Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка мобільного 2D-платформера на Unity »

Виконав: студент групи К21-2
Спеціальність 122 Комп'ютерні науки
Усенко А.І.
(прізвище та ініціали)
Керівник доц. Фірсов О. Д.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
Доцент кафедри кібербезпеки та інфомармаційних технологій
(посада)
к.т.н., доц..
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Усенко А.І Розробка 2D платформеру з використанням Unity. Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Об'єктом дослідження є процес розробки інтерактивних ігрових застосунків у середовищі Unity.
Предмет дослідження – створення 2D платформера з елементами взаємодії, анімації та фізики на основі рушія Unity та мови програмування C#.

Метою роботи є розробка повнофункціонального 2D платформера, що забезпечує динамічний ігровий процес, адаптований до різних пристроїв, з особливою увагою до мобільних платформ.

Дана робота присвячена створенню 2D платформера, який дозволяє гравцю керувати персонажем, взаємодіяти з елементами рівня, збирати об'єкти, уникати перешкод та досягати цілей. У процесі дослідження проаналізовано сучасні підходи до розробки ігор на Unity, обґрунтовано вибір ігрової архітектури, систем контролю персонажа, колізій, анімацій та оптимізації.

Особливу увагу приділено адаптації гри для мобільних пристроїв, включаючи управління дотиком, масштабування інтерфейсу та зниження навантаження на систему. Практична цінність роботи полягає у створенні ігрового прототипу, який може бути основою для подальшої комерціалізації або використання в освітньому процесі з розробки ігор.

Ключові слова: 2D платформер, Unity, C#, мобільна гра, ігровий рушій, фізика, анімація, оптимізація.

ABSTRACT

Usenko A.I. Development of a 2D Platformer Using Unity. Master's thesis for obtaining the degree of Master in specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2024.

The object of the research is the process of developing interactive game applications in the Unity environment.

The subject of the research is the creation of a 2D platformer with elements of interaction, animation, and physics based on the Unity engine and the C# programming language.

The aim of the thesis is to develop a fully functional 2D platformer that provides a dynamic gameplay experience and is adapted for various devices, with a particular focus on mobile platforms.

This work is dedicated to the creation of a 2D platformer that allows the player to control a character, interact with level elements, collect items, avoid obstacles, and achieve objectives. The research includes analysis of modern approaches to game development in Unity, justification of the chosen game architecture, character control systems, collisions, animations, and optimization techniques.

Special attention is given to adapting the game for mobile devices, including touch control implementation, interface scaling, and system performance optimization. The practical value of this work lies in the creation of a game prototype that can serve as a basis for future commercialization or educational use in game development courses.

Keywords: 2D platformer, Unity, C#, mobile game, game engine, physics, animation, optimization.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах стрімкого розвитку інформаційних технологій та зростання популярності мобільних пристроїв, галузь розробки відеоігор займає важливе місце в цифровій економіці. Відеоігри давно перестали бути виключно розважальним засобом – вони стали інструментом навчання, творчої самореалізації, соціальної взаємодії та бізнесу. Особливу увагу привертають 2D платформери – жанр, який поєднує простоту реалізації з широкими можливостями для геймплейних рішень. Їхня популярність зумовлена доступністю, легкістю освоєння, а також гнучкістю у дизайні та механіці.

У цьому контексті особливо актуальною є розробка 2D платформера з використанням сучасних інструментів, таких як рушій Unity та мова програмування C#. Завдяки ним можливо створити кросплатформений продукт з якісною графікою, адаптованим під мобільні пристрої, що відповідає вимогам сучасного ринку.

Інноваційність даної роботи полягає у поєднанні класичних елементів жанру 2D платформерів із сучасними підходами до архітектури ігрового програмного забезпечення, а також у забезпеченні адаптації гри під мобільні платформи. Розробка враховує особливості сенсорного керування, обмеження ресурсів пристроїв та вимоги до продуктивності, що дозволяє отримати якісний кінцевий продукт.

Практичне значення результатів роботи полягає у створенні повнофункціонального ігрового застосунку, що може бути використаний як основа для комерційного продукту, інструменту для навчання або прикладного проекту в межах освітнього процесу.

Окрім того, розроблений проект може бути корисним для:

- демонстрації принципів розробки кросплатформених ігор.
- практичного освоєння архітектурних патернів у Unity.
- формування навичок оптимізації продуктивності на мобільних пристроях.

- впровадження елементів UI/UX дизайну з урахуванням сенсорного керування.

Цифровізація дозвілля, зростання мобільного сегменту та потреба в інноваціях у сфері розробки ігор формують високий попит на нові, ефективні підходи до створення 2D ігрових продуктів. Це, в свою чергу, ставить перед розробниками нові виклики: необхідність забезпечення гнучкої архітектури гри, стабільної роботи на різних пристроях, простого та інтуїтивного керування. У таких умовах зростає цінність глибокого аналізу технологій і рішень, що використовуються під час розробки.

Мета роботи – розробка повнофункціонального 2D платформера з використанням Unity та C#, який підтримує динамічний ігровий процес та адаптований до мобільних пристроїв.

Методи дослідження – аналіз сучасних технологій ігрової розробки, проектування архітектури програмного забезпечення, програмування на C#, методи оптимізації та тестування мобільних застосунків.

У відповідності до поставленої мети у кваліфікаційній роботі вирішувались наступні завдання:

1. Провести аналіз розвитку індустрії відеоігор та особливостей створення 2D платформерів.
2. Обґрунтувати вибір технологій та інструментів для реалізації проекту.
3. Розробити архітектуру гри та основні ігрові компоненти.
4. Реалізувати основний функціонал гри з урахуванням адаптації під мобільні пристрої.
5. Провести тестування та оптимізацію гри відповідно до функціональних та нефункціональних вимог.

Об’єкт дослідження – процес розробки інтерактивних ігрових застосунків у середовищі Unity.

Предмет дослідження – 2D платформер, реалізований за допомогою рушія Unity та мови C# з урахуванням особливостей мобільних платформ.

Практичне значення результатів роботи полягає у створенні повнофункціонального ігрового застосунку, що може бути використаний як основа для комерційного продукту, інструменту для навчання або прикладного проєкту в межах освітнього процесу.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 60 сторінок основного тексту, 38 рисунків та 1 таблицю.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Розвиток індустрії відеоігор у цифрову епоху

Індустрія відеоігор від моменту свого виникнення пройшла значний шлях розвитку, перетворившись із невеликих науково-технічних експериментів у масштабну глобальну галузь розваг. Перші відеоігри з'явилися в середині XX століття як прості програми для великих комп'ютерів, однак вже незабаром вони набули популярності серед широкої аудиторії.

У 1970-х роках індустрія зробила величезний стрибок завдяки появі аркадних автоматів, які залучили масового гравця. Ігри на кшталт Pong і Space Invaders стали символами епохи і популяризували ігрову культуру в суспільстві [1]. Водночас розпочався розвиток домашніх консолей, які дали змогу грати улюблені ігри не лише в аркадних залах, а й у власних оселях.

У 1980-х і 1990-х роках завдяки підвищенню потужності апаратного забезпечення і розвитку графічних технологій ігри почали набувати нових форм і жанрів. З'явилися складні сюжети, розгалужені ігрові світи, а також перші спроби інтеграції фізичних моделей і анімації, що зробило ігровий процес більш реалістичним і захоплюючим [2].

Сьогодні відеоігри — це не лише розвага, а й серйозна індустрія, що охоплює різні платформи: ПК, консолі, мобільні пристрої та хмарні сервіси. Розробка ігор стала комплексним процесом, що включає дизайн, програмування, анімацію, фізичне моделювання та оптимізацію для різних пристроїв. Особливе місце займають 2D та 3D платформи, що дозволяють створювати багатогранні ігрові світи і забезпечують широкий спектр можливостей для гравців.

Розвиток цифрових технологій продовжує трансформувати індустрію: впроваджуються штучний інтелект, мережеві рішення, а також механіки, адаптовані для мобільних і сенсорних пристроїв, що розширює аудиторію

користувачів. Завдяки цьому ігрові продукти стали доступнішими і різноманітнішими, а індустрія продовжує динамічно зростати.

Додатково варто зазначити, що відеоігри дедалі активніше проникають у суміжні сфери — освіту, охорону здоров'я, психологію та маркетинг. Гейміфікація, як інструмент залучення та мотивації, використовується в освітніх платформах, тренажерах для професійної підготовки та навіть у терапевтичних цілях. Такий міждисциплінарний підхід демонструє, що сучасна ігрова індустрія має не лише розважальний, а й потужний прикладний потенціал.

Особливу увагу сьогодні привертають інтерактивні симулятори та тренажери, які активно впроваджуються в різноманітні галузі:

- Медицина: використовуються симулятори для навчання хірургів, терапевтів і медичних сестер. Наприклад, віртуальні операційні кімнати дозволяють без ризику для пацієнтів відпрацьовувати складні маніпуляції та клінічні сценарії. Деякі системи використовують VR/AR для реалістичного занурення в процес лікування.
- Авіація: вже давно є прикладом використання ігрових технологій у серйозній підготовці. Пілоти тренуються на авіасимуляторах, які точно моделюють фізику польоту, погодні умови, несправності та аварійні ситуації.
- Військова справа: армії світу застосовують ігрові рушії (наприклад, Unreal Engine або Unity) для створення тренажерів, що моделюють бойові ситуації, управління технікою, взаємодію в команді тощо. Це забезпечує ефективне навчання без значних матеріальних витрат.
- Автомобільна галузь: симулятори керування автомобілем застосовуються для підготовки водіїв, тестування нових систем безпеки, адаптації електроніки до реальних дорожніх умов. Вони також корисні в дослідженні поведінки водія в екстремальних ситуаціях.
- Освіта: цифрові тренажери допомагають учням і студентам засвоювати матеріал у візуально-наочний спосіб, що сприяє кращому розумінню

складних тем, зокрема в технічних, інженерних і природничих дисциплінах.

- Психологія та нейрореабілітація: за допомогою ігрових програм здійснюється відновлення когнітивних функцій після травм або хвороб. Тренажери використовуються для корекції поведінки, тренування пам'яті, уважності, моторики тощо.

Завдяки активному впровадженню ігрових технологій у ці сфери, відеоігри вже давно перестали бути лише інструментом для відпочинку. Вони стали ефективним засобом навчання, дослідження, тренування та моделювання складних процесів, а сама індустрія набуває дедалі більшого суспільного значення.

1.2 Інструменти розробки відеоігор: огляд можливостей Unity

У процесі створення відеоігор ключову роль відіграє вибір програмного забезпечення, яке забезпечує розробника всіма необхідними засобами для реалізації креативних і технічних рішень (див. табл. 1.1). Одним із найпопулярніших середовищ для розробки як 2D, так і 3D ігор є Unity — мультиплатформенний ігровий рушій, який відзначається гнучкістю, потужною функціональністю та доступністю.

Unity дозволяє створювати інтерактивні проекти завдяки інтеграції графічного редактора, середовища розробки та численних вбудованих інструментів. Завдяки підтримці понад двадцяти платформ (Android, iOS, Windows, macOS, WebGL, PlayStation, Xbox, Nintendo Switch та ін.) Unity забезпечує можливість створення кросплатформенних ігор без значних змін у вихідному коді.

Ігрова логіка в Unity створюється за допомогою мови програмування C#, що надає доступ до повноцінного об'єктно-орієнтованого підходу, делегатів, подій, LINQ та інших сучасних можливостей. Зокрема, структура MonoBehaviour дозволяє реалізувати життєвий цикл ігрових об'єктів (методи

Start(), Update(), FixedUpdate() тощо), що є фундаментальним елементом архітектури Unity.

Для створення 2D-ігор Unity пропонує набір спеціалізованих інструментів, зокрема:

- Sprite Renderer — дозволяє відображати графіку у вигляді спрайтів, які є основними елементами візуального представлення 2D-об'єктів;
- Tilemap Editor — модуль для створення ігрових рівнів з тайлів, що дозволяє ефективно компонувати великі сцени з повторюваних елементів;
- Rigidbody2D та Collider2D — компоненти для фізичної симуляції, взаємодії об'єктів, гравітації та обробки зіткнень;
- Animator та Animation — система анімації, що дає змогу створювати плавні переходи між станами (наприклад, стояння, біг, стрибок) за допомогою state machine;
- Sorting Layers та Order in Layer — механізми керування порядком візуального відображення об'єктів на сцені.

Також Unity має потужний UI-систему (Canvas, RectTransform, Button, TextMeshPro тощо), яка дозволяє реалізовувати інтуїтивно зрозумілий користувацький інтерфейс навіть на пристроях з обмеженим розміром екрана, як-от смартфони. Для мобільних ігор важливою є також підтримка сенсорного керування через Input.touchCount, Touch.phase та Input.GetTouch(), що дозволяє налаштовувати ігрові механіки відповідно до особливостей мобільної взаємодії.

Однією з важливих переваг Unity є інтеграція з середовищем розробки Visual Studio, яка дозволяє розробнику працювати зі зручним автозаповненням, налагодженням коду та системою контролю версій. Крім того, Unity має Asset Store — цифровий маркетплейс, де можна знайти тисячі безкоштовних та платних ресурсів: графіка, моделі, звуки, скрипти, шаблони та повні ігрові системи, що значно пришвидшують процес розробки.

Unity надає широкі можливості для оптимізації мобільних застосунків — зменшення ваги збірки, контроль над якістю текстур, використання

компресованих аудіоформатів, а також можливість використовувати профайлери (Profiler, Memory Profiler, Frame Debugger) для відстеження продуктивності гри [3].

Вбудовані механізми налаштування якості (Quality Settings), управління ресурсами (Asset Bundles, Addressables), а також підтримка асинхронного завантаження сцен (SceneManager.LoadSceneAsync) дозволяють створювати продуктивні, плавні ігри, що коректно працюють на більшості мобільних пристроїв.

