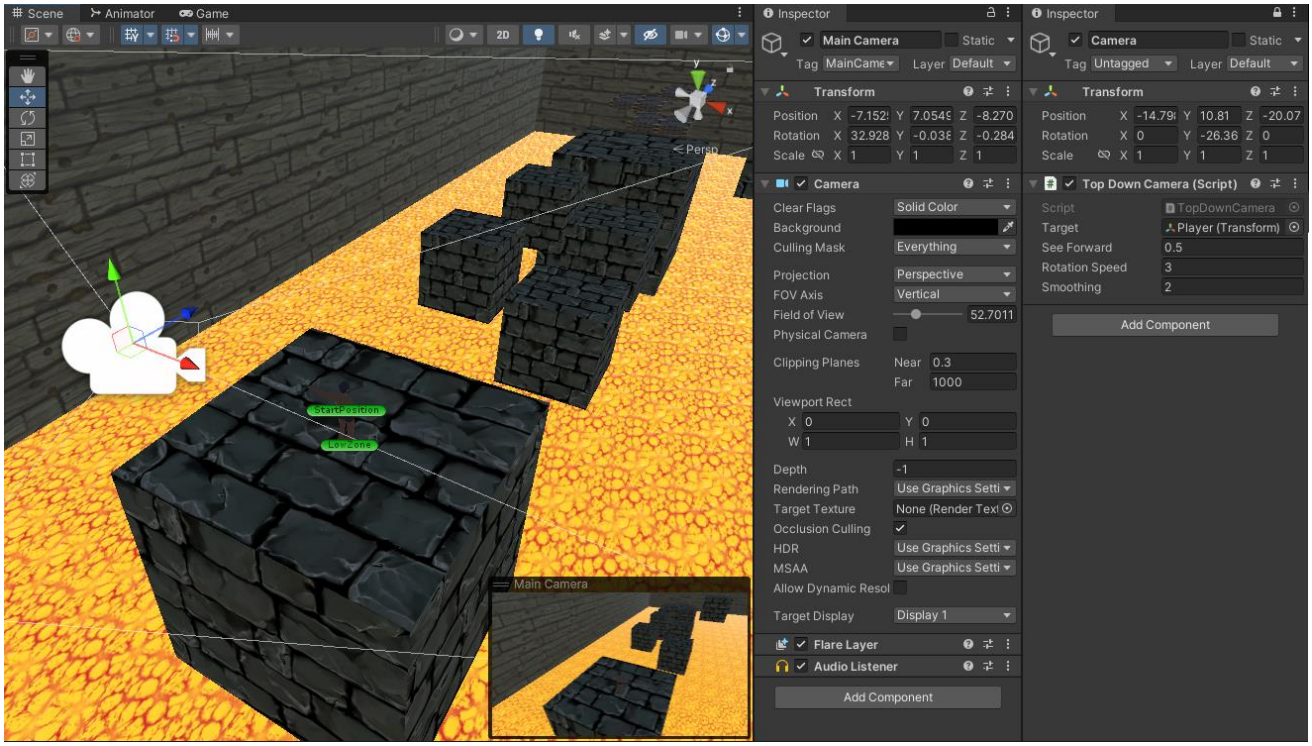


Рисунок 1.34. Налаштування об’єкта камери на сцені

У налаштуваннях самої камери (компонент Camera) встановлено режим Clear Flags як Solid Color, що визначає, яким чином очищується фон під час рендерингу кадру. Проекція встановлена у значення Perspective, що дозволяє надати глибинний вигляд сцені. Значення Field of View (приблизно 52.7) задає кут огляду, що впливає на те, скільки простору видно гравцю одночасно. Також налаштовано параметри Clipping Planes (Near = 0.3, Far = 1000), які визначають, на якій відстані об’єкти починають та припиняють бути видимими.

Щоб реалізувати плавне та динамічне стеження за гравцем, до об'єкта камери було додано скрипт типу TopDownCamera. Через параметр Target до цього скрипта прив’язується сам об’єкт гравця (Player), що дозволяє камері автоматично слідкувати за його переміщенням. Додаткові параметри (наприклад, Rotation Speed = 3, Smoothing = 0) відповідають за швидкість та плавність обертання та адаптації камери до рухів персонажа. Через параметр See Forward визначено

напрямок, у якому камера орієнтується відносно цільового об’єкта, забезпечуючи правильну точку огляду.

Об’єкт предмета було оформлено у вигляді окремого префабу (наприклад, SprintItem), що дозволяє багаторазово використовувати його у різних частинах сцени. Основу об’єкта становить стандартний геометричний об’єкт типу Sphere із відповідним компонентом Mesh Renderer, який забезпечує відображення форми та матеріалу в грі. Для забезпечення фізичної взаємодії додано компонент Sphere Collider з активованою опцією Is Trigger. Це дозволяє виявити факт зіткнення гравця з предметом, але без фізичного блокування - тобто, персонаж може пройти крізь об'єкт, активувавши при цьому подію підбирання.

