ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему:

«ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РОЗРОБКИ ТА МАРКЕТИНГУ 2D-ІГОР»

Виконав: студент *2* курсу, групи *6ПР1*
спеціальності 121 «Інженерія програмного забезпечення»
Міроник М.В.
(прізвище та ініціали)

Керівник: *Хохлов В.А.*
(прізвище та ініціали)

Рецензент *М.О.Вороненко*
(прізвище та ініціали)

Херсон – 2025 р.

Факультет *Інформаційних технологій та дизайну*
Кафедра *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *магістр*
Галузі знань *12 «Інформаційні технології»*
Спеціальність *121 «Інженерія програмного забезпечення»*
Освітньо-професійної програми *«Програмна інженерія»/«Програмне забезпечення систем»*

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗіТ,
доцент
_____О.Є.Огнєва
«____» _____ 2025 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Міроника Миколи Вікторовича
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження особливостей розробки та маркетингу 2D-ігор*

керівник роботи *Хохлов Вадим Анатолійович, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ХНТУ від «*15*» *вересня* 20*25* року № *484-с*.
2. Строк подання студентом роботи *05.12.2025*
3. Вихідні дані до роботи *літературні та періодичні джерела, матеріали передипломної практики*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *аналіз предметної галузі, аналіз аналогічних проектів; розробка технічного завдання; розробка архітектури додатку; розробка додатку, оптимізація та тестування.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
20 рисунків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____*15.09.2025*_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.09.2025-09.10.2025	
2	Розробка технічного завдання	10.10.2025-22.10.2025	
3	Розробка архітектури додатку	23.10.2025-03.11.2025	
4	Розробка, оптимізація та тестування додатку	04.11.2025-18.11.2025	
5	Написання пояснювальної записки до дипломного проекту	19.11.2025-05.12.2025	

Студент _____ *Міроник М.В*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Хохлов В.А.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота магістра складається з таких структурних елементів: вступу, трьох розділів, висновків та списку використаних джерел.

Перший розділ «Дослідження та аналіз предметної області» включає п’ять підрозділів: «Аналітичний огляд джерел», «Аналіз поточної ситуації», «Порівняння існуючих рішень», «Недоліки та можливості для вдосконалення в розробці та маркетингу 2D-ігор» та «Висновки до розділу». У межах цього розділу здійснюється комплексний аналіз предметної області, пов’язаної зі створенням і просуванням 2D-ігор. Досліджуються сучасні тенденції ігрової індустрії, підходи до розробки 2D-проєктів, а також маркетингові стратегії, що застосовуються для їх популяризації. Окрему увагу приділено аналізу існуючих ігрових рішень, порівнянню підходів різних розробників та виявленню ключових проблем і можливостей для вдосконалення як технічної реалізації ігор, так і процесів їх ринкового просування.

Другий розділ «Основи методології розробки додатку» містить такі підрозділи: «Архітектура і дизайн додатку», «Використані інструменти і технології», «Опис структури проєкту», «Декомпозиція програмного додатку», «Процес тестування та налагодження» та «Висновки до розділу». У цьому розділі розглянуто методологічні підходи до розробки 2D-ігор, особливості побудови архітектури ігрового застосунку та формування його дизайну. Описано використані інструменти, технології та програмні засоби, що застосовуються під час створення 2D-ігор, а також наведено структуру проєкту та логіку взаємодії його компонентів. Значну увагу приділено питанням декомпозиції ігрового застосунку, процесам тестування та налагодження, які забезпечують стабільність роботи гри та позитивний користувацький досвід.

Третій розділ «Розробка та застосування продукту» складається з підрозділів: «Результати розробки програмного продукту», «Послідовність

роботи користувача з додатком», «Аналіз результатів застосування продукту», «Перспективи подальшого розвитку додатку» та «Висновки до розділу». У цьому розділі представлено практичну реалізацію ігрового продукту у форматі 2D-гри, описано результати її розробки та основні функціональні можливості. Наведено послідовність взаємодії користувача з грою, проаналізовано отримані результати застосування продукту з точки зору зручності використання та потенціалу для залучення аудиторії. Також визначено перспективи подальшого розвитку проєкту, зокрема можливості розширення ігрового функціоналу, вдосконалення дизайну та застосування додаткових маркетингових інструментів для просування гри.

ABSTRACT

The master’s qualification thesis consists of the following structural parts: an introduction, three chapters, a conclusion, and a list of references.