Окрім потужних технічних інструментів, Unity активно підтримує розробників — як новачків, так і професіоналів — через розвинену освітню екосистему та спільноту, що є одним із ключових факторів її популярності та широкого розповсюдження. Розробники мають доступ до великого обсягу офіційної документації, яка постійно оновлюється і містить чіткі пояснення щодо роботи з різними компонентами рушія, API, системами анімації, фізики, UI, AI тощо. У документації також містяться практичні приклади коду, які можна інтегрувати у власні проекти.

Одним з найцінніших ресурсів є Unity Learn — офіційна освітня платформа від Unity Technologies, яка пропонує безкоштовні та платні курси, інтерактивні навчальні шляхи (learning paths), проекты для практики та сертифікаційні програми. Unity Learn створено як для початківців, так і для досвідчених користувачів, і вона охоплює теми від базового ознайомлення з інтерфейсом редактора до розробки складних 2D/3D ігор, VR/AR-додатків, сценаріїв зі штучним інтелектом тощо.

Особливо важливо, що багато курсів супроводжуються інтерактивними туторіалами, в яких користувач може безпосередньо в середовищі браузера чи Unity Editor виконувати завдання, слідуючи інструкціям. Це створює комфортне середовище для практичного навчання без необхідності налаштовувати складне середовище вручну.

Також Unity надає вбудовану інтеграцію з Unity Hub, що дозволяє встановлювати демонстраційні проекти, шаблони, стартові сцени та офіційні

приклади, які можна одразу використати як базу для власного проекту. Такі шаблони охоплюють різні жанри — від платформерів і гонок до шутерів, квестів та VR-додатків. Це особливо корисно для тих, хто краще навчається через практику — вони можуть "розібрати" вже готову гру та зрозуміти, як реалізовані основні механіки.

Ще одним важливим елементом є офіційний форум Unity, який є платформою для обговорення технічних проблем, обміну досвідом, порад і практик між користувачами різного рівня. Багато досвідчених розробників та навіть працівників компанії активно беруть участь в обговореннях, допомагаючи вирішити складні питання. Крім того, є величезна база питань і відповідей на Unity Answers, що дозволяє швидко знаходити рішення поширених проблем.

Unity також підтримує спільноту розробників через конференції, хакатони, офіційні стріми, онлайн-змагання (наприклад, Unity Game Jam) і програми підтримки інді-розробників. Через ці заходи компанія не лише ділиться знаннями, але й заохочує обмін ідеями, розвиток проєктів, створення нових інноваційних продуктів.

Також Unity активно співпрацює з навчальними закладами та технічними школами, надаючи академічні ліцензії, методичні матеріали та сертифікати, що дозволяє інтегрувати навчання ігрової розробки в офіційні навчальні програми. Така підтримка дає змогу молодим розробникам починати роботу в середовищі Unity ще під час навчання у школах, технікумах або університетах.

У результаті, завдяки великій кількості доступного контенту, навчальних матеріалів, шаблонів та активній спільноті, Unity створює максимально сприятливі умови для самостійного навчання та професійного розвитку. Для початківців це означає можливість почати з нуля і досить швидко перейти до створення власних повноцінних ігрових проєктів, не потребуючи глибоких попередніх знань у програмуванні чи графіці.

Unity також підтримує систему пакетів (Package Manager), яка дає змогу встановлювати, оновлювати та керувати додатковими модулями, такими як Cinemachine (просунута камера), ProBuilder (інструменти для побудови

геометрії), Input System (нова система керування введенням) та багато інших. Такий підхід дає змогу зберігати ядро рушія компактним, а необхідні розширення підключати за потребою, що сприяє зручній структуризації проекту та економії ресурсів.

У рамках розробки 2D-платформерів особливо корисними є інструменти налаштування фізики, зокрема параметри гравітації, тертя та маси об'єктів. Unity надає можливість точно налаштовувати взаємодію між колайдерами (через Physics2D settings), що дозволяє створити реалістичну або, навпаки, стилізовану фізику, відповідно до вимог гри. Наприклад, шляхом коригування коефіцієнтів тертя й сили відскоку можна отримати бажану динаміку руху персонажа по платформах різної поверхні.

Ще однією важливою особливістю є підтримка візуального скриптингу (Visual Scripting, раніше Bolt), що надає змогу розробникам без глибоких знань програмування створювати логіку гри за допомогою блоків і візуальних діаграм. Хоча для складних проектів традиційне програмування на C# лишається пріоритетним, Visual Scripting може бути корисним для прототипування, тестування і швидкої побудови ігрових механік.

Завдяки наявності великої кількості шаблонів проєктів (2D, 3D, URP, HDRP) Unity дозволяє з самого початку налаштувати середовище відповідно до потреб розробника.

Unity також підтримує популярні бібліотеки й фреймворки, що використовуються в ігровій індустрії, зокрема Zenject для залежностей, DOTween для анімації, Cinemachine для камер, Firebase для аналітики й зберігання даних, та багато інших.

Таблиця 1.1

Порівняльна характеристика популярних ігрових рушіїв

Параметр	Unity	Unreal Engine	Godot
Основна мова прорамування	C#	C++/Blueprints	GDScript, C#
Підтримка 2D-ігор	Висока	Обмежена	Висока
Підтримка 3D-ігор	Висока	Висока	Середня
Графічні можливості	Хороші	Відмінні	Помірні
Крива навчання	Помірна	Висока	Низька
Доступність	Безкоштовна з умовами	Безкоштовна з умова	Повністю безкоштов
Розмір спільноти	Дуже велика	Велика	Менша, але активна
Підтримка мобільних платф	Так	Так	Так

1.3 Класифікація платформерів та їх ключові механіки

Платформери — це піджанр відеоігор, де основна мета полягає в переміщенні персонажа через рівні, використовуючи стрибки, біг та інші акробатичні дії для подолання перешкод і ворогів. З моменту свого виникнення жанр еволюціонував, породивши різноманітні піджанри та механіки.

Класифікація за вимірністю

- 2D платформери: Класичний тип, де рух обмежений двома координатами — горизонталлю та вертикаллю. Приклади: Super Mario Bros., Celeste.
- 2.5D платформери: Поєднують 3D-графіку з 2D-геймплеєм. Виглядають тривимірно, але ігровий процес лінійний. Приклади: Trine, Little Nightmares.

- 3D платформери: Дозволяють повноцінну навігацію в тривимірному просторі, потребують складніших механік камери та навігації. Приклади: Super Mario 64, Crash Bandicoot.[4]

Класифікація за стилем геймплею

- Класичні (arcade) платформери: Акцент на простих механіках стрибків та збору предметів. Гра зазвичай лінійна, з поступовим зростанням складності.
- Метроїдванії: Платформери з відкритим світом, де гравець досліджує карту, відкриває нові здібності для доступу до раніше недоступних зон. Приклади: Hollow Knight, Ori and the Blind Forest.
- Run-and-Gun (біг і стрільба): Комбінують платформер і стрілялку, з великою кількістю боїв. Приклади: Cuphead, Metal Slug.
- Фізичні платформери: Орієнтуються на фізику руху та об'єктів. Механіка часто пов'язана з гравітацією, тертям, інерцією. Приклади: Limbo, Inside.

Основні геймплейні механіки платформерів

Платформери базуються на певному наборі механік, які визначають характер гри. Найпоширеніші з них:

- Стрибки: Основна форма навігації, яка визначає складність рівнів.
- Подвійний стрибок: Дозволяє здійснити другий стрибок у повітрі — підвищує динаміку.
- Стіни для стрибків: Можливість відштовхування від стін, корисна для вертикальних рівнів.
- Платформи, що зникають: Тимчасові платформи, які ускладнюють таймінг рухів.
- Вороги та пастки: Створюють перешкоди, змушують гравця планувати дії.
- Збір предметів: Підвищує мотивацію дослідження та розвитку персонажа.
- Перемикачі/механізми: Створюють взаємодію з елементами оточення.
- Рівні зі зміною гравітації: Додають оригінальність та змінюють звичну навігацію.

Приклади успішних платформерів

- Super Mario Bros.: Класичний 2D платформер, який встановив стандарти жанру.
- Celeste: Сучасний 2D платформер з акцентом на точність рухів та емоційну історію.
- Hollow Knight: Метроїдванія з глибоким світом та складними боями.
- Cuphead: Run-and-Gun платформер з унікальним візуальним стилем та високою складністю.
- Limbo та Inside: Фізичні платформери з атмосферним дизайном та інноваційними механіками.[5]

Unity є потужним інструментом для створення 2D платформерів. Він надає розробникам можливості для реалізації різноманітних механік, таких як стрибки, подвійні стрибки, стрибки від стін та інші. Крім того, Unity підтримує використання фізичних рушіїв, що дозволяє створювати реалістичні рухи та взаємодії об'єктів.

1.4 Соціальні та економічні аспекти індустрії відеоігор

Індустрія відеоігор сьогодні є однією з найдинамічніших і найприбутковіших сфер розважальної індустрії, що має значний вплив на світову економіку та соціальну культуру. Відеоігри стали не просто способом розваги, а комплексним явищем, що охоплює різноманітні аспекти життя мільярдів людей.

Світовий ринок відеоігор демонструє стабільне зростання. За останні роки обсяги продажів ігор, обладнання та супутніх послуг досягли рекордних значень, а індустрія відеоігор за обсягом прибутків випереджає кінематограф та музику. Згідно з різними джерелами, у 2023 році ринок ігор сягнув понад 200 мільярдів доларів.

Монетизація ігор еволюціонувала: крім традиційної моделі продажу копій, з’явилися різні формати отримання доходів, серед яких:

- Мікротранзакції — внутрішньоігрові покупки косметичних предметів, ресурсів або ігрової валюти.
- Підписки — послуги, що дають доступ до бібліотек ігор або ексклюзивного контенту (наприклад, Xbox Game Pass, PlayStation Now).
- Доповнення (DLC) та розширення, які постійно підтримують інтерес гравців.
- Реклама — вбудована в безкоштовні ігри, що підтримуються рекламою.

Цей підхід створює постійний і стійкий до змін ринку грошовий потік, що стимулює подальші інвестиції та розвиток галузі.

Індустрія відеоігор також формує величезний ринок праці: від розробників програмного забезпечення, художників, сценаристів і звукорежисерів до маркетологів, менеджерів проектів, тестувальників і навіть професійних стрімерів та кіберспортсменів. В багатьох країнах ігрова індустрія стала важливою складовою національної економіки.

Крім того, ігри стимулюють розвиток суміжних галузей — виробництва комп'ютерного та ігрового обладнання, аксесуарів (геймпади, VR-шоломи), а також цифрових платформ (Steam, Epic Games Store, App Store, Google Play).

Відеоігри вже давно перестали бути лише дитячим захопленням. Сьогодні геймери — це люди різного віку, статі та соціального статусу. Інтерактивність ігор створює унікальні можливості для соціалізації та комунікації, що реалізується через:

- Онлайн-мультиплеєр ігри, де гравці об'єднуються у спільноти, беруть участь у командних заходах, що формує відчуття приналежності.
- Форумні спільноти, чати та платформи для обговорення, обміну досвідом і стратегій.
- Стрімінгові платформи (Twitch, YouTube Gaming), які дозволяють транслювати ігровий процес, об'єднуючи мільйони глядачів.

Одним із найбільш помітних соціальних феноменів стала популярність кіберспорту — професійних змагань у відеоігри. Кіберспорт має свою

інфраструктуру, спонсорів, трансляції на телебаченні та великі грошові призи. Для багатьох молодих людей він став не просто хобі, а професією.

Відеоігри також застосовуються у навчанні та тренуваннях: освітні ігри допомагають засвоювати матеріал через інтерактивний процес, симулятори — тренувати професійні навички (пілоти, медики, військові), а когнітивні ігри — розвивати пам'ять, увагу та логічне мислення.

1.5 Постановка задачі

Розробити програмний продукт — ігрову програму, яка реалізує наступні функціональні можливості:

1. Створення інтерактивного ігрового середовища з двовимірною графікою.
2. Реалізація основних ігрових механік (рух, стрибки, зіткнення, взаємодія з об'єктами).
3. Забезпечення візуалізації та анімації об'єктів у реальному часі.
4. Розробка системи керування персонажем за допомогою пристроїв введення.
5. Побудова рівнів з можливістю масштабування або додавання нових сцен.
6. Відтворення звукових ефектів або музичного супроводу в ігровому процесі.
7. Оптимізація продуктивності та стабільності роботи програми на різних пристроях.
8. Формування умов завершення гри, підрахунок балів або інші елементи гейміфікації.

1.6 Висновок до першого розділу

Перший розділ дослідження був присвячений комплексному аналізу відеоігор як культурного, соціального та економічного феномену сучасності. Відеоігрова індустрія за останні десятиліття пройшла шлях від нішового

захоплення до одного з найпотужніших і найдинамічніших секторів цифрової економіки. Її стрімкий розвиток супроводжується глибокими змінами в суспільстві, культурі споживання, формуванні нових професій і стилів життя, а також відкриває нові горизонти у сфері технологічного прогресу.

У ході розгляду було з'ясовано, що відеоігри відіграють дедалі важливішу роль у глобальному інформаційному середовищі, не лише як засіб розваги, а й як інструмент освіти, соціалізації, культурного самовираження та комунікації. З огляду на все більшу інтеграцію цифрових технологій у повсякденне життя, відеоігри перетворюються на потужний засіб впливу на масову свідомість, естетичні уподобання та навіть політичні погляди молодого покоління.

Окрему увагу було приділено аналізу технологічних основ сучасного ігрового виробництва. У цьому контексті значну роль відіграють універсальні інструменти розробки, які дають змогу створювати інтерактивні середовища, оптимізовані для різноманітних платформ і пристроїв. Саме завдяки таким інструментам розробка ігор стала доступною для широкого кола ентузіастів, інді-команд і стартапів, які можуть конкурувати з великими студіями, пропонуючи інноваційні концепції та свіжий ігровий досвід.