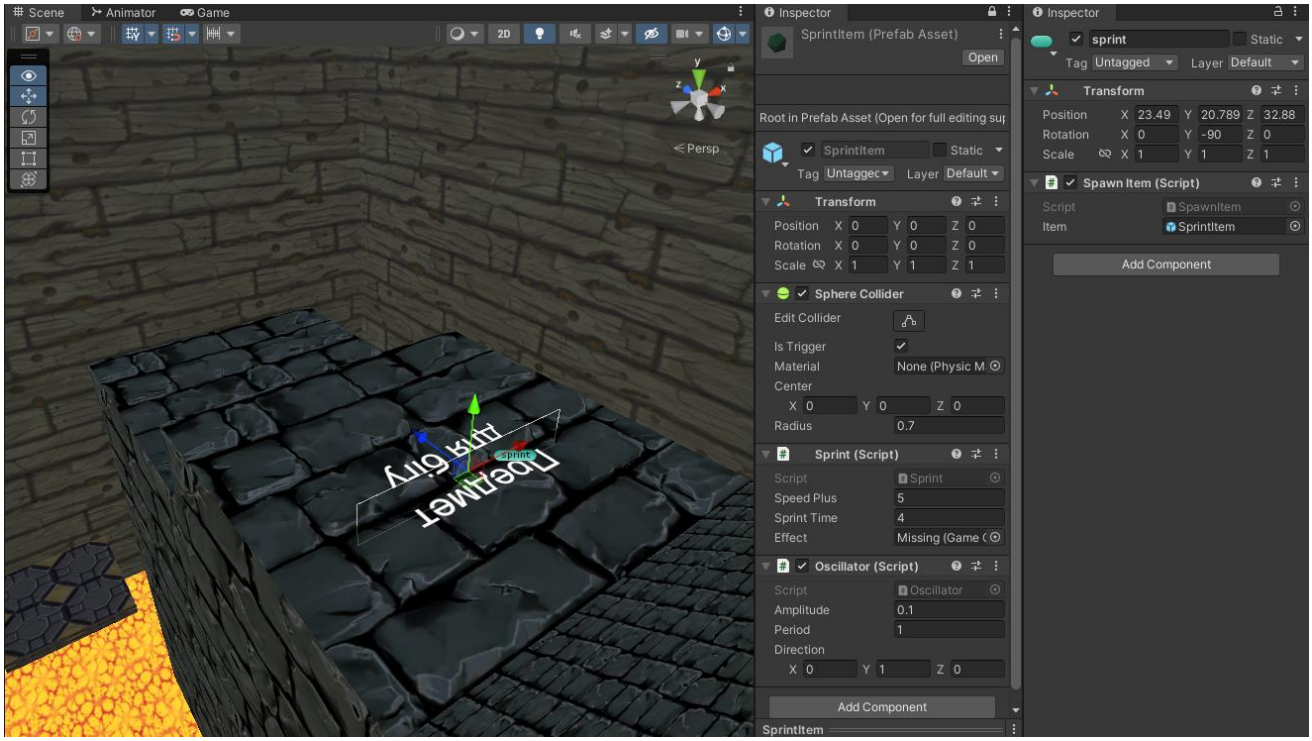


Рисунок 1.35. Налаштування інтерактивного об’єкта на сцені

У сцені об’єкт розміщується за допомогою компонента Transform, де вказано точну позицію, обертання та масштаб. Через тег або шар (Layer) предмети можна легко ідентифікувати для окремої обробки логікою гравця.

До об’єкта додано кілька користувацьких скриптів, серед яких основний відповідає за функціональність предмета. У ньому визначено ключові параметри: на скільки одиниць збільшується швидкість гравця (Speed Plus = 5) і як довго зберігається ефект (Sprint Time = 4 секунди). Таким чином, при контакті з

предметом гравець тимчасово отримує пришвидшення, що відкриває можливість долати складні ділянки рівня.

Щоб підсилити візуальне сприйняття, до предмета додано також анімаційний скрипт типу Oscillator, який надає об’єкту легке коливання вгору-вниз. Це робить предмет більш помітним та привабливим для гравця. Параметри Amplitude (0.1) та Period (1) визначають плавність руху, а напрямок (Y = 1) - вертикальну вісь коливання.

В окремому об'єкті або на платформі, де розташований предмет, був реалізований скрипт типу SpawnItem, що відповідає за появу бонусу. Це дозволяє зручно керувати кількістю та частотою появи таких об’єктів на рівні, наприклад, генерувати їх при старті або після певних подій. Через поле Item у цьому скрипті вказується префаб, який потрібно створити.

1.8 Тестування проекту

Процес роботи супроводжувався постійним тестуванням функціональності, перевіркою взаємодії між об’єктами та налаштуванням анімацій. Кожен крок був спрямований на створення технічно цілісного ігрового прототипу, де всі елементи взаємодіють між собою логічно, візуально узгоджено й відповідають завданням ігрового дизайну. Особлива увага приділялась тестуванню основних механік - таких як керування персонажем, робота фізичних об’єктів, коректність колізій та передбачуваність рухів.

Одним із ключових етапів стало тестування системи введення, яка є незалежною від типу пристрою введення. Була перевірена її сумісність з клавіатурою. Під час тестів перевірялись можливості пересування, стрибків, ривків та взаємодії з активними предметами, щоб переконатись у правильному реагуванні на введення.

Наступним напрямом стало випробування механіки подвійного стрибка. Для цього створено спеціальні об'єкти, що активують або деактивують відповідну здатність персонажа. Перевірялося, чи зникає предмет після активації, чи з’являється ефект, і чи змінюється фізична поведінка персонажа в повітрі.

Повторне проходження зони без перезапуску також тестувалось для виявлення можливих багів у повторному отриманні бонусу.

Окремий блок тестування був присвячений роботі камери. Її позиція, швидкість згладжування та реакція на поворот через клавіші Q та E тестувались у реальних ігрових умовах. Було перевірено, чи зберігається стабільне спостереження за персонажем навіть під час різких змін напряму або при активації об'єктів на сцені.

В умовах гри протестована поведінка спавну з таймером: чи не з’являється предмет раніше часу, чи зникає старий об'єкт при появі нового. Також перевірялась система появи ефектів після взаємодії, зокрема, при телепортації, втраті контролю персонажем або підборі бонусу.