Chapter One, «Research and Analysis of the Subject Area», includes five subsections: «Analytical Review of Sources», «Analysis of the Current Situation», «Comparison of Existing Solutions», «Drawbacks and Opportunities for Improvement in 2D Game Development and Marketing», and «Conclusions to Chapter One». This chapter provides a comprehensive analysis of the subject area related to the development and marketing of 2D games. It examines current trends in the game industry, approaches to 2D game development, and marketing strategies used to promote game projects. Particular attention is paid to the analysis of existing game solutions, the comparison of different development approaches, and the identification of key challenges and opportunities for improving both the technical implementation of games and their market promotion processes.

Chapter Two, «Fundamentals of Application Development Methodology», contains the following subsections: «Application Architecture and Design», «Tools and Technologies Used», «Project Structure Description», «Decomposition of the Software Application», «Testing and Debugging Process», and «Conclusions to Chapter Two». This chapter discusses methodological approaches to 2D game development, the specifics of building the architecture of a game application, and the formation of its design. It describes the tools, technologies, and software solutions used during the development of 2D games, as well as the project structure and the logic of interaction between its components. Special attention is given to application decomposition and to testing and debugging processes that ensure stable game performance and a positive user experience.

Chapter Three, «Development and Application of the Product», consists of the following subsections: «Results of Software Product Development», «User Interaction Workflow», «Analysis of Product Application Results», «Prospects for

Further Development of the Application», and «Conclusions to Chapter Three». This chapter presents the practical implementation of the game product in the form of a 2D game and describes the results of its development and core functional features. The user interaction workflow is outlined, and the results of using the product are analyzed in terms of usability and audience engagement potential. In addition, prospects for further project development are identified, including opportunities to expand game functionality, improve visual design, and apply additional marketing tools to promote the game.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Дослідження особливостей розробки та маркетингу 2D-ігор» містить: 90 сторінок, 20 рисунків, 2 додатки, 17 літературних джерел.

Об'єкт дослідження – процеси розробки та просування 2D-ігор, що включають технічні, дизайнерські та маркетингові аспекти створення ігор для різних платформ.

Предмет дослідження – інструменти, методи та підходи до розробки 2D-ігор, а також стратегії маркетингу і дистрибуції ігрових продуктів, їх вплив на успішність проєкту, залучення користувачів та монетизацію.

Мета кваліфікаційної роботи – дослідження ключових етапів розробки 2D-ігор та аналіз ефективних маркетингових стратегій з метою створення конкурентоспроможного ігрового продукту.

Методи дослідження – аналіз наукових і публіцистичних джерел у сфері геймдеву та ігрового маркетингу, практичне вивчення процесу створення 2D-ігор з використанням сучасних рушіїв і мов програмування, а також порівняльний аналіз маркетингових підходів на основі існуючих ігрових проєктів.

Наукова новизна роботи полягає у систематизації та узагальненні підходів до розробки 2D-ігор у поєднанні з маркетинговими інструментами, орієнтованими на інді-розробників та невеликі студії.

Об'єктом розробки є концепція 2D-гри та супровідна маркетингова модель її просування на ігровому ринку. Розробка включає аналіз вимог до ігрового продукту, проєктування ігрової механіки та візуального стилю, реалізацію основних ігрових елементів, а також формування маркетингової стратегії з урахуванням цільової аудиторії. Архітектура проєкту побудована з

урахуванням масштабованості, зручності підтримки та можливостей подальшого розвитку, а
інтерфейс гри орієнтований на зрозумілість і зручність для гравців.

Результатом кваліфікаційної роботи є поглиблене дослідження особливостей розробки 2D-ігор та практичних підходів до їх маркетингового просування в умовах сучасного ігрового ринку. У роботі систематизовано основні етапи створення 2D-ігор — від формування ідеї та концепції до реалізації ігрових механік і візуального оформлення, а також проаналізовано вплив маркетингових інструментів на популярність і комерційний успіх ігрових проєктів. Отримані результати демонструють взаємозв'язок між якістю геймдизайну, технічною реалізацією гри та ефективністю її просування серед цільової аудиторії. Практична значущість роботи полягає у можливості використання отриманих висновків і рекомендацій у діяльності інді-розробників і невеликих ігрових студій. Новизна кваліфікаційної роботи визначається інтегрованим підходом до поєднання процесів розробки 2D-ігор із маркетинговими стратегіями, спрямованими на підвищення впізнаваності ігрового продукту, залучення користувачів та формування стабільної ігрової спільноти.