Крім того, соціально-економічні аспекти індустрії демонструють її глибокий вплив на глобальну економіку. Зростання ринку праці, нові форми зайнятості, включення до економіки креативних індустрій, формування нових бізнес-моделей — усе це свідчить про стратегічну важливість ігрового сектора. Не менш значущим є й вплив відеоігор на соціальні структури — формування онлайн-спільнот, розвиток кіберспорту, нові форми соціальної інтеракції та колективної творчості.

У результаті опрацювання матеріалів першого розділу можна зробити висновок, що відеоігри сьогодні — це багатогранне явище, що поєднує в собі технології, мистецтво, бізнес і соціальні практики. Усвідомлення цього дозволяє краще зрозуміти не лише технічну сторону створення ігор, але й ті глибинні зміни, які відбуваються у світі під впливом цифрової культури. Ці аспекти

стануть основою для подальшого практичного розгляду специфіки розробки ігрового проекту, що буде здійснено в наступних розділах.

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ІГРОВОГО ПРОЕКТУ

2.1 Вибір програмних засобів для реалізації проекту

Вибір програмного забезпечення для створення відеогри є одним із ключових етапів, що впливає на ефективність розробки, зручність підтримки коду, а також можливість масштабування проекту. У сучасній практиці геймдизайну та програмної інженерії активно застосовуються спеціалізовані інструменти, які значно спрощують реалізацію як 2D, так і 3D ігор.

Платформи для створення ігор (ігрові рушії) надають засоби для управління графікою, фізикою, анімаціями, аудіо та логікою поведінки об'єктів. Серед найпопулярніших рушіїв, які використовуються в розробці ігор, можна виділити такі як Unity, Unreal Engine, Godot, GameMaker Studio, Defold та інші. Кожен із них має свої особливості, переваги та обмеження залежно від типу гри, цільової платформи та досвіду розробника.

Наприклад, рушій Unity, який підтримує C# як основну мову програмування, широко використовується для створення мобільних, інді та VR/AR ігор. Його основні переваги — це багатоплатформеність, потужний редактор сцен, підтримка двовимірної та тривимірної графіки, велика спільнота розробників та доступ до тисяч безкоштовних і платних плагінів через Unity Asset Store. Інший рушій — Godot — є повністю безкоштовним і відкритим, підтримує мови програмування GDScript, C# та C++ і з кожною версією набуває все більшої популярності серед інді-розробників. Вибір між рушіями залежить від конкретних потреб: для 2D платформерів часто перевага надається рушіям, які мають зручну інтеграцію зі спрайтами, тайлмапами, колізіями та анімацією.

Вибір рушія повинен враховувати також технічні вимоги проекту, наприклад, підтримку мобільних платформ (Android, iOS), інтеграцію з базами даних, підтримку мережевої взаємодії, зручність розгортання проекту на маркетах, таких як Google Play чи App Store. Крім того, важливим критерієм є

наявність актуальної документації та активної спільноти користувачів, що значно полегшує процес навчання та вирішення технічних проблем [6].

Для роботи з візуальними ресурсами (спрайтами, фонами, анімаціями) зазвичай використовуються інструменти на кшталт Adobe Photoshop, Aseprite, Krita, які дають змогу створювати та редагувати піксельну та векторну графіку. Для створення звукових ефектів і музики можуть застосовуватись Audacity, FL Studio або інші DAW-застосунки. Для написання коду часто використовують середовища розробки Visual Studio, Visual Studio Code, JetBrains Rider тощо.

У сучасному процесі розробки ігор також важливою є система контролю версій. Найпоширенішою технологією є Git, яка дає змогу вести колективну розробку, фіксувати зміни, відновлювати попередні версії коду та вирішувати конфлікти. Платформи GitHub, GitLab або Bitbucket дозволяють зручно управляти репозиторіями, створювати задачі та документи, здійснювати рев'ю коду тощо.

Таким чином, для реалізації сучасного 2D-проєкту, зокрема платформера, доцільно використовувати ігровий рушій, який має зручний інтерфейс, підтримує актуальні стандарти, має великий вибір додаткових бібліотек і підтримку з боку спільноти.

2.2 Вимоги до програмного забезпечення

Для реалізації ігрового проекту було висунуто низку вимог до програмного забезпечення, які охоплюють як функціональні, так і нефункціональні аспекти. Основною метою було забезпечити стабільне середовище розробки, що дозволяє створювати інтерактивну 2D-гру з можливістю подальшого масштабування та адаптації під мобільні платформи.

Функціональні вимоги:

- Можливість побудови ігрової сцени з двовимірними об'єктами;
- Підтримка фізики, анімацій та системи частинок;

- Реалізація системи керування персонажем, обробки зіткнень та логіки геймплею;
- Зручна інтеграція графічних та звукових ресурсів;
- Підтримка різних типів платформ (Android, Windows).

Нефункціональні вимоги:

- Зручний і стабільний інтерфейс середовища розробки
- Широка спільнота користувачів та доступність документації
- Підтримка розширень та плагінів для розширення функціональності
- Наявність інтегрованого відлагоджувача та засобів для профілювання продуктивності

Для реалізації проекту було обрано Unity — один з найпоширеніших ігрових рушіїв, який має потужні засоби для розробки 2D ігор, підтримує C# як основну мову скриптів та дозволяє швидко створювати прототипи і повноцінні ігрові продукти. Його переваги включають візуальний редактор сцен, систему компонентів та велику кількість навчальних матеріалів [7].

На рис. 2.1 зображено інтерфейс редактора Unity, який активно використовувався під час розробки гри. Unity надає інструменти для роботи зі спрайтами, колайдерами, rigidbody-компонентами, а також підтримку подій та систем анімації.

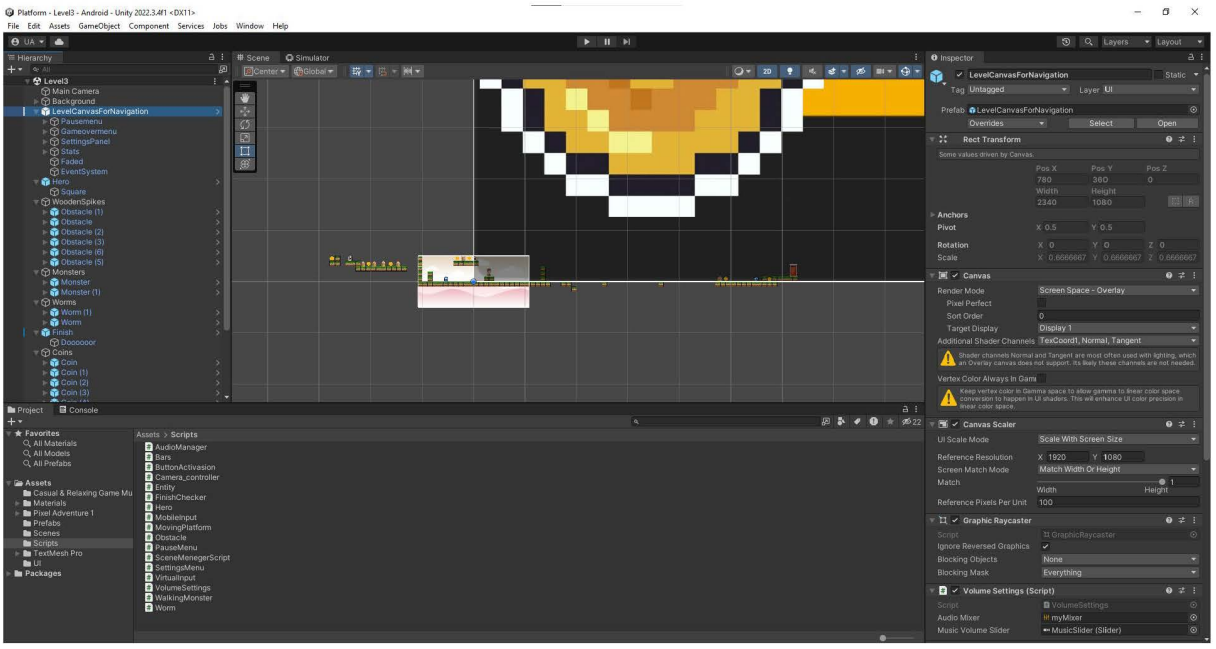

Рисунок 2.1 – Інтерфейс редактора Unity

Допоміжним інструментом для написання скриптів виступає Visual Studio, яка інтегрується з Unity через плагіни. Це середовище дозволяє писати, тестувати та відлагоджувати код на C# з автозаповненням, підсвічуванням синтаксису та підтримкою Git. На рис. 2.2 зображено інтерфейс Visual Studio з прикладом реалізації логіки керування персонажем.

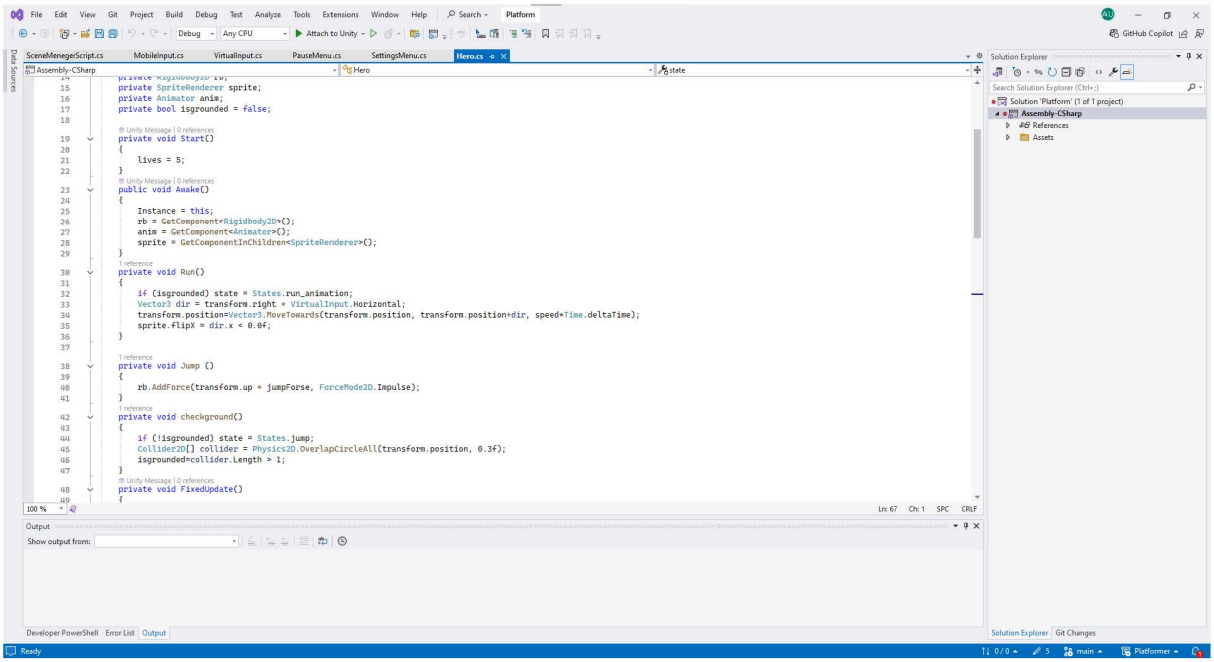

Рисунок 2.2 – Інтерфейс Visual Studio

Крім того, для збереження проекту та організації командної роботи було використано систему контролю версій Git разом з платформою GitHub, що дозволило:

- вести історію змін файлів;
- повертатися до попередніх версій проєкту у разі потреби;
- забезпечити резервне копіювання коду;
- використовувати гілки для розробки окремих модулів гри без ризику вплинути на основну версію.

GitHub став зручним середовищем для керування версіями, особливо під час активної фази розробки, коли зміни вносилися щодня.

Таким чином, обрана зв'язка інструментів — Unity + Visual Studio + GitHub — повністю задовольняє вимоги проєкту та дозволяє реалізувати задуману гру відповідно до сучасних стандартів ігрової індустрії.

2.3 Засоби для реалізації клієнтської частини гри

Клієнтська частина гри є інтерфейсом між користувачем і внутрішньою логікою застосунку. Вона охоплює візуальні компоненти, засоби взаємодії (UI/UX), системи введення, анімації, аудіовідтворення та інші елементи, що забезпечують інтерактивність і сприйняття гри. Реалізація цієї частини має вирішальне значення для користувацького досвіду.

Одним із ключових засобів для побудови клієнтської частини в 2D-іграх є Canvas-система в Unity, яка дозволяє створювати адаптивні інтерфейси, що підлаштовуються під різні роздільні здатності пристроїв [8].

Переваги Canvas-системи:

- Підтримка масштабування для різних екранів
- Інтеграція з іншими UI-компонентами
- Можливість створення складної ієрархії елементів

Ключові елементи: Canvas, RectTransform, UI елементи (Button, Image, Text). Крім цього, Unity UI Toolkit — новіша система, що поєднує декларативний підхід до побудови UI (на основі UXML і USS) та значно розширює можливості кастомізації й повторного використання елементів [9].

Переваги UI Toolkit:

- Створення UI через розмітку
- Поділ логіки і стилів
- Підтримка кастомних компонентів

Для керування в іграх все частіше застосовується Unity Input System, яка дозволяє абстрагувати пристрої введення (клавіатура, миша, джойстик, тачскрін тощо) через єдину конфігурацію [10].

Переваги Input System:

- Кросплатформенність
- Підтримка кількох схем керування
- Зручне редагування через UI

Пристрої, що підтримуються:

- Клавіатура
- Геймпад
- Тачскрін
- Миша

Анімація також є важливою частиною клієнтської складової. Animator Controller та система State Machines дозволяють реалізовувати плавні переходи між станами персонажів або об'єктів. Unity підтримує 2D Animation Package, що включає інструменти для скелетної анімації та Sprite Rigging [11].