Тестувались платформи, які мають горизонтальне або вертикальне коливання. Було важливо переконатися, що ці платформи рухаються плавно, зберігають задану амплітуду і не створюють аномалій під час руху персонажа поверх них. Також перевірялась поведінка персонажа у момент стику з платформою - особливо, чи передається рух платформи на гравця через фізику або при зіткненні.

Під час фінального етапу тестування був проведений перегляд усіх об'єктів на сцені: платформи, сфери, колони, активні предмети та декоративні елементи. Було перевірено, щоб усі зони були прохідні, не було блокуючих колізій або невидимих стін, а також щоб всі компоненти працювали стабільно при запуску сцени з будь-якого положення.

2 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

2.1 Аналіз небезпечних і шкідливих факторів, що впливають на програміста під час розробки програмного комплексу

Під час розробки програмного комплексу програміст проводить значну частину часу в офісному середовищі, де основним знаряддям праці є персональний комп'ютер. Таке робоче середовище, незважаючи на уявну безпеку, характеризується наявністю низки факторів, що мають потенційно шкідливий вплив на здоров'я працівника. В процесі роботи на користувачів персональних комп'ютерів можуть мати вплив наступні небезпечні та шкідливі фактори:

- Невідповідність параметрів мікроклімату нормам;
- Недостатній рівень освітленості;
- Ураження електрострумом;
- Статична електрика;
- Надмірне напруження зору внаслідок тривалої роботи з екранами дисплеїв
- Висока статичність робочої пози — тривале сидіння без змін положення тіла
- Можливість розвитку професійних захворювань опорно-рухового апарату (остеохондроз, тунельний синдром)

2.2 Гігієнічні вимоги до виробничого середовища

Потрібно враховувати санітарні нормативи освітлення, вимоги до параметрів мікроклімату, ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електро- магнітного, ультрафіолетового та інфрачервоного полів. Конкретні показники зазначених санітарних норм викладені в Державних санітарих правилах і нормах роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98.

Температура в приміщенні повинна підтримуватись у межах 20–24°С, а відносна вологість — 40–60%. Надто сухе повітря провокує пересихання слизових

оболонок, очей і може спричинити швидке зниження працездатності. Навпаки, надмірна вологість посилює відчуття втоми й може викликати появу плісняви на обладнанні. Наявність справної вентиляції або кондиціонування — обов'язкова умова для підтримання мікроклімату на стабільному рівні, особливо в теплий період року.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп'ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні. Бажано розмістити комп'ютери в кімнаті таким чином, щоб світло потрапляло на екрани моніторів з півдня чи північного сходу.

У разі надмірного шуму чи вібрації технічного обладнання, роботодавець повинен забезпечити працівників антивібраційними килимками.

2.3 Вимоги до організації робочого місця працівника

Роботодавець, який використовує найману працю робітників, повинен забезпечити відповідність їхніх робочих місць комфортним та безпечним умовам. Розмір одного робочого місця має становити не менше 6 квадратних метрів. При необхідності, суміжні робочі місця співробітників, що працюють з комп'ютером, слід розділити перегородками висотою до 2 метрів. При визначенні достатнього розміру приміщення і робочого місця на одну особу необхідно додатково враховувати шафи, сейфи, тумби або інші предмети меблів чи обладнання, які знаходяться в кімнаті. На столі працівника можливо розмістити допоміжні для роботи пристрої (принтери, колонки, сканери), а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику. Робочий стілець співробітника має бути підйомно-поворотним, легко регульованим за висотою та забезпечувати належну підтримку та зручне положення спини і хребта особи. Щодня необхідно проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп'ютера від запиленості.

На підприємстві забороняється: проводити ремонт та технічне обслуговування комп'ютера за робочим місцем працівника; самочинно ремонтувати або

намагатись здійснити технічне налагодження комп'ютера без залучення компетентних спеціалістів; складувати на робочому місці зайві документи, деталі та предмети, що не потрібні для роботи; використовувати монітори з нечітким зображенням та монітори, у яких наявні поламки екрану; працювати з матричним принтером без антивібраційного покриття та зі знятою кришкою.

Допускати до роботи осіб, які не пройшли інструктаж з охорони праці для роботи з комп'ютером, не дозволяється.

2.4 Електробезпека

Приміщення, у якому розміщено обладнання з імпульсними джерелами живлення, згідно з вимогами ОНТП 24-86 та Правил улаштування електроустановок (ПУЕ-87), класифікується як таке, що не має підвищеної небезпеки ураження електричним струмом. Такий висновок ґрунтується на дотриманні декількох важливих умов: відносна вологість повітря не перевищує 75%, температурні параметри повітря в межах допустимих значень, тобто не вище 35 °С, а також відсутність хімічно агресивного середовища, що могло б погіршити ізоляційні властивості матеріалів або спричинити корозійне ураження елементів електромережі.

Живлення всіх електричних пристроїв у приміщенні здійснюється від двофазної електричної мережі з напругою 220 В і частотою 50 Гц, з обов'язковим заземленням нейтралі. Установлення автоматичних вимикачів захисту від перевантаження і короткого замикання є необхідною умовою для зниження ризиків надмірного струмового навантаження. Для підтримання високого рівня безпеки електропостачання впроваджується схема захисного заземлення, що дає змогу зменшити ймовірність ураження людини електричним струмом у разі аварійних ситуацій, таких як пробій ізоляції або порушення роботи приладу.

Заземлення виконується за допомогою гнучкого мідного дроту зі сплетенням і поперечним перерізом не менш ніж 1,5 мм². Важливим елементом системи електробезпеки є повторне захисне заземлення нульового проводу, що забезпечує зниження напруги дотику при різних режимах роботи обладнання. Відповідно до вимог ГОСТ 12.2.007.0-75, усі пристрої, що використовуються в

системі, крім ЕОМ II класу, відносяться до I класу і повинні мати надійну робочу ізоляцію, згідно з вимогами ГОСТ 12.1.009-76.