Ключові слова: 2D-ІГРИ, РОЗРОБКА ІГОР, ГЕЙМДЕВ, МАРКЕТИНГ ІГОР, ІГРОВИЙ ДИЗАЙН, ІНДІ-ІГРИ, ІГРОВА ІНДУСТРІЯ, ПРОСУВАННЯ ПРОДУКТУ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IT - інформаційні технології

ПЗ – програмне забезпечення

ПП - програмний продукт

ГР – геймрушій (ігровий рушій)

UI – інтерфейс користувача

UX – користувацький досвід

2D – двовимірна графіка

FPS – кількість кадрів за секунду

NPC – неігровий персонаж

AI – штучний інтелект

QA – забезпечення якості

MVP – мінімально життєздатний продукт

KPI – ключові показники ефективності

CPA – вартість залучення користувача

ROI – повернення інвестицій

ASO – оптимізація сторінки застосунку в магазині

CTR – коефіцієнт кліків

CPI – вартість встановлення гри

ВСТУП

Сучасна ігрова індустрія є однією з найбільш динамічних та прибуткових галузей інформаційних технологій, яка поєднує програмну інженерію, креативні індустрії та маркетинг. На відміну від традиційного програмного забезпечення, орієнтованого переважно на виконання визначених бізнес-функцій, відеоігри спрямовані на створення емоційного досвіду, занурення користувача у віртуальний світ і формування тривалого інтересу. Саме ця особливість визначає унікальність процесів розробки та просування ігрових продуктів[1].

Гра не є лише технічним інструментом — вона виступає культурним продуктом, що поєднує технології, мистецтво та психологію. Процес її створення включає взаємодію фахівців різних напрямів: програмістів, геймдизайнерів, художників, сценаристів, звукоінженерів, тестувальників і маркетологів. Успішність ігрового продукту визначається не тільки технічною стабільністю, а й рівнем залученості гравця, тривалістю його інтересу та здатністю формувати унікальний користувацький досвід. У зв’язку з цим розробка ігор потребує спеціальних методологій, що поєднують технічну дисципліну з креативністю та аналізом реакції аудиторії.

Однією з ключових рис геймдеву є висока гнучкість процесів розробки. На відміну від класичних IT-рішень, вимоги до ігрових продуктів можуть змінюватися навіть після релізу під впливом відгуків гравців і ринкових тенденцій[2]. Ігри розвиваються як «живі продукти», що передбачає регулярні оновлення, балансування механік і додавання контенту. Саме тому в ігровій індустрії широко застосовуються гнучкі методології розробки, зокрема Agile-підходи та практики CI/CD.

Важливу роль у створенні ігор відіграє геймдизайн, який визначає концепцію, механіки та спосіб взаємодії користувача з продуктом. Він формує емоційний і змістовний каркас гри, поєднуючи творчі ідеї з аналізом поведінки гравців. Постійне тестування та коригування ігрових елементів дозволяє досягти балансу між креативністю та очікуваннями цільової аудиторії.

Не менш значущим чинником успіху ігрових проєктів є маркетинг, який в ігровій індустрії базується на емоційному сприйнятті продукту та активності спільноти[3]. Просування здійснюється через трейлери, соціальні мережі, стримінгові платформи та співпрацю з інфлюенсерами. Досвід сучасних інді-проєктів демонструє, що ефективні маркетингові стратегії можуть істотно підвищувати популярність ігор.

Актуальність даної кваліфікаційної роботи зумовлена необхідністю комплексного дослідження особливостей розробки та маркетингу 2D-ігор як одного з найбільш доступних і поширених сегментів ігрової індустрії[4]. Саме 2D-ігри часто створюються незалежними розробниками та невеликими студіями, для яких поєднання технічних рішень, геймдизайну та маркетингу є вирішальним чинником успіху.

Метою кваліфікаційної роботи є дослідження особливостей розробки 2D-ігор та аналіз маркетингових підходів до їх просування з метою формування ефективної моделі створення конкурентоспроможного ігрового продукту. Для досягнення цієї мети застосовується системний підхід, що охоплює аналіз предметної області, вивчення існуючих ігрових рішень, а також оцінку процесів розробки, тестування й дистрибуції[5].

Практична значущість роботи полягає у можливості використання отриманих результатів і рекомендацій у діяльності інді-розробників та невеликих ігрових студій, які прагнуть створювати якісні 2D-ігри та ефективно просувати їх на сучасному ігровому ринку[6].

Апробація результатів бакалаврської кваліфікаційної роботи.