Компоненти анімації:

- Animator
- Animation Clip
- State Machine

Переваги 2D Animation Package включають скелетну анімацію, прив'язку до кісток (bone rigging) та динамічне керування позами.

Для озвучення подій та атмосфери використовуються засоби AudioSource та AudioMixer, які дозволяють детально налаштовувати звукові ефекти, гучність і просторове звучання (див. табл. 2.1).

Таблиця 2.1

Основні компоненти звуку в Unity

Компонент	Призначення

AudioSource	Компонент, який додається до об'єкта сцени та відповідає за відтворення аудіо. Може налаштовуватись для просторового звучання, регулювання гучності, циклічного програвання тощо.
AudioClip	Представляє окремий аудіофайл (наприклад, .wav або .mp3), який використовується в AudioSource для програвання.
AudioMixer	Центральний інструмент для мікшування аудіо, що дозволяє групувати кілька джерел звуку, застосовувати ефекти, регулювати гучність окремих каналів і створювати динамічні зміни звучання під час гри.

2.4 Засоби для реалізації внутрішньої логіки гри

Реалізація внутрішньої логіки є ключовим етапом у створенні 2D платформера, адже саме вона визначає поведінку об’єктів, взаємодію між ними, а також забезпечує контроль за ігровим процесом. У середовищі Unity основою побудови логіки є компоненти MonoBehaviour, які дозволяють створювати скрипти для різноманітних об'єктів сцени. Ці скрипти можуть контролювати пересування персонажів, обробку взаємодій, зміни станів, оновлення ігрових параметрів у реальному часі.

Unity також надає розвинену фізичну систему на базі рушія Box2D, яка дозволяє реалізовувати реалістичну поведінку об’єктів на сцені. За допомогою таких компонентів, як Rigidbody2D та Collider2D, можна моделювати гравітацію, зіткнення, інерцію тощо. Це особливо важливо у платформерах, де точність взаємодії з платформами, перешкодами та ворогами має вирішальне значення для ігрового досвіду [12].

Важливою складовою внутрішньої логіки є користувацькі (кастомні) компоненти. Вони розробляються для окремих функціональностей, таких як

керування рухом персонажа, штучний інтелект ворогів або обробка системи здоров'я. Кожен компонент виконує чітко визначену роль і відповідає за окремий аспект логіки. Наприклад, один скрипт може відповідати лише за пересування персонажа, інший — за його здатність завдавати шкоди, ще інший — за логіку ворожих об'єктів.

Також часто використовуються сервісні класи — окремі об'єкти, які забезпечують спільний функціонал (наприклад, керування інвентарем, обробку даних користувача, або логіку збережень гри). Вони можуть впроваджуватись через шаблон Singleton або керуватись системою залежностей (Dependency Injection), що підвищує гнучкість архітектури.

Внутрішня логіка гри також охоплює керування анімацією персонажів, яка в Unity реалізується за допомогою системи Animator та Animation Controller. Ці інструменти дозволяють створювати плавні переходи між різними станами — наприклад, ходьбою, стрибком, атакою або смертю персонажа. Логіка переходів між анімаційними станами часто залежить від параметрів, які змінюються під час гри, що дозволяє динамічно реагувати на дії гравця.

Окрему увагу слід приділити звуковому супроводу. Unity має вбудовану систему для відтворення звукових ефектів та музики, що використовує компоненти AudioSource та AudioClip. Вони дозволяють керувати звуками у відповідь на події в грі, зокрема відтворенням звуків стрибка, пошкоджень, перемог або фонової музики. Це підвищує емоційне занурення гравця та покращує загальне враження від гри.

Для побудови візуального середовища гри активно використовується система Tilemap, яка дозволяє швидко створювати рівні на основі набору тайлів. Вона підтримує як прямокутні, так і ізометричні сітки, а також інтегрується з системою колізій. Такий підхід не тільки прискорює процес розробки, але й забезпечує високу продуктивність навіть на мобільних пристроях, що є важливим критерієм для 2D-проектів.

Крім цього, важливу роль у розробці внутрішньої логіки відіграє структура коду. У сучасній практиці рекомендується дотримуватись принципів SOLID, що

дозволяє зробити код гнучким, розширюваним та стійким до змін. Особливо актуальним є принцип розділення обов'язків (Single Responsibility Principle), який передбачає, що кожен клас має відповідати лише за одну частину функціональності [13].

Усі ці інструменти об'єднуються в цілісну архітектурну систему, де різні компоненти взаємодіють між собою через чітко визначені зв'язки. У проекті застосовується шаблон розділення відповідальностей (Separation of Concerns), що дозволяє підтримувати порядок у кодовій базі, легко масштабувати проєкт та ефективно тестувати окремі модулі.

До ключових елементів внутрішньої логіки 2D платформера можна віднести:

- контроль фізики об'єктів (гравітація, колізії);
- обробку користувацьких дій (рух, атака, стрибки);
- взаємодію з ворогами та оточенням;
- анімацію та звуковий супровід;
- генерацію та оновлення рівнів за допомогою Tilemap;
- підтримку чистої та модульної архітектури коду.

2.5 Висновки до другого розділу

У даному розділі було проведено аналіз інструментів та технологій, які доцільно застосовувати для реалізації 2D ігрового проекту, зокрема платформера. Розглянуто низку популярних ігрових рушіїв, серед яких обґрунтовано було обрано Unity як найбільш придатний для поставлених цілей. До основних причин вибору належать: підтримка двовимірної графіки, широкий спектр інструментів для створення та редагування ігрових сцен, гнучка система компонування об'єктів, багата екосистема плагінів та активна спільнота розробників.

На основі аналізу було сформульовано ключові функціональні та нефункціональні вимоги до програмного забезпечення. Вони стали основою для подальшого вибору інструментів, що охоплюють не лише рушій, а й

середовище для написання коду (Visual Studio), систему контролю версій (Git + GitHub), а також додаткові компоненти для роботи зі звуком, графікою та UI.

Окрему увагу було приділено засобам для реалізації клієнтської частини гри, що визначає взаємодію користувача з програмою. Було обрано сучасні та гнучкі технології, Вони дозволяють створити адаптивний, зручний та функціональний інтерфейс користувача, який може працювати на різних пристроях і роздільних здатностях.

Загалом, на основі аналізу інструментів та технологій, зроблено обґрунтований вибір програмного забезпечення, який повністю відповідає потребам реалізації 2D платформера та забезпечує можливість ефективної розробки, масштабування та подальшої підтримки ігрового проекту.

РОЗДІЛ 3. РОЗРОБКА 2D ПЛАТФОРМЕРА НА UNITY

3.1 Актуальність розробки

У сучасному світі ігрова індустрія продовжує демонструвати стрімкий розвиток та зростання, охоплюючи дедалі ширшу аудиторію користувачів різного віку. Одним із найпопулярніших жанрів цифрових ігор залишаються платформери — двовимірні аркадні ігри, у яких користувач керує персонажем, що переміщується рівнями з перешкодами, збирає об'єкти та долає ворогів. Переваги жанру полягають у простоті освоєння геймплею, динамічності дій та можливості охопити широку цільову аудиторію.

Особливої актуальності жанр набуває у сфері мобільної розробки, де важливо забезпечити короткі та захопливі ігрові сесії. Згідно з аналітичними звітами, платформери входять до числа найпопулярніших жанрів серед мобільних та інді-ігор. За даними платформи GameAnalytics, серед найбільш завантажуваних мобільних ігор 2023 року платформери стабільно посідають провідні позиції, демонструючи високі показники утримання гравців та монетизації [14]. Багато провідних компаній, зокрема Nintendo, King та Voodoo, випустили низку успішних 2D-платформерів, що здобули велику популярність серед гравців. На рисунках 3.1 і 3.2 зображено приклади подібних ігор, які демонструють типові візуальні та геймплейні особливості жанру.

Розробка власної гри у жанрі 2D-платформера дозволяє здобути практичні навички роботи з сучасними ігровими рушіями, зокрема Unity, що є одним з найпоширеніших інструментів для створення як інді, так і комерційних проектів. Під час реалізації даного проекту основними завданнями були: ознайомлення з базовими та просунутими можливостями Unity, проектування внутрішньої логіки гри, реалізація основних ігрових

механік, адаптація інтерфейсу до мобільних пристроїв, а також практичне закріплення знань із програмування в середовищі C#.

Таким чином, обрана тема є не лише цікавою з точки зору реалізації ігрового задуму, а й сприяє набуттю компетентностей, необхідних для подальшого професійного розвитку в IT-сфері, зокрема в напрямі розробки ігор.

Рисунок 3.1 – Приклад мобільного платформера Geometry Dash

Рисунок 3.2 – Приклад мобільного платформера Super Mario Bros

3.2 Структура проекту

Організація структури проекту є надзвичайно важливою під час створення будь-якої гри, зокрема 2D платформера. Грамотно побудована структура дозволяє ефективніше масштабувати проект, полегшує роботу над розширенням функціональності, покращує читабельність коду та пришвидшує навігацію серед файлів. Це особливо актуально при командній роботі або в довгострокових проектах, де кількість ресурсів і скриптів постійно зростає. Як зазначено в [15], впорядкована проектна структура зменшує ризик виникнення помилок та прискорює процес розробки.

У проєкті, реалізованому в середовищі Unity, структура організована з урахуванням логіки поділу відповідальностей та подальшої масштабованості. Основні папки виглядають наступним чином:

- Animations, Coin, Doors, Hero, WalkingMonster, Worm — розміщують графіку, анімації та пов’язані ресурси для окремих об’єктів гри.
- Level Design — призначена для збереження елементів побудови рівнів.
- Music — звукове оформлення.
- Scenes — сцени гри: меню, рівні тощо.
- Prefabs — шаблони об'єктів, які використовуються багаторазово.
- UI — графічні інтерфейси користувача.
- Scripts — одна з ключових папок, що містить скрипти з логікою гри.

Папка Scripts відіграє критичну роль, оскільки саме в ній розміщуються всі основні сценарії. У ній можна виділити кілька груп за функціональністю:

- Скрипти персонажа: Hero, MobileInput, Camera_controller.
- Скрипти об’єктів і логіки рівня: MovingPlatform, Obstacle, Entity, FinishChecker, ButtonActivation.
- Меню та управління сценами: PauseMenu, SceneMenegerScript, SettingsMenu, VolumeSettings.
- Аудіо та інтерфейс: AudioManager, Bars, VirtualInput.

Усі названі компоненти є частинами більшої архітектури, яка дозволяє легко розширювати функціональність без порушення існуючого коду. Такий

підхід є добрим прикладом дотримання принципу Single Responsibility, коли кожен скрипт відповідає лише за одну функцію.

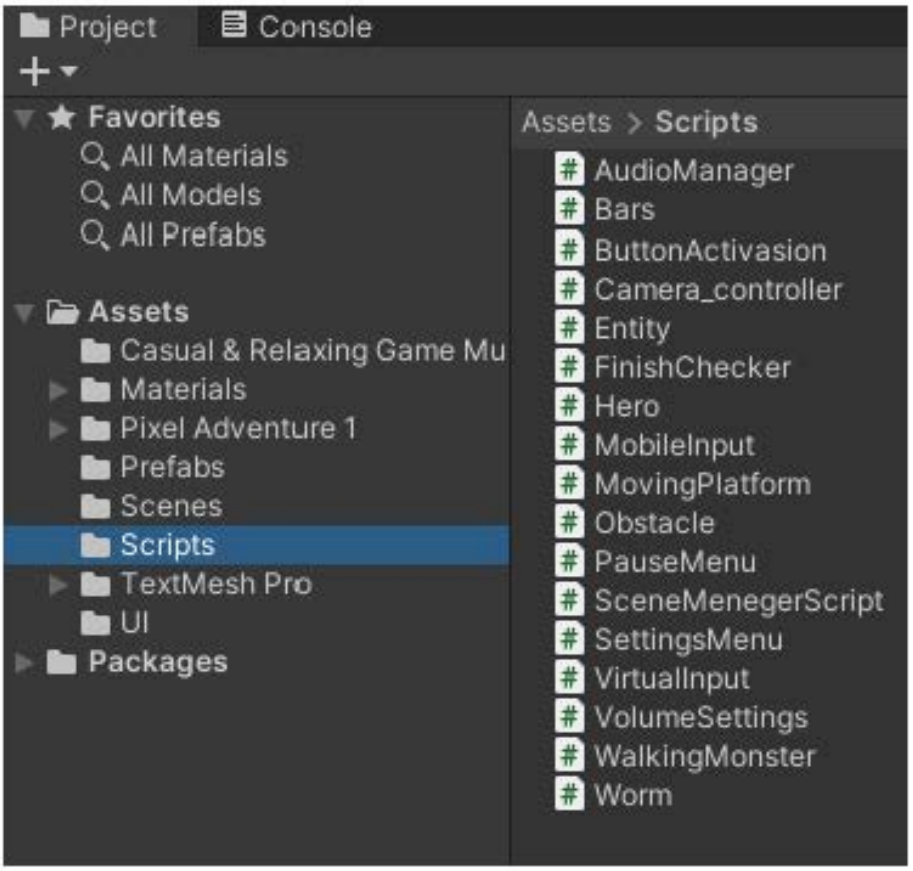

Рисунок 3.3 – Загальна файлова структура проекту та папки Scripts

3.3 Реалізація ігрової логіки

Розробка ігрової логіки є центральною частиною будь-якого ігрового проекту, оскільки саме вона визначає поведінку всіх елементів гри, взаємодію між ними та забезпечує інтерактивність, що є основою ігрового процесу. У межах даного проекту створення 2D-платформера на базі Unity стало чудовою можливістю застосувати здобуті знання, отримати новий практичний досвід і глибше ознайомитися з особливостями розробки ігор у сучасному середовищі. Нижче детально описано етапи реалізації ключових аспектів гри.