Підключення апаратури здійснюється відповідно до вимог Правил безпечної експлуатації електроустановок (ПБЕ) і ПУЕ. Усі вимоги дотримані, додаткові заходи з електробезпеки не потребуються, що свідчить про належний рівень технічного та нормативного забезпечення безпечної експлуатації обладнання.

2.5 Пожежна безпека

Організація пожежної безпеки в приміщеннях, де експлуатуються персональні комп’ютери та інше електронне обладнання, вимагає ретельного врахування специфіки розташованої в них техніки та чутливості електроніки до дії певних вогнегасних засобів. При виборі первинних засобів пожежогасіння необхідно віддавати перевагу таким засобам, які не спричиняють пошкодження обладнання, зокрема рекомендовано використовувати вуглекислотні вогнегасники. Вуглекислота, на відміну від води чи піни, не залишає слідів, не викликає короткого замикання та не ушкоджує елементну базу комп’ютерної техніки, що є принципово важливим для збереження працездатності систем.

Будівлі, приміщення, обладнання та інші об’єкти повинні бути забезпечені первинними засобами пожежогасіння, які включають вогнегасники, ящики з сухим піском, негорючі теплоізоляційні покривала з повсті чи грубововняної тканини, а також відповідний інструмент для гасіння пожеж на ранній стадії розвитку. Розміщення цих засобів повинне відповідати нормам пожежної безпеки й бути організованим таким чином, щоб забезпечити їх швидке виявлення та доступність у критичний момент. Саме тому в приміщенні встановлюються відповідні позначки, які мають відповідати чинним державним стандартам і бути чітко помітними на висоті 2–2,5 м від рівня підлоги.

Розміщення переносних вогнегасників має здійснюватися згідно з такими умовами: вогнегасники можуть бути навішені на вертикальні конструкції не вище ніж на 1,5 м від рівня підлоги до нижнього краю, з урахуванням того, щоб вони не перешкоджали відкриттю дверей; також допускається їхнє встановлення в

спеціальні тумби, пожежні шафи або монтаж на кронштейни. Розташування вогнегасників має забезпечувати видимість і можливість прочитання маркування на корпусі пристрою без потреби його переміщення. Такий підхід спрямований на забезпечення високого рівня оперативності реагування на пожежну небезпеку та зменшення ризику поширення вогню в середовищі з наявністю електронного обладнання.

У разі виникнення пожежної небезпеки важливим чітко організована процедура евакуації працівників. У приміщеннях, де розміщено комп'ютерне обладнання, повинні бути облаштовані вільні й позначені евакуаційні шляхи, що ведуть до основних та запасних виходів. Усі маршрути евакуації мають бути завчасно перевірені на наявність перешкод, забезпечені відповідними вказівниками, які світяться в темряві або дублюються фотолюмінісцентними елементами. Працівники мають бути ознайомлені з порядком дій у разі тривожного сигналу: негайне припинення роботи, відключення обладнання (за можливості), організоване залишення робочого місця без створення паніки та блокування проходів. Відпрацьовані плани евакуації повинні бути розміщені в доступних місцях та містити зрозумілу схему маршруту, позначення всіх виходів і розташування первинних засобів пожежогасіння. Регулярне проведення тренувальних евакуацій дозволяє мінімізувати час реагування та знижує ризики для життя та здоров'я персоналу в умовах реальної небезпеки.

ВИСНОВКИ

У межах цієї роботи було виконано комплексне дослідження, проектування та реалізацію основних складових ігрового проекту на платформі Unity, зосередженого на створенні динамічного геймплею з паркурною механікою. Починаючи з концептуальної ідеї та аналізу подібних ігор, було сформовано бачення цілісного ігрового досвіду, що спирається на точну фізичну модель керування персонажем, а також ретельно продуманий левелдизайн, який поєднує складність, ритм і поступове нарощення викликів.

Під час реалізації ігрової механіки особливу увагу приділено імплементації системи керування персонажем: було додано спеціалізовані компоненти, налаштовано фізичні параметри, розроблено логіку стрибків, ривків, уповільнення падіння, а також враховано взаємодію з середовищем, включаючи телепортацію у випадку падіння в лаву чи дотику до стін. Кожен параметр був вивірений експериментальним шляхом для досягнення чутливого, плавного й водночас точного керування.

Візуальна складова гри була побудована з використанням якісних ассетів зі стилізованим low poly дизайном. Зокрема, застосовано безкоштовну RPG-модель персонажа, адаптовану під систему контролю руху, а також набір смарт-матеріалів, що надало проекту оригінальну візуальну ідентичність. Архітектура рівня включає статичні й рухомі платформи, сфери, колони, трикутні перешкоди та інші елементи, які не лише формують просторову структуру, а й беруть участь у механіці переміщення.