Одержані результат можна використовувати для удосконалення методики та інструментарію для створення та маркетингу 2D-відеоігор. Запропоноване дослідження використовувати для подальшого удосконалення цих методик.

Публікації:

1. Міроник М.В., Хохлов В.А. ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РОЗРОБКИ ТА МАРКЕТИНГУ ВІДЕОІГОР/ VI Всеукраїнської конференції молодих вчених, аспірантів та студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2025».

Структура: робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, додатків. Кваліфікаційна робота магістра складається з 85 сторінок, 20 рисунків, 1 додатку, 25 джерел.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналітичний огляд джерел

У сучасному середовищі розробки 2D-ігор аналітичний огляд наукових, професійних і галузевих джерел є необхідною передумовою створення конкурентоспроможного ігрового продукту. Ігрова індустрія постійно еволюціонує під впливом технологічного розвитку, змін у споживчій поведінці гравців та зростання конкуренції на ринку цифрових розваг[7]. У зв'язку з цим розробники змушені систематично аналізувати сучасні тенденції, жанрові особливості, популярні ігрові механіки та очікування цільової аудиторії, що дозволяє формувати обґрунтовані рішення на всіх етапах створення гри.

На відміну від традиційного програмного забезпечення, де основними критеріями якості виступають функціональність, стабільність і відповідність специфікаціям, у розробці ігор важливу роль відіграють емоційні реакції користувачів, рівень занурення у віртуальний світ та привабливість ігрового процесу[8]. Аналіз джерел у галузі геймдеву свідчить, що сучасні підходи до створення ігор ґрунтуються не лише на технічних показниках, а й на даних про поведінку гравців, таких як тривалість ігрових сесій, частота повернень, реакція на складність рівнів та особливості взаємодії з ігровими механіками. Таким чином, аналітика стає інструментом, який доповнює творчий процес, але не замінює його повністю.

На рисунку нижче наведено декомпозиційну схему гри з точки зору гравця та розробника, що дозволяє наочно продемонструвати різницю між сприйняттям продукту кінцевим користувачем і внутрішніми процесами його створення. Для гравця гра є цілісним емоційним досвідом, тоді як для

розробника вона складається з набору технічних, дизайнерських і логічних компонентів, які необхідно гармонійно поєднати.

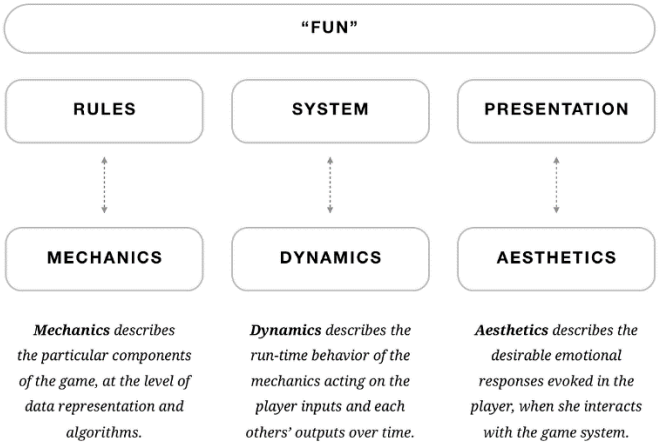

Рисунок 1.0 – декомпозиція гри за точки зору гравця та розробника

Важливим аспектом, що підкреслюється в досліджуваних джерелах, є необхідність балансування між використанням статистичних даних і збереженням творчого бачення команди розробників. З одного боку, аналіз зворотного зв'язку від користувачів і показників їхньої взаємодії з грою дозволяє виявити проблемні місця, покращити баланс ігрових механік та підвищити загальний рівень залученості. З іншого боку, надмірна орієнтація на кількісні метрики може призвести до стандартизації ігрових рішень та втрати унікальності продукту[9]. Саме тому у сучасній практиці геймдеву все більшого значення набуває поєднання аналітичного підходу з інтуїцією та креативністю геймдизайнерів.

Окрему увагу у джерелах приділено ролі геймдизайну як центрального елементу процесу розробки ігор[10]. Геймдизайн розглядається як міждисциплінарна діяльність, що поєднує елементи програмування, психології, художнього мислення та наративу. Геймдизайнер формує загальну концепцію гри, визначає правила, механіки, темп і спосіб взаємодії користувача з ігровим

середовищем[11]. Саме ця діяльність надає грі унікальний характер, тоді як програмна реалізація забезпечує технічне втілення задумів. На рисунку 1.1 представлено декомпозицію відеогри на основні складові, що ілюструє взаємозалежність між технічними, творчими та користувацькими компонентами.