Управління персонажем

Головний персонаж — це центральна фігура гри, і саме його взаємодія з довкіллям формує основну частину геймплею. Для реалізації управління було

створено окремий скрипт, у якому обробляються натискання клавіш (у ПК-версії) або сигнали з віртуальних кнопок (у мобільній версії). Обробка введення реалізована з використанням стандартних засобів Unity (Input.GetKey, Input.GetButtonDown тощо), а рух персонажа — за допомогою фізики через компонент Rigidbody2D, який дозволяє досягти плавності та природності рухів. Як видно на рисунку 3.4, структура скрипта передбачає чітке розділення логіки на частини: переміщення, стрибки, перевірка на землю тощо.

Варто зазначити, що правильна реалізація управління персонажем значно впливає на відчуття від гри. За даними Unity Learn [16], плавність управління та швидкий відгук — одні з головних факторів, що впливають на користувацьке враження у платформерах.

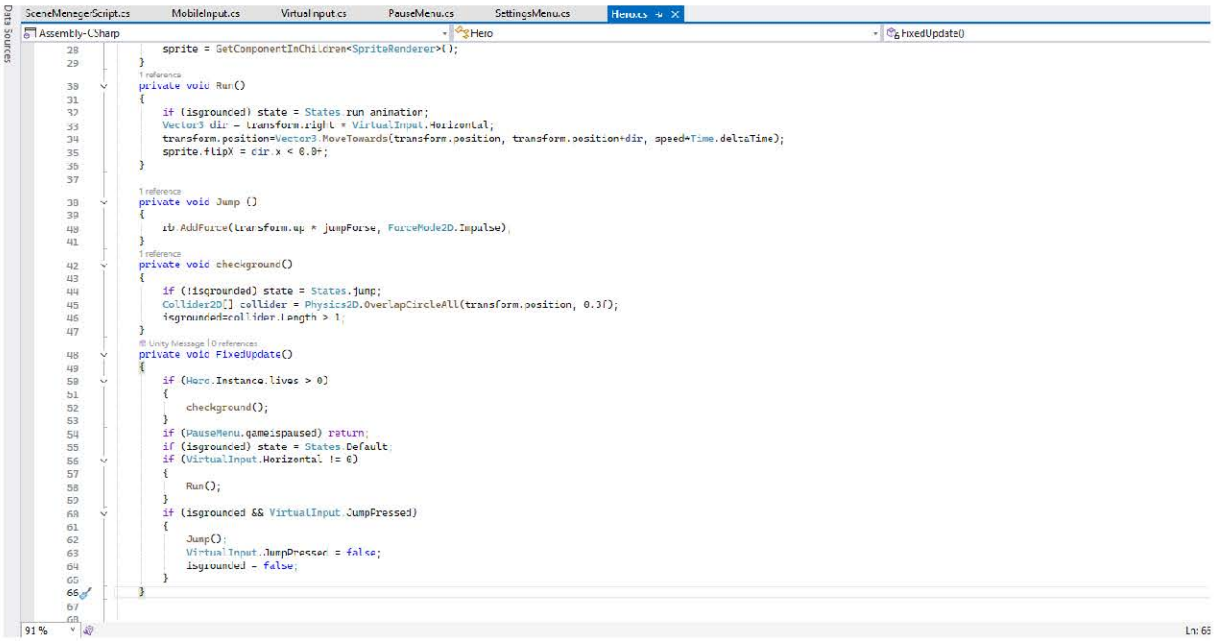

Рисунок 3.4 – Структура скрипта Hero.cs

Обробка колізій та стрибки

У платформерах критично важливою є точна й надійна система колізій. Щоб запобігти помилковим стрибкам або "залипанню" на поверхнях, у грі реалізовано перевірку наявності землі за допомогою Physics2D.OverlapCircle. Це дозволяє переконатись, що персонаж торкається підлоги, перш ніж дозволити стрибок. Система тегів дозволяє відрізняти об’єкти середовища: платформи,

небезпеки, цілі тощо. Наприклад, контакт із шипами або ворогами призводить до перезапуску сцени чи смерті персонажа.

На рисунку 3.5 представлено логіку, яка відповідає за обробку таких взаємодій. Згідно з документацією Unity Manual [17] , правильне використання колайдерів дозволяє суттєво покращити продуктивність і точність ігрової фізики.

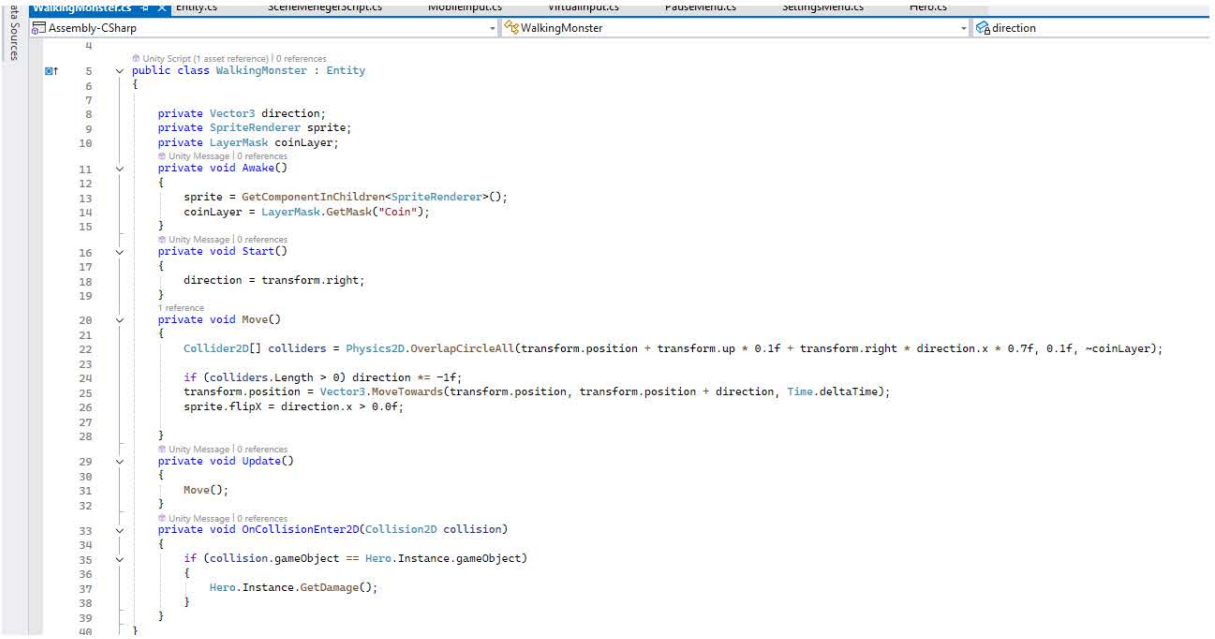

Рисунок 3.5 – Структура скрипта WalkingMonster.cs

Анімації персонажа

Щоб забезпечити плавні переходи між станами персонажа (наприклад, біг, стрибок, падіння), використано Animator — потужний інструмент Unity для керування анімаціями. У проекті створено окрему станову машину, яка містить усі основні стани персонажа. Переходи між ними залежать від змін параметрів, які задаються скриптами залежно від дій гравця.

На рисунку 3.6 зображено загальну структуру анімаційної машини. Завдяки правильно налаштованим умовам переходу, анімації відбуваються плавно та відповідають дійсності. Чітко організована система станів допомагає уникнути конфліктів між анімаціями та забезпечує надійність логіки.

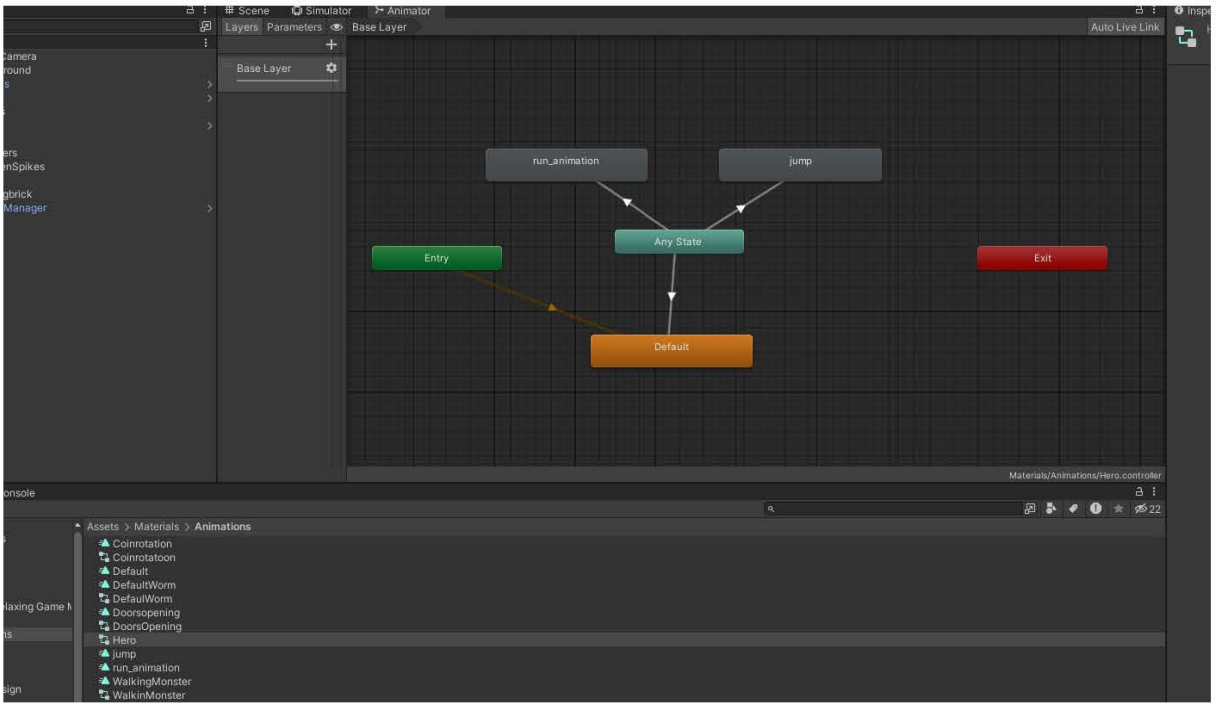

Рисунок 3.6 – Вигляд машини переходів анімацій у Unity

Вороги та NPC

Створення ворожих персонажів було реалізоване шляхом створення скриптів для кожного типу ворога. Наприклад, один з ворогів — "Walking Monster" — пересувається по заданій траєкторії, змінюючи напрямок при зіткненні з колайдером. Кожен з ворогів має свої анімації та власну логіку смерті.

Як видно на рисунку 3.7, монстр має межі руху та використовує колайдери для коректного відпрацювання своєї логіки. Якісна реалізація NPC дозволяє створювати виклик для гравця, що сприяє залученню. Відповідно до GDC-трендів [18], гнучкість ворогів значно підвищує реіграбельність гри.

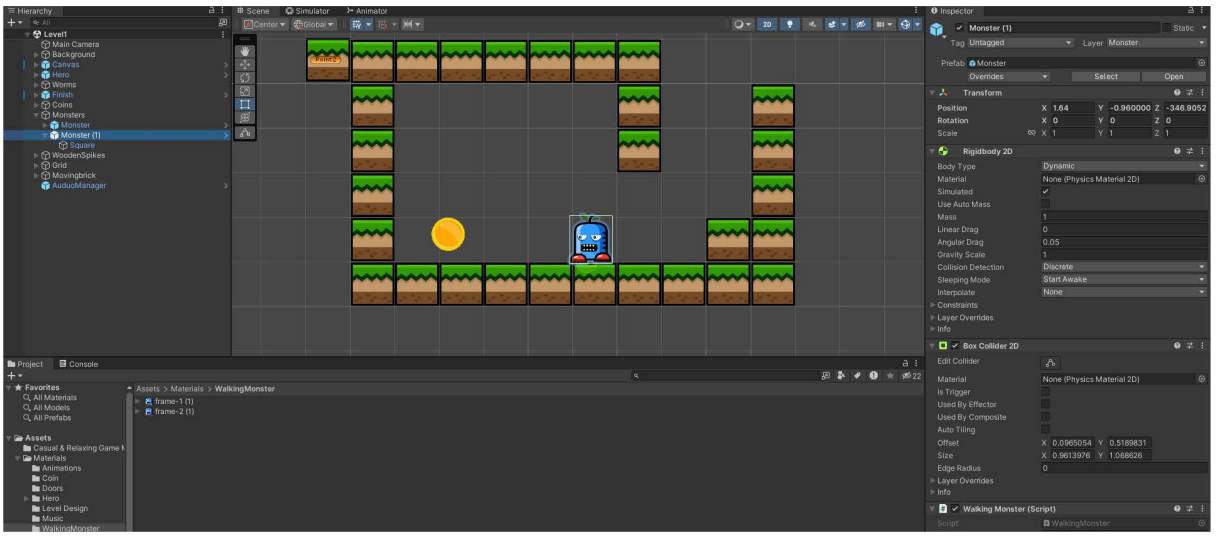

Рисунок 3.7 – Приклад одного з монстрів який охороняє нагороду

Перехід між сценами

У грі передбачено перехід між рівнями, меню та іншими сценами. Це реалізовано за допомогою стандартного API Unity — SceneManager.LoadScene. У місцях завершення рівня розташовано об'єкт-тригер, взаємодія з яким активує перехід до нової сцени. Цей об'єкт наочно показано на рисунку 3.8.

Ця реалізація відповідає сучасним стандартам і дозволяє легко змінювати або додавати нові рівні до гри. Згідно з офіційним посібником Unity, SceneManager є базовим способом навігації в багатосценових проектах.