Процес роботи супроводжувався постійним тестуванням функціональності, перевіркою взаємодії між об’єктами та налаштуванням анімацій. Кожен крок був спрямований на створення технічно цілісного ігрового прототипу, де всі елементи взаємодіють між собою логічно, візуально узгоджено й відповідають завданням ігрового дизайну.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ален, Б. М. Програмування ігор для початківців на C#, Технічна література, 2019. - 304 с.
2. Грант, М. Unity в дії: створення 3D-ігор з використанням Unity, Львів: Видавництво Старого Лева, 2018. - 432 с.
3. Рей, М. Програмування ігор на C# з Unity Технічна література, 2019. - 368 с.
4. Секстон, Дж. Unity. Основи створення ігрових рівнів, 2017. - 320 с.
5. Чандлер, Дж. Основи роботи з фізикою в Unity. Технічна література, 2018. - 280 с.
6. Ліберті, Дж. Основи програмування на C#. Технічна література, 2019. - 416 с.
7. Трост, А. Основи 3D-графіки та моделювання в ігрових рушіях, Одеса: Технопрес, 2020. - 292 с.
8. Ріхтер, Д. CLR via C#. Програмування на платформі Microsoft .NET Framework. Технічна література, 2020. - 896 с.
9. Брей, Т. Програмування ігрових світів. Структура, логіка, взаємодія, Львів: Літера, 2018. - 348 с.
10. Unity Technologies. Unity User Manual 2022 LTS [Електронний ресурс]. - https://docs.unity3d.com/Manual/index.html (дата звернення: 20.05.2024).
11. Unity Asset Store – Free Low Poly Human RPG Character [Електронний ресурс]. -https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/free-low-poly-human-rpg-character-219979 (дата звернення: 20.05.2024).
12. Unity Asset Store – KCISA Korean Traditional Smart Materials Vol.6 [Електронний ресурс]. -https://assetstore.unity.com/packages/2d/textures-materials/tiles/kcisa-korean-traditional-smart-materials-vol-6-303901 (дата звернення: 20.05.2024).

ДОДАТОК Б. Лістинг коду

Частина коду MovementCharacterController.cs

```
namespace PlatformCharacterController
{
public class MovementCharacterController : MonoBehaviour
{
[Header("Player Controller Settings")] [Tooltip("Speed for the player.")]
public float RunningSpeed = 5f;
public float SlopeLimit = 45;
public float SlideFriction = 0.3f;
public float Gravity = -30f;
[Tooltip("Max speed for the player when fall.")] [Range(0, 100)]
public float MaxDownYVelocity = 15;
public bool CanControl = true;
[Header("Jump Settings")] [Tooltip("This allow the character to jump.")]
public bool CanJump = true;
public float JumpHeight = 2f;
public bool CanDoubleJump = true;
[Header("Dash Settings")]
public bool CanDash = true;
public float DashForce = 5f;
public Animator PlayerAnimator;
[Header("Effects")] [Tooltip("This position is in the character feet and is use to
instantiate effects.")]
public Transform LowZonePosition;
public GameObject JumpEffect;
public GameObject DashEffect;
[Header("Use this to capture inputs")] public Inputs PlayerInputs;
[Header("Platforms")] public Transform CurrentActivePlatform;
private Vector3 _moveDirection;
private Vector3 _activeGlobalPlatformPoint;
private Vector3 _activeLocalPlatformPoint;
private Quaternion _activeGlobalPlatformRotation;
private Quaternion _activeLocalPlatformRotation;
private Transform _characterTransform;
private CharacterController _controller;
private Vector3 _velocity;
private float _horizontal;
private float _vertical;
private bool _jump;
private bool _dash;
private Transform _cameraTransform;
private Vector3 _forward;
private Vector3 _right;
private float _originalRunningSpeed;
private float _dashCooldown;
private float _gravity;
private bool _doubleJump;
private bool _isCorrectGrounded;
private bool _isGrounded;
private bool _activeFall;
private Vector3 _hitNormal;
```

```
private Vector3 _move;
private Vector3 _direction;
private void Awake()
{
PlayerInputs = GetComponent<Inputs>();
_controller = GetComponent<CharacterController>();
_characterTransform = transform;
_originalRunningSpeed = RunningSpeed;
}
private void Start()
{
_cameraTransform = Camera.main.transform;
_dashCooldown = DashCooldown;
_gravity = Gravity;
}
private void Update()
{
CheckGroundStatus();
_horizontal = PlayerInputs.GetHorizontal();
_vertical = PlayerInputs.GetVertical();
_jump = PlayerInputs.Jump();
_dash = PlayerInputs.Dash();
if (_jump && !HoldingObject)
{
Jump(JumpHeight);
}
if (_dash && !HoldingObject)
{
Dash();
}
if (DashCooldown > 0)
{
DashCooldown -= Time.fixedDeltaTime;
}
else
{
DashCooldown = 0;
}
SetRunningAnimation((Math.Abs(_horizontal) > 0 || Math.Abs(_vertical) > 0));
if (!CurrentActivePlatform || !CurrentActivePlatform.CompareTag("Platform"))
return;
if (CurrentActivePlatform)
{
var newGlobalPlatformPoint =
CurrentActivePlatform.TransformPoint(_activeLocalPlatformPoint);
_moveDirection = newGlobalPlatformPoint - _activeGlobalPlatformPoint;
if (_moveDirection.magnitude > 0.01f)
{
    _controller.Move(_moveDirection);
}
if (!CurrentActivePlatform) return;
var newGlobalPlatformRotation = CurrentActivePlatform.rotation *
_activeLocalPlatformRotation;
```

```
var rotationDiff = newGlobalPlatformRotation *
Quaternion.Inverse(_activeGlobalPlatformRotation);
rotationDiff = Quaternion.FromToRotation(rotationDiff * Vector3.up, Vector3.up) *
rotationDiff;
_characterTransform.rotation = rotationDiff * _characterTransform.rotation;
_characterTransform.eulerAngles = new Vector3(0, _characterTransform.eulerAngles.y,
0);
UpdateMovingPlatform();
}
else
{
if (!(_moveDirection.magnitude > 0.01f)) return;
_moveDirection = Vector3.Lerp(_moveDirection, Vector3.zero, Time.deltaTime);
_controller.Move(_moveDirection);
}}
private void FixedUpdate()
{
_forward = _cameraTransform.TransformDirection(Vector3.forward);
_forward.y = 0f;
_forward = _forward.normalized;
_right = new Vector3(_forward.z, 0.0f, -_forward.x);
_move = (_horizontal * _right + _vertical * _forward);
_direction = (_horizontal * _right + _vertical * _forward);
if (!_isCorrectGrounded && _isGrounded)
{
_move.x += (1f - _hitNormal.y) * _hitNormal.x * (1f - SlideFriction);
_move.z += (1f - _hitNormal.y) * _hitNormal.z * (1f - SlideFriction);
}
_move.Normalize();
if (!_slowFall && _controller.enabled)
{
_controller.Move(Time.deltaTime * RunningSpeed * _move);
}
_isCorrectGrounded = (Vector3.Angle(Vector3.up, _hitNormal) <= SlopeLimit);
if (_direction != Vector3.zero)
{
transform.forward = _direction;
}
if (_velocity.y >= -MaxDownYVelocity)
{
_velocity.y += Gravity * Time.deltaTime;
}
SetGroundedState();
}
```