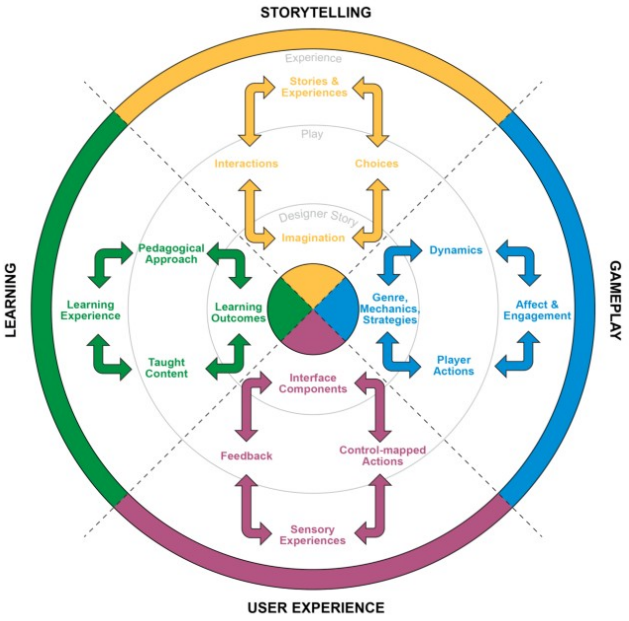

Рисунок 1.1. – декомпозиція відеогри на основні складові

Аналіз професійної літератури та практичних кейсів показує, що у розробці 2D-ігор широко застосовуються спеціалізовані ігрові рушії, серед яких значне місце займає Unity 2D. Даний інструмент забезпечує зручне середовище для реалізації ідей завдяки підтримці компонентного підходу, інтегрованих засобів роботи з фізикою, анімацією та інтерфейсом користувача[12]. Використання Unity 2D дозволяє скоротити час на розробку прототипів, швидко тестувати ігрові механіки та оперативно вносити зміни, що відповідає вимогам гнучких методологій розробки.

У більшості досліджень процес створення ігор описується як послідовність кількох ключових етапів[12]. Початковим етапом є концептуалізація, на якій визначаються ідея гри, її жанр, базові механіки та цільова аудиторія. Наступним кроком є проєктування, що включає підготовку дизайн-документації, створення прототипів і формування візуального стилю. Після цього відбувається етап програмної реалізації, під час якого впроваджуються механіки, система управління, фізика, анімація та інтерфейс. Завершальним етапом є тестування й доопрацювання продукту з урахуванням отриманого зворотного зв’язку, що дозволяє підвищити якість гри перед її випуском або подальшим розвитком.

Отже, проведений аналітичний огляд джерел підтверджує, що розробка 2D-ігор є складним багаторівневим процесом, який поєднує технічні, творчі та аналітичні аспекти[13]. Успішний ігровий продукт формується в результаті гармонійної взаємодії цих компонентів, що обґрунтовує необхідність комплексного підходу до дослідження предметної області у межах даної кваліфікаційної роботи.

1.2 Аналіз поточної ситуації

На сучасному етапі розвитку індустрії відеоігор 2D-ігри посідають важливе місце завдяки своїй доступності, відносно невисоким витратам на розробку та широким можливостям для творчої реалізації[14]. Поточна ситуація характеризується зростанням кількості незалежних студій та індивідуальних розробників, які використовують 2D-формат як основний інструмент для створення конкурентоспроможних ігрових продуктів. Водночас зростає і рівень очікувань з боку гравців, що змушує розробників приділяти значну увагу не лише технічній якості, а й емоційній складовій гри[15].

На відміну від традиційного програмного забезпечення, такого як веб-додатки, корпоративні системи або десктопні програми, 2D-ігри розробляються в умовах значної невизначеності[16]. Початкові вимоги до гри можуть змінюватися вже на етапі прототипування або раннього тестування. Це пояснюється тим, що більшість ігрових рішень перевіряються не лише логічно, а й через безпосередній досвід користувачів, який складно спрогнозувати заздалегідь.

Поточна ситуація на ринку демонструє, що аналітика поведінки гравців стала невід'ємною частиною процесу розробки. Розробники активно використовують дані про час перебування в грі, частоту взаємодії з механіками, рівень складності та точки виходу користувачів. Ці дані впливають на подальші