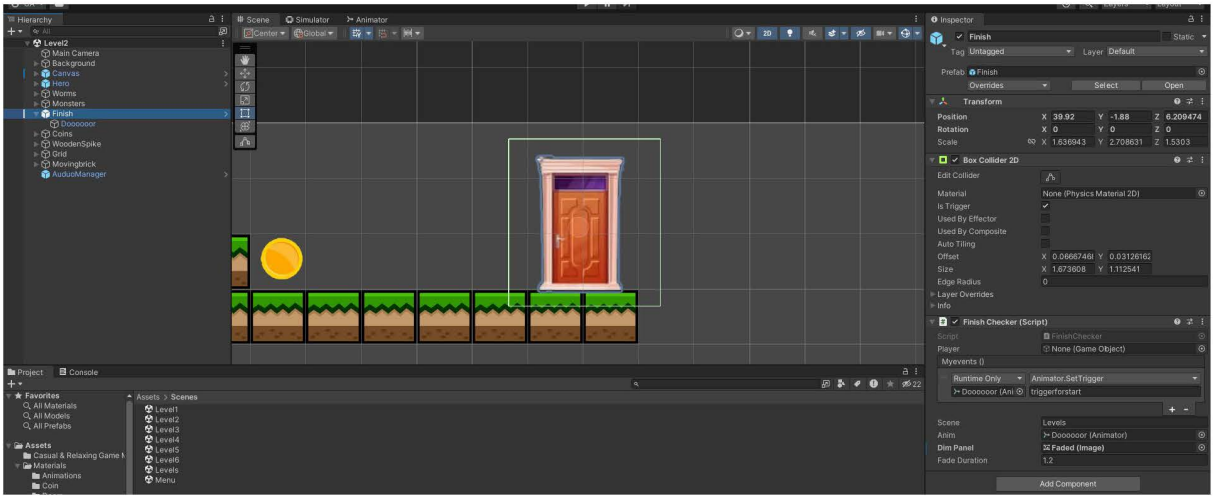

Рисунок 3.8 – Вигляд переходу на інший рівень та його логіки роботи

Префаби в Unity — один із найефективніших інструментів для повторного використання об’єктів. У проекті було створено низку префабів: монети, вороги,

платформи, кнопки керування тощо. Це дозволяє створити об’єкт один раз і потім використовувати його на всіх рівнях гри без необхідності повторного налаштування.

На рисунку 3.9 представлено приклад префабу монети. Усі параметри цього об’єкта (колайдер, анімація, скрипт підбору) збережено та легко масштабуються. Правильне використання префабів не лише покращує оптимізацію, але й значно прискорює розробку.

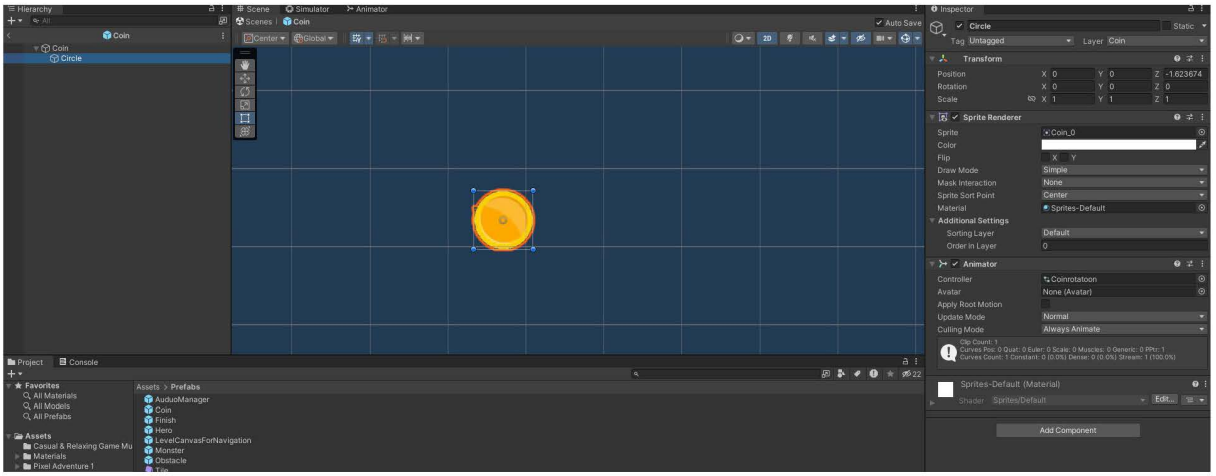

Рисунок 3.9 – Вигляд префабу монети

UI та меню

Користувацький інтерфейс відіграє важливу роль у взаємодії гравця з грою. Меню, HUD, кнопки керування — усе це має бути інтуїтивно зрозумілим і доступним. У грі реалізовано головне меню, меню паузи, а також внутрішньоігровий HUD, який відображає, наприклад, кількість зібраних монет.

На рисунку 3.10 показано структуру меню, реалізованого за допомогою Canvas. Усі кнопки мають прив’язані методи, які викликають потрібні функції (вихід з гри, зміна гучності, перехід до іншої сцени). Як зазначено у UI Toolkit Unity, правильне проектування UI підвищує якість взаємодії та сприяє зануренню в гру.

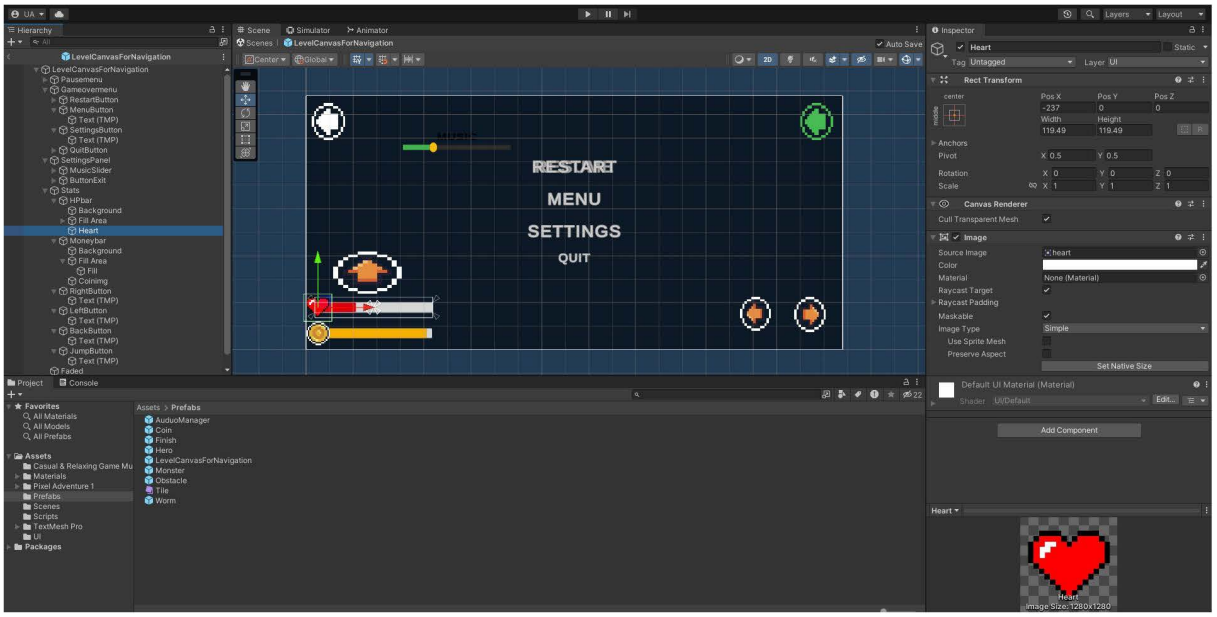

Рисунок 3.10 – Вигляд префабу меню

Звуковий супровід

Для створення приємної ігрової атмосфери у проекті використано один фоновий саундтрек, який безперервно повторюється під час гри. Його відтворення забезпечується за допомогою централізованого скрипта AudioManager, що дозволяє зручно керувати гучністю звуку. Гравець має можливість змінювати рівень гучності фонової музики через меню налаштувань, а обране значення зберігається між ігровими рівнями. Якісний звуковий супровід сприяє створенню приємного ігрового досвіду та позитивно впливає на загальне враження від гри.

З огляду на те, що гра орієнтована на мобільні пристрої, реалізація зручного сенсорного керування була ключовою. Було створено віртуальні кнопки переміщення та стрибка, які викликають відповідні методи у контролері персонажа. Таке рішення забезпечує комфортну гру на смартфонах з різним розміром екрану. Це відповідає загальним принципам мобільної оптимізації щодо важливості адаптації керування для сенсорних пристроїв.

Для забезпечення порядку та ефективності в коді проекту було створено окремі менеджери: SceneManagerScript, VolumeSettings, UIManager тощо. Вони

відповідають за конкретні функції, що дозволяє уникнути дублювання коду та забезпечити гнучкість у подальшій розробці.

На рисунку 3.11 можна побачити приклад логіки менеджера сцен, який централізовано відповідає за навігацію між усіма частинами гри. Усі скрипти розміщені в директорії Scripts, Як зазначено у Unity Architecture Guidelines [19], підтримка чистої структури — запорука ефективної командної роботи та масштабування проекту.

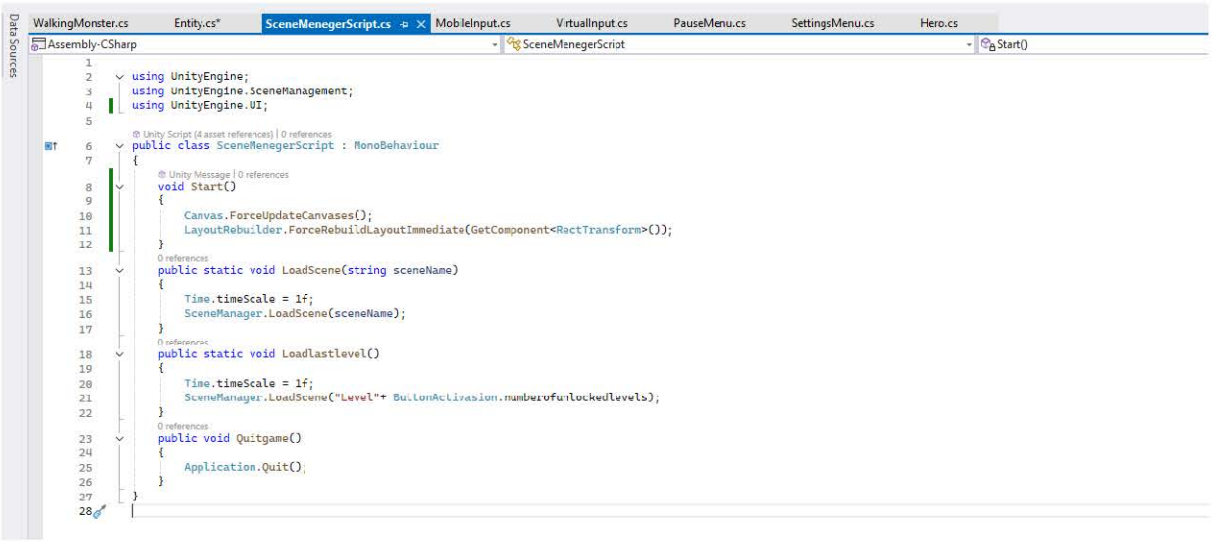

Рисунок 3.11 – Вигляд скрипта SceneMenegerScript.cs

У результаті вдалося створити гнучку, стабільну архітектуру гри, яка відповідає сучасним вимогам до мобільних платформерів, легко піддається розширенню й підтримці, а також демонструє високий рівень технічного та візуального виконання.

3.4 Тестування та налагодження гри

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, зокрема — ігор. Саме на цьому етапі перевіряється працездатність усіх реалізованих функцій, виявляються помилки, а також забезпечується стабільна та передбачувана поведінка програми для користувача. Без ретельного

тестування навіть якісно написана логіка може працювати некоректно в реальних умовах.

Після завершення реалізації основної ігрової логіки було проведено етап тестування та налагодження гри з метою виявлення помилок та підвищення стабільності роботи всіх компонентів проєкту.

Ручне тестування

Було виконано ручне тестування ігрового процесу безпосередньо в редакторі Unity. Перевірялася коректність:

- руху персонажа;
- анімацій;
- колізій з ігровими об'єктами;
- збирання предметів;
- перемикань між сценами;
- роботи UI-елементів.

У результаті тестування було виявлено такі помилки:

- Персонаж застрягав у стінах під час стрибка при дотику до краю платформи.
- Колектібли (збиральні предмети) не зникали після взаємодії з ними.
- UI-лічильник не оновлювався при збиранні предметів.
- Анімації в деяких ситуаціях відображались некоректно (наприклад, біг виконувався під час стрибка).

Виявлені помилки було виправлено шляхом:

- вдосконалення колізійної логіки та зміни параметрів Rigidbody2D;
- додавання зникнення об'єкта після збирання за допомогою Destroy(gameObject);
- налаштування оновлення UI за допомогою подій;
- корекції параметрів Animator та переходів між станами.

Використання інструментів налагодження

Для спрощення виявлення проблем було активно використано:

- Debug.Log для виведення внутрішніх значень змінних у консоль;

- Gizmos для візуалізації зон дії об’єктів (наприклад, радіус взаємодії з колектіблами);
- Profiler — для перевірки продуктивності під час гри, зокрема використання CPU та пам’яті;
- Console — для перегляду помилок і попереджень у реальному часі.

На рисунку 3.9 можна побачити приклад виводу повідомлень у консоль за допомогою Debug.Log, який використовувався для відстеження стану змінних та перевірки правильності виконання подій.