Частина коду Inputs.cs

```
public abstract class Inputs : MonoBehaviour
{public abstract float GetHorizontal();
public abstract float GetVertical();
public abstract bool Jump();
public abstract bool Dash();}
```

Частина коду SpawnItem.cs

```
public class SpawnItem : MonoBehaviour
{
public GameObject Item;
private GameObject _currentItem;
private float _spawnTime = 4;
private void FixedUpdate()
{
if (!_currentItem)
{
_spawnTime -= Time.deltaTime;
if (_spawnTime <= 0)
{
_spawnTime = 4;
SpawnTheItem();
}}}
public void SpawnTheItem()
{
if (_currentItem)
{
Destroy(_currentItem);
}
if (Item)
{
_currentItem = Instantiate(Item, transform.position, transform.rotation);
}
else
{
Debug.LogWarning("Please add a item to spawn.");
}}}
```

Частина коду Oscillator.cs

```
public class Oscillator : MonoBehaviour
{
public float m_Amplitude = 1.0f;
public float m_Period = 1.0f;
public Vector3 m_Direction = Vector3.up;
Vector3 m_StartPosition;
void Start()
{
m_StartPosition = transform.position;
}
void FixedUpdate()
{
var pos = m_StartPosition + m_Direction * m_Amplitude * Mathf.Sin(2.0f * Mathf.PI *
Time.time / m_Period);
transform.position = pos;}}
```

Частина коду Teleport.cs

```
namespace PlatformCharacterController
{
public class Teleport : MonoBehaviour
{
[Tooltip("Time to start teleporting the player.")]
public float StartTeleport;
[Tooltip("The player wait this time to can control the player again.")]
public float TimeToControlPlayer;
[Tooltip("Effect to start teleport.")] public GameObject TeleportEffect;
public Transform TeleportPosition;
private IEnumerator TeleportPlayer(Transform player)
{
yield return new WaitForSeconds(StartTeleport);
if (TeleportEffect)
{
Instantiate(TeleportEffect, player.position, player.rotation);
}
player.position = TeleportPosition.position;
}
private void OnTriggerEnter(Collider other)
{
if (other.CompareTag("Player"))
{
StartCoroutine(other.GetComponent<MovementCharacterController>()
.DeactivatePlayerControlByTime(TimeToControlPlayer));
StartCoroutine(TeleportPlayer(other.transform));
}}}}
```

Частина коду Sprint.cs

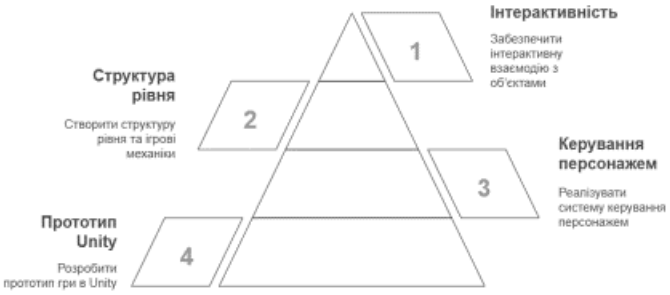
```
namespace PlatformCharacterController
{
public class Sprint : MonoBehaviour
{
[Tooltip("The speed value to add at the player speed.")]
public float SpeedPlus = 5;
[Tooltip("This is the time that the speed plus is active in the player.")]
public float SprintTime = 4;
public GameObject Effect;
private bool _active = true;
private void SprintPlayer(MovementCharacterController player)
{
_active = false;
if (Effect)
{Instantiate(Effect, transform.position, transform.rotation);}
player.ChangeSpeedInTime(SpeedPlus, SprintTime);
Destroy(gameObject);
}
private void OnTriggerEnter(Collider other)
{
if (other.CompareTag("Player") && _active){
SprintPlayer(other.GetComponent<MovementCharacterController>());}}}}
```

ДОДАТОК Б. Слайди мультимедійної презентації

ПРЕЗЕНТАЦІЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Здобувача освіти групи 2БКС-29:
Головаченка Леоніда Дмитровича

Мета роботи



Аналіз подібних ігор

Only Up!

Celeste

Інструменти розгорбки гри

Ігровий рушій

Unity використовується як ігровий рушій.

Мова програмування

C# є вибраною мовою програмування.

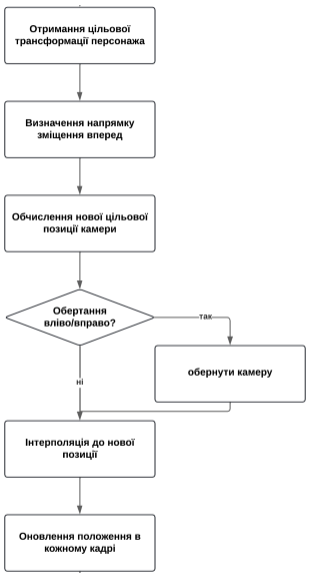
Редактор коду

Visual Studio Code використовується для написання коду.

Архітектура системи

- Зв'язок інпутів із поведінкою персонажа через абстрактну систему введення
- Архітектура з чітким поділом: інпут → обробка → реакція в ігровому середовищі
- Гнучкість, модульність і точність руху завдяки відокремленню скриптів за функціональністю

Функціональні схеми проекту

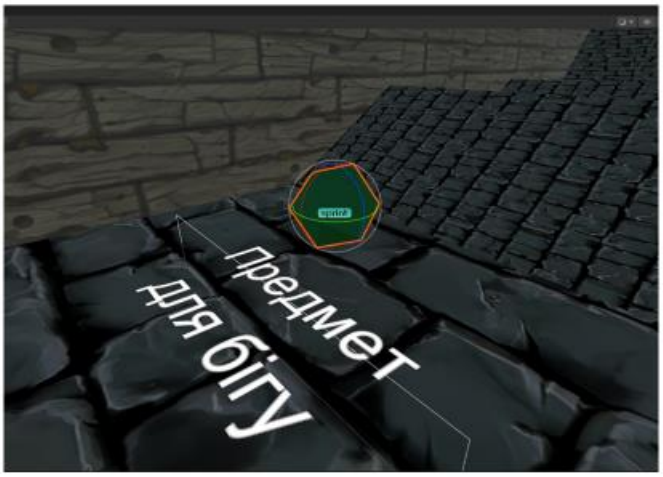


Реалізація персонажа

- Система керування
- Анімація персонажу
- Налаштування харктеристик: швидкість, гравітація, подвійний стрибок, ривок
- Модель персонажу

Інтерактивні предмети

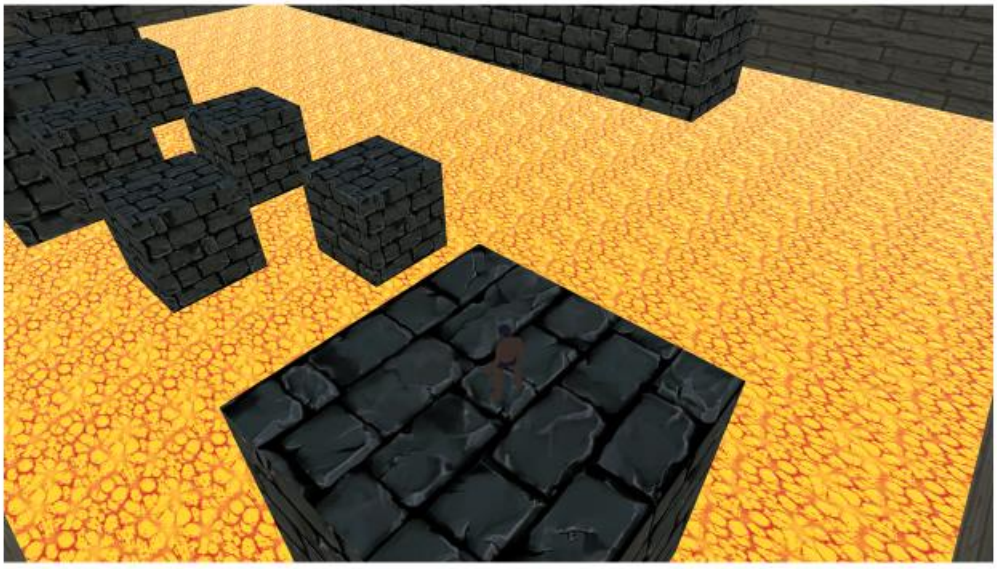
- Система появи об'єктів із затримкою
- Створення предметів, що взаємодіють із гравцем
- Змінення базових характеристик персонажу



Камера в проекті

- Постійно слідує за гравцем у тривимірному просторі
- Зберігає стабільну відстань та кут огляду
- М'яке згладжування руху для плавного спостереження
- Можливість повертати камеру навколо персонажа
- Управління поворотом реалізоване через клавіші Q (вліво) та E (вправо)
- Додатково підтримує функції обертання через скрипт

Тестування та результати



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача (здобувачки) освіти

відділення комп'ютерних систем

Головаченка Леоніда Дмитровича

(прізвище, ім`я та по батькові)

Спеціальність *123 Комп'ютерна інженерія*

Освітньо-професійна програма *Комп'ютерна інженерія*

Керівник кваліфікаційної роботи

Іванова Лілія Вікторівна

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи

«Створення інформаційного чат-боту для абітурієнтів коледжу »

Обсяг розрахунково-пояснювальної записки *63* сторінок

Обсяг графічної (презентаційної) частини *12* аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаного кваліфікаційної роботи завданню

кваліфікаційна робота у повному обсязі відповідає темі та завданню

б) характеристика виконання кожного розділу кваліфікаційної роботи

Кваліфікаційна робота складається з розділів: 1. Аналітичний огляд існуючих рішень. 2. Обґрунтування вибору технологій для створення ігрових проектів. 3. Архітектура ігрового проекту. 4. Розробка елементів дизайну та інтерфейсу користувача. 5. Створення макету та прототипу гри. 6. Тестування гри. 7.Охорона праці. 8. Висновки. 9. Список використаних джерел інформації.

в) оцінка якості виконання пояснювальної записки та графічної частини кваліфікаційної роботи

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у кваліфікаційній роботі. Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

г) перелік позитивних якостей кваліфікаційної роботи

1. Актуальна тематика

2. Сучасні технології реалізації програмного продукту

3. Якісне подання результатів роботи

д) основні недоліки кваліфікаційної роботи

Не визначеність головного меню та базових налаштувань

Відсутність опису реалізації і тестування левел-дизайну

У розділі тестування відсутні конкретні результати

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Рудніченко Микола Дмитрович*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

«23» 06 2025 р.