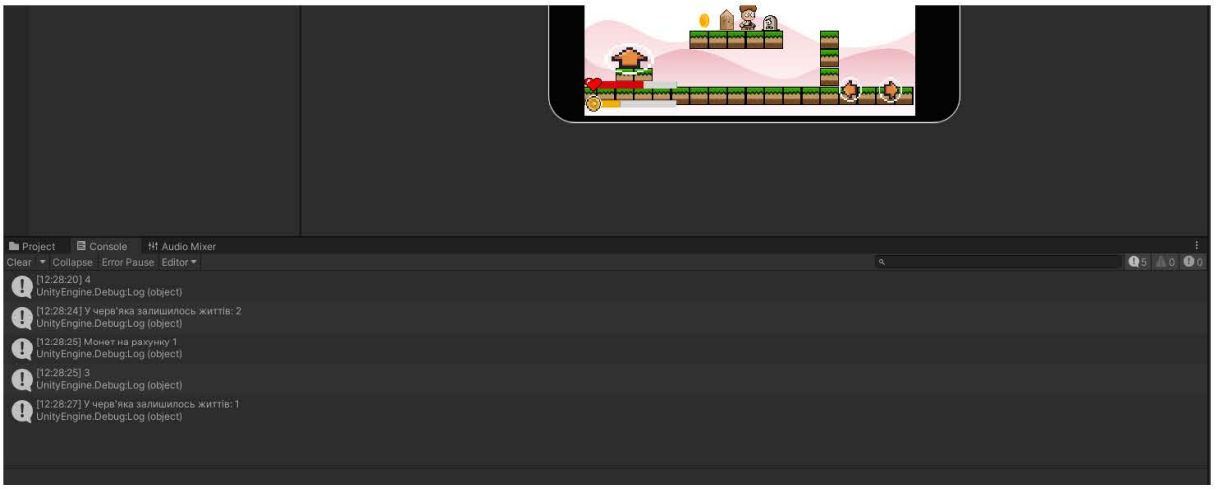

Рисунок 3.12 - Вигляд консолі з повідомленнями про події

Результати

Після усунення помилок та повторного тестування було досягнуто стабільної роботи проекту. Гра коректно реагує на дії користувача, UI оновлюється в реальному часі, переходи між сценами виконуються без затримок, а всі ігрові об’єкти взаємодіють згідно з очікуваною логікою.

3.5 Збірка проекту

Завершальним етапом розробки є створення збірки проєкту, яка дозволяє запускати гру як окремий додаток. Для цього у середовищі Unity передбачено

інструмент Build Settings (див.рис. 3.13), що забезпечує збірку гри під різні платформи.

Основні налаштування перед збіркою

Перед створенням збірки здійснюється налаштування параметрів у розділі Player Settings, зокрема:

- задається унікальний ідентифікатор пакета (Package Name);
- визначається назва гри, версія та номер білду;
- обирається орієнтація екрана (Portrait);
- конфігурується масштабування інтерфейсу за допомогою компонента Canvas Scaler;
- встановлюються іконки та інші ресурси.

Для забезпечення можливості збірки під Android середовище Unity має бути налаштоване з використанням відповідних інструментів Android SDK, JDK і NDK, які інтегруються через Unity Hub.

Створення інсталяційного файлу

Після завершення налаштувань виконується формування інсталяційного файлу шляхом натискання кнопки Build. У результаті генерується .apk файл, який може бути встановлений на Android-пристрій для перевірки працездатності гри в реальних умовах.

Під час збірки може виникати низка типових проблем, зокрема:

- відсутність Android SDK або JDK у системі;
- помилки, пов’язані з незареєстрованими сценами;
- надмірне використання ресурсів (наприклад, великі текстури, що впливають на розмір файлу).

Ці труднощі усуваються шляхом перевірки налаштувань, оптимізації ресурсів і коректного включення всіх сцен у білд.

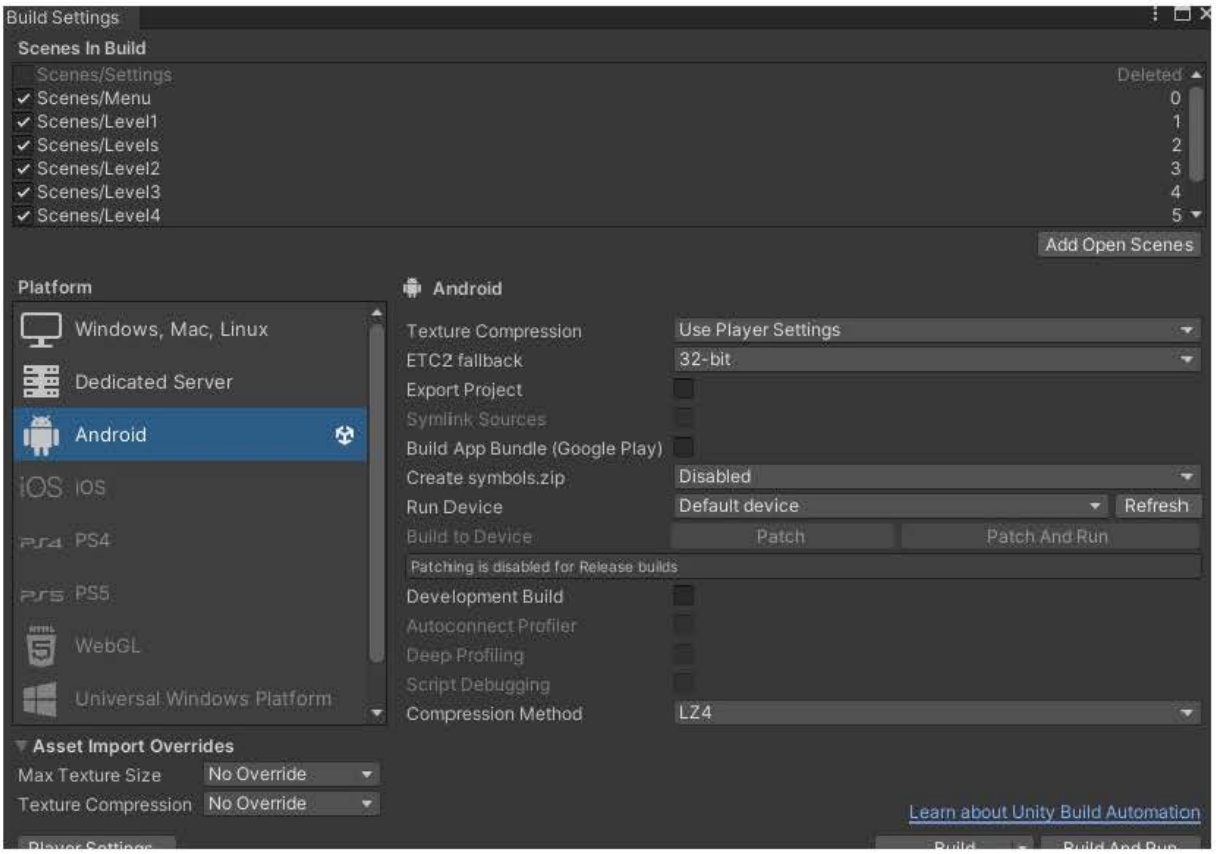

Рисунок 3.13 - вигляд інструменту Build Settings

3.5 Висновки до третього розділу

У цьому розділі було здійснено повноцінну реалізацію 2D платформера на базі рушія Unity, що охоплює всі ключові етапи створення гри — від проектування структури до розробки основної ігрової логіки. Розробка охоплювала управління головним персонажем, реалізацію системи колізій, обробку взаємодії з об'єктами, створення ворогів, анімацію та інтерфейс, що

дозволило забезпечити цілісність геймплейного досвіду.

Під час проектування використовувалися принципи модульності, розділення відповідальностей та повторного використання коду за допомогою префабів і шаблонних скриптів. Це дало змогу побудувати масштабовану архітектуру гри, зручно адаптовану як для подальшого розвитку, так і для публікації на мобільних платформах.

Особливу увагу було приділено зручності керування, стабільності роботи скриптів та чіткості анімацій — усі ці аспекти є критично важливими для комфортного користувацького досвіду. Також реалізовано адаптацію управління для мобільних пристроїв, що відповідає вимогам сучасного ринку мобільних ігор.

Отримані результати демонструють практичну ефективність застосованих підходів та підтверджують доцільність використання Unity як основного інструменту для створення 2D ігор. Розробка стала важливим етапом у формуванні компетентностей з програмування, дизайну ігрової логіки та організації проектів у сфері геймдизайну.

ВИСНОВОК

У межах виконання кваліфікаційної роботи було розроблено повнофункціональний 2D платформер з використанням рушія Unity та мови програмування C#, адаптований для мобільних пристроїв. Робота охопила повний цикл створення ігрового застосунку – від аналізу предметної області та вибору інструментів до реалізації функціональності, тестування та оптимізації.

Під час реалізації проекту було досягнуто наступних результатів:

- Проведено аналіз стану та тенденцій розвитку ігрової індустрії, з акцентом на жанр 2D платформерів і потреби мобільного сегменту.
- Обґрунтовано вибір технологій, зокрема рушія Unity, що забезпечує гнучкість, багатофункціональність та кросплатформеність.
- Спроектовано модульну архітектуру гри, яка дозволяє легко масштабувати проект, додавати нові рівні та функції.
- Реалізовано основні механіки геймплею: переміщення персонажа, стрибки, колізії, взаємодію з об'єктами, облік балів, завершення гри тощо.
- Забезпечено відтворення графіки, анімації та звукового супроводу в режимі реального часу.

  Розроблено зручну систему керування з урахуванням особливостей сенсорних екранів.
- Проведено тестування проєкту та оптимізацію його продуктивності, що забезпечило стабільну роботу на різних мобільних пристроях.

Результатом виконаної роботи є стабільний, функціональний ігровий продукт, який може бути використаний:

- як основа для подальшої комерційної розробки,
- як навчальний приклад при вивченні технологій Unity та C#,
- як демонстраційний зразок для портфоліо розробника.

Проект підтверджує ефективність використання сучасних архітектурних підходів, інструментів та оптимізаційних технік при створенні 2D ігор, особливо

у сфері мобільних застосунків. Усі поставлені в роботі завдання були успішно виконані, а мета – досягнута.

Пропозиції щодо вдосконалення:

- Розширення функціоналу гри:
  - додавання нових рівнів, локацій та сюжетних елементів;
  - впровадження нових типів ворогів, перешкод або бонусів;
  - реалізація системи збереження прогресу гравця.
- Покращення візуальної частини:
  - використання покращеної анімації.
- Розробка системи досягнень та рейтингу:
  - мотивація гравця через систему нагород, балів, глобальних таблиць лідерів.
- Публікація гри у цифрових магазинах:
  - підготовка та розгортання гри на платформах Google Play, App Store тощо.
- Покращення UI/UX:
  - адаптація інтерфейсу до різних розмірів екранів;
  - впровадження більш інтуїтивного та адаптивного меню.
- Використання сервісів аналітики:
  - інтеграція Google Firebase або аналогів для збору статистики про дії користувачів з метою подальшого вдосконалення геймплею.

Розширення функціональності, вдосконалення візуальної складової та залучення сучасних інструментів дозволить перетворити цей проект у конкурентоспроможний продукт на ринку мобільних ігор.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Історія відеоігор. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/History_of_video_games (дата звернення: 28.05.2025).

2. Еволюція відеоігор. [Електронний ресурс]. Режим доступу: https://www.techradar.com/news/the-evolution-of-video-games-50-years-of-fun (дата звернення: 28.05.2025).

3. Інструменти для налагодження. [Електронний ресурс]. Режим доступу: https://unity.com/how-to/profiling-and-debugging-tools (дата звернення: 28.05.2025).

4. Що таке платформер. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Platformer (дата звернення: 28.05.2025).

5. Як зробити якісний платформер. [Електронний ресурс]. Режим доступу: https://devsourcehub.com/mastering-2d-platformer-mechanics-in-unity/ (дата звернення: 28.05.2025).

6. Найкращі ігрові рушії. [Електронний ресурс]. Режим доступу: https://gamedevacademy.org/best-game-engines/ (дата звернення: 28.05.2025).

7. Unity Technologies. (2023). Unity Manual. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/index.html (дата звернення: 28.05.2025).

8. Unity Documentation, Canvas. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/UICanvas.html (дата звернення: 28.05.2025).

9. Unity UI Toolkit Guide. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/UIElements.html (дата звернення: 28.05.2025).

10. Unity Input System Manual. [Електронний ресурс]. Режим доступу:https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/index.html (дата звернення: 28.05.2025).

11. Unity 2D Animation. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Packages/com.unity.2d.animation@latest (дата звернення: 28.05.2025).

12. Unity Documentation. Rigidbody2D and Physics – Unity Manual. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/class-Rigidbody2D.html (дата звернення: 28.05.2025).

13. Martin R.C. "Clean Architecture: A Craftsman's Guide to Software Structure and Design", Prentice Hall, 2017.

14. GameAnalytics. (2023). Mobile Gaming Genre Trends. [Електронний ресурс]. Режим доступу: https://gameanalytics.com/blog/mobile-game-genres-trends-2023 (дата звернення: 28.05.2025).

15.Unity Learn. Unity project structure: best practices. [Електронний ресурс]. Режим доступу: https://learn.unity.com/tutorial/project-structure (дата звернення: 28.05.2025).

16. Важливість плавності управління та швидкого відгуку. [Електронний ресурс]. Режим доступу:https://learn.unity.com/ (дата звернення: 28.05.2025).

17. Колайдери у Unity. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Manual/CollidersOverview.html (дата звернення: 28.05.2025).

18. Геймдизайн. [Електронний ресурс]. Режим доступу: https://www.gamedeveloper.com/design (дата звернення: 28.05.2025).

19. Архітектура у розробці ігор. [Електронний ресурс]. Режим доступу: https://unity.com/how-to/advanced-programming-and-code-architecture (дата звернення: 28.05.2025).