Відокремлений структурний підрозділ

Одеський технічний фаховий коледж ОНТУ

ВІДГУК

Керівника про кваліфікаційну роботу бакалавра

Головаченка Леоніда Дмитровича

(прізвище, ім`я та по батькові)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи *«Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левелдизайну»*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) Обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті

Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) Самостійність роботи над кваліфікаційною роботою

Студент самостійно обрала напрям та тематику кваліфікаційної роботи. Провів аналіз існуючих рішень і зробив необхідні висновки для реалізації проекту. Виявив навички самостійно опрацьовувати новий матеріал та виконувати пошук необхідної літератури та інших джерел інформації

в) Теоретична підготовка бакалавра_____

відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва_____

У роботі представлено результати розробки ігрового проєкту на платформі Unity з акцентом на побудову повноцінної системи керування персонажем та створення ефективного левелдизайну. Особливу увагу приділено не лише технічному втіленню базових механік руху, а й архітектурі, що дозволяє масштабувати логіку керування для різних типів пристроїв та платформ. Отримані кількісні результати стосуються продуктивності системи, часу реакції на дії гравця, плавності анімацій та стабільності роботи на різних типах пристроїв.

Загальна оцінка_____5(відмінно)_____

Прізвище, ім`я, по батькові_____Іванова Лілія Вікторівна_____

Місце роботи і посада керівника проєкту_ВСП «Одеський технічний фаховий коледж ОНТУ» к.т.н., зав. кафедрою Комп'ютерної інженерії_____

Підпис _____

«20» 06 2025р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Головаченко Л.Д.
здобувач освіти гр. 2БКС-29, та

Іванова Л.В.,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левел-дизайну» (автор роботи – Головаченко Л.Д., керівник роботи – Іванова Л.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Головаченко Л.Д. /

Керівник _____ / Іванова Л.В. /

«20» червня 2025 р.

ДОВІДКА

кафедри комп'ютерної інженерії
про допуск до захисту кваліфікаційної роботи
здобувача (здобувачки) освіти II курсу
відділення комп'ютерних систем групи 2БКС-29

Головаченка Леоніда Дмитровича

на тему *Створення ігрового проекту на платформі Unity: розробка ігрової механіки та левел-дизайну*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до кваліфікаційної роботи виконана з некритичними порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *20.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 19.06.2025 р. значення коефіцієнту подібності в роботі становить 7,90%, коефіцієнт цитування – 1,51%.*

(підпис) *20.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) кваліфікаційної роботи

здобувача (здобувачки) освіти *Головаченка Л.Д.* (П.І.Б.)

проведена « *20* » *червня* 2025 р.

Висновки *Пояснювальна записка до кваліфікаційної роботи виконана у повному обсязі. Випускна кваліфікаційна робота відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Зав. кафедри КІ (підпис) *Іванова Л.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/19/2025
Дата редагування 6/19/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Створення ігрового проєкту на платформі Unity: розробка ігрової механіки та левел-дизайну
Автор
Головаченко Леонід Дмитрович
Науковий керівник / Експерт
Іванова Лілія Вікторівна
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1

КЦ

25
Довжина фрази для коефіцієнта подібності 2

13383
Кількість слів

108703
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	·	1
Білі знаки	ß	0
Парафрази (SmartMarks)	a	50

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	http://www.kpl.volyn.ua/download/ohorona/47.pdf	110 0.82 %
2	http://www.kpl.volyn.ua/download/ohorona/47.pdf	71 0.53 %
3	http://www.kpl.volyn.ua/download/ohorona/47.pdf	59 0.44 %
4	https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download	52 0.39 %
5	https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download	49 0.37 %

6	http://www.kpl.volyn.ua/download/ohorona/47.pdf	48 0.36 %
7	https://card-file.ontu.edu.ua/bitstreams/0e6c3361-ffbf-4469-86a1-fe84a1fe21cd/download	36 0.27 %
8	https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download	31 0.23 %
9	http://www.kpl.volyn.ua/download/ohorona/47.pdf	30 0.22 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c5cd348b-fc64-4a25-9a5b-6cc8d62db909/content	25 0.19 %

з домашньої бази даних (0.58 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Аналіз продуктивності блокових криптоалгоритмів у багатоядерній системі [redacted] 6/18/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	78 (9) 0.58 %

з програми обміну базами даних (0.09 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Інформаційна система «Відділ кадрів». Курсова робота [redacted] 12/5/2024 Zhytomyr Agricultural Technical Professional College (Zhytomyr Agricultural Technical Professional College)	12 (1) 0.09 %

з Інтернету (7.23 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**http://www.kpl.volyn.ua/download/ohorona/47.pdf**	**341 (7) 2.55 %**
2	**https://card-file.ontu.edu.ua/bitstreams/0e6c3361-ffbf-4469-86a1-fe84a1fe21cd/download**	**264 (21) 1.97 %**
3	**https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download**	**172 (8) 1.29 %**
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	52 (4) 0.39 %
5	**https://card-file.ontu.edu.ua/server/api/core/bitstreams/c5cd348b-fc64-4a25-9a5b-6cc8d62db909/content**	**36 (2) 0.27 %**
6	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	29 (2) 0.22 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/d5a3d14f-d5cb-460f-9c49-cba3f9d50554/content	23 (1) 0.17 %
8	https://csharp.hotexamples.com/examples/UnityEngine/Collider/CompareTag/php-collider-comparetag-method-examples.html	22 (2) 0.16 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/6eb6bf1c-5813-45e6-93c5-25539b4709d3/content	11 (1) 0.08 %
10	https://cv.archives.gov.ua/pdf/pozhezhka.pdf	10 (2) 0.07 %
11	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	7 (1) 0.05 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)