ДОДАТОК А

***Код скрипта Audio Manager:

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
   [SerializeField] AudioSource musicSource;
   public AudioClip Background;
   private bool isPaused=true;

   private void Start()
   {
      musicSource.clip = Background;
      musicSource.Play();
      isPaused = PauseMenu.gameispaused;
   }

   private void Update()
   {
      if (PauseMenu.gameispaused && !isPaused)
      {
         musicSource.Pause();
         isPaused = true;
      }
      else if (!PauseMenu.gameispaused && isPaused)
      {
         musicSource.Play();
         isPaused = false;
      }
   }
}
```

***Код скрипта Bars:

```
using System.Data.SqlTypes;
using UnityEngine;
```

```
using UnityEngine.Audio;
using UnityEngine.UI;

public class Bars : MonoBehaviour
{
   [SerializeField] private Slider Healthbarslider;
   [SerializeField] private Slider Moneybarslider;

   private Entity money;
   private Hero hhhrrr;
   private void Start()
   {
      money = FindObjectOfType<Hero>();
      hhhrrr = FindObjectOfType<Hero>();
   }
   private void Update()
   {
         Healthbarslider.value = hhhrrr.lives;
         Moneybarslider.value = money.coinnumber;
   }
}
```

***Код скрипта ButtonActivasion:

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

public class ButtonActivasion : MonoBehaviour
{
   public Button[] buttons;

   public Color dimmedTextColor = Color.gray;
   public Color normalTextColor = Color.white;
   public static int numberofunlockedlevels = 1;
   public void Awake()
   {
      for(int i = 0; i < buttons.Length; i++)
```

```
        {
            buttons[i].interactable = false;
        }
        for (int i = 0; i < numberofunlockedlevels; i++)
        {
            buttons[i].interactable = true;
        }
    }
    private void Update()
    {
        for (int i = 0; i < buttons.Length; i++)
        {
            TextMeshPro buttonText = buttons[i].GetComponentInChildren<TextMeshPro>();

            if (buttonText != null)
            {
                buttonText.color = buttons[i].interactable ? normalTextColor : dimmedTextColor;
            }
        }
    }
}
```

***Код скрипта Camera_controller:

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


public class Camera_controller : MonoBehaviour
{
    [SerializeField] private Transform player;
    private Vector3 pos;

    private void Awake()
    {
        Hero foundHero = FindObjectOfType<Hero>();
        if (foundHero != null)
```

```
        {
            player = foundHero.transform;
        }
    }

    private void Update()
    {
        if (player != null)
        {
            pos = player.position;
            pos.z = -10f;
            transform.position = Vector3.Lerp(transform.position, pos, Time.deltaTime);
        }
    }
}
```

***Код скрипта Entity :

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

public class Entity : MonoBehaviour
{
    public int coinnumber;
    public int lives;
    public virtual void GetDamage()
    {
        lives--;
        Debug.Log(lives);
        if (lives < 1)
            Die();
        if (lives <= 0)
        {
            Die();
        }
    }
```

```
    public virtual void Die()
    {
        Destroy(this.gameObject);
    }
}
```

***Код скрипта FinishChecker :

```
using System.Collections;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.SceneManagement;
using UnityEngine.UI;


public class FinishChecker : MonoBehaviour
{
    public GameObject player;
    private float startAlpha = 0f;
    private float endAlpha = 1f;
    private float time = 3/2;
    public UnityEvent Myevents;
    [SerializeField] public string scene;
    [SerializeField] private Animator anim;
    public Image dimPanel;
    public float fadeDuration = 1f;
    int levelnumber;

    private void Start()
    {
        levelnumber = int.Parse(SceneManager.GetActiveScene().name.Substring(5));
        if (dimPanel != null)
        {
            dimPanel.enabled = false;
            SetAlpha(0);
        }
    }
    private void OnTriggerEnter2D(Collider2D other2D)
```

```
    {
        if (other2D.CompareTag("Player"))
        {
            dimPanel.enabled = true;
            StartCoroutine(LoadSceneWithDelay());
            if (ButtonActivasion.numberofunlockedlevels == levelnumber)
            {
                ButtonActivasion.numberofunlockedlevels = levelnumber + 1 ;
            }

        }
    }

    private IEnumerator LoadSceneWithDelay()
    {
        Myevents.Invoke();
        yield return new WaitForSeconds(time);
        float elapsedTime = 0f;
        while (elapsedTime < fadeDuration)
        {
            elapsedTime += Time.deltaTime;
            float newAlpha = Mathf.Lerp(startAlpha, endAlpha, elapsedTime / fadeDuration);
            SetAlpha(newAlpha);
            yield return null;

        }
        SetAlpha(endAlpha);
        SceneManager.LoadScene(scene);
    }
    private void SetAlpha(float alpha)
    {
        Color color = dimPanel.color;
        color.a = alpha;
        dimPanel.color = color;
    }
}
```

***Код скрипта Hero:

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


public class Hero : Entity
{
    public float fallThreshold = -10f;
    [SerializeField] private float speed = 5f;
    [SerializeField] private float jumpForse = 15f;
    [SerializeField] private LayerMask groundLayerMask;
    public static Hero Instance { get;set; }
    private Rigidbody2D rb;
    private SpriteRenderer sprite;
    private Animator anim;
    private bool isgrounded = false;

    private void Start()
    {
        lives = 5;
    }
    public void Awake()
    {
        Instance = this;
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponentInChildren<SpriteRenderer>();
    }
    private void Run()
    {
        if (isgrounded) state = States.run_animation;
        Vector3 dir = transform.right * VirtualInput.Horizontal;
        transform.position=Vector3.MoveTowards(transform.position,                transform.position+dir,
speed*Time.deltaTime);
        sprite.flipX = dir.x < 0.0f;
    }
```

```
private void Jump ()
{
   rb.AddForce(transform.up * jumpForse, ForceMode2D.Impulse);
}
private void checkground()
{
   if (!isgrounded) state = States.jump;
   Collider2D[] collider = Physics2D.OverlapCircleAll(transform.position, 0.3f);
   isgrounded=collider.Length > 1;
}
private void FixedUpdate()
{
   if (Hero.Instance.lives > 0)
   {
      checkground();
   }
   if (PauseMenu.gameispaused) return;
   if (isgrounded) state = States.Default;
   if (VirtualInput.Horizontal != 0)
   {
      Run();
   }
   if (isgrounded && VirtualInput.JumpPressed)
   {
      Jump();
      VirtualInput.JumpPressed = false;
      isgrounded = false;
   }
}


public enum States
{
   Default,
   run_animation,
   jump
}
```

```
    private States state
    {
        get { return (States)anim.GetInteger("state"); }
        set { anim.SetInteger("state", (int)value); }
    }
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (!other.CompareTag("Coin")) return;
        {
            coinnumber++;
            Debug.Log("Монет на рахунку " + coinnumber);
            Destroy(other.gameObject);
        }
    }
}
```

***Код скрипта MobileInput:

```
 using UnityEngine;

public class MobileInput : MonoBehaviour
{
    public void MoveLeftStart() => VirtualInput.Horizontal = -1f;
    public void MoveRightStart() => VirtualInput.Horizontal = 1f;
    public void StopMoving() => VirtualInput.Horizontal = 0f;
    public void Jump() => VirtualInput.JumpPressed = true;
}
```

Код скрипта MovingPlatform:

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

public class MovingPlatform : MonoBehaviour
{
    [SerializeField] float speed;
    Vector3 targetpos;
    public GameObject ways;
```

```
public Transform[] waypoints;
int pointindex;
int pointcount;
int direction;

private SpriteRenderer sprite;
private void Awake()
{
   waypoints = new Transform[ways.transform.childCount];
   for (int i = 0; i < ways.gameObject.transform.childCount; i++)
   {
      waypoints[i]=ways.transform.GetChild(i).gameObject.transform;
   }
}
private void Start()
{
   pointcount= waypoints.Length;
   pointindex = 0;
   targetpos = waypoints[pointindex].transform.position;
}
private void Update()
{
   var step = speed* Time.deltaTime;
   transform.position=Vector3.MoveTowards(transform.position, targetpos, step);
   if (transform.position == targetpos)
   {
      Nextpoint();
   }
}
void Nextpoint()
{
   if (pointindex == pointcount - 1)
   {
      direction = -1;
   }
   if (pointindex == 0)
   {
      direction=1;
```

```
        }
        pointindex += direction;
        targetpos = waypoints[pointindex].transform.position;
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Player"))
        {
            collision.collider.transform.SetParent(transform);
        }
    }
    private void OnCollisionExit2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Player"))
        {
            if (collision.collider != null && collision.collider.gameObject.activeInHierarchy)
            {
                collision.collider.transform.SetParent(null);
            }
        }
    }

}
```

***Код скрипта Obstacle :

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

public class Obstacle : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
```

```
    }

}
```

***Код скрипта PauseMenu :

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
  public static bool gameispaused = false;
  public GameObject pauseMenuUI;
  public GameObject RestartMenuUI;
   public Transform player;
 private float fallThreshold = -30f;

   private void Start()
   {
        pauseMenuUI.SetActive(false);
        RestartMenuUI.SetActive(false);
   }

   void Update()
   {
      if (Input.GetKeyDown(KeyCode.Escape))
      {
         if (gameispaused)
         {
            Resume();
         }
         else
         {
            Pause();
         }
      }
      if (Hero.Instance.lives <= 0 || player.position.y < fallThreshold)
      {
```

```
            Gameover();
        }

    }
    public void Gameover()
    {
        RestartMenuUI.SetActive(true);
    }
    public void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    public void Resume()
    {
        if (pauseMenuUI != null)
        {
            pauseMenuUI.SetActive(false);
            Time.timeScale = 1f;
            gameispaused = false;
        }

    }

    public void Pause()
    {
        if (RestartMenuUI.activeSelf)
        {
            return;
        }
        if (pauseMenuUI != null)
        {
            pauseMenuUI.SetActive(true);
            Time.timeScale = 0f;
            gameispaused = true;
        }
    }
}
```

***Код скрипта SceneMenegerScript :

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class SceneMenegerScript : MonoBehaviour
{
   void Start()
   {
      Canvas.ForceUpdateCanvases();
      LayoutRebuilder.ForceRebuildLayoutImmediate(GetComponent<RectTransform>());
   }
   public static void LoadScene(string sceneName)
   {
      Time.timeScale = 1f;
      SceneManager.LoadScene(sceneName);
   }
   public static void Loadlastlevel()
   {
      Time.timeScale = 1f;
      SceneManager.LoadScene("Level"+ ButtonActivasion.numberofunlockedlevels);
   }
   public void Quitgame()
   {
      Application.Quit();
   }
}
```

***Код скрипта SettingsMenu :

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

public class SettingsMenu : MonoBehaviour
```

```
{
    public GameObject settingsMenuUI;
    void Start()
    {
        settingsMenuUI.SetActive(false);
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if(settingsMenuUI != null)

            settingsMenuUI.SetActive(false);
        }
    }
    public void settingsbuttonpressed()
    {
        settingsMenuUI.SetActive(true);

    }
    public void exitsettingsbuttonpressed()
    {
        settingsMenuUI.SetActive(false);
    }

}
```

***Код скрипта VirtualInput :

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

public static class VirtualInput
{
    public static float Horizontal = 0f;
    public static bool JumpPressed = false;
}
```

***Код скрипта VolumeSettings:

```
using UnityEngine;
using UnityEngine.Audio;
using UnityEngine.UI;

public class VolumeSettings : MonoBehaviour
{
   [SerializeField] private AudioMixer audioMixer;
   [SerializeField] private Slider musicVolumeSlider;
   private const string MusicVolumeKey = "MusicVolume";
   private void Start()
   {
      float savedVolume = PlayerPrefs.GetFloat(MusicVolumeKey, 0.75f);
      musicVolumeSlider.value = savedVolume;
      SetMusicVolume(savedVolume);
   }
   public void SetMusicVolume()
   {
      float volume = musicVolumeSlider.value;
      SetMusicVolume(volume);
   }
   private void SetMusicVolume(float volume)
   {
      audioMixer.SetFloat("music", Mathf.Log10(volume) * 20);
      PlayerPrefs.SetFloat(MusicVolumeKey, volume);
   }
}
```

***Код скрипта WalkingMonster:

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

public class WalkingMonster : Entity
{
```

```
    private Vector3 direction;
    private SpriteRenderer sprite;
    private LayerMask coinLayer;
    private void Awake()
    {
        sprite = GetComponentInChildren<SpriteRenderer>();
        coinLayer = LayerMask.GetMask("Coin");
    }
    private void Start()
    {
        direction = transform.right;
    }
    private void Move()
    {
        Collider2D[] colliders = Physics2D.OverlapCircleAll(transform.position + transform.up * 0.1f + transform.right * direction.x * 0.7f, 0.1f, ~coinLayer);

        if (colliders.Length > 0) direction *= -1f;
        transform.position = Vector3.MoveTowards(transform.position, transform.position + direction, Time.deltaTime);
        sprite.flipX = direction.x > 0.0f;

    }
    private void Update()
    {
        Move();
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Hero.Instance.GetDamage();
        }
    }
}
```

***Код скрипта Worm:

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

public class Worm : Entity
{
    private void Start()
    {
        lives = 3;
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject == Hero.Instance.gameObject)
        {
            Vector2 contactPoint = collision.GetContact(0).point;
            Vector2 enemyPosition = transform.position;
            float playerBottom = Hero.Instance.transform.position.y -
Hero.Instance.GetComponent<Collider2D>().bounds.extents.y;

            if (playerBottom > enemyPosition.y + 0.2f)
            {
                lives--;
                Debug.Log("У черв'яка залишилось життів: " + lives);

                if (lives < 1)
                    Die();
            }
            else
            {
                Hero.Instance.GetDamage();
            }
        }
    }
}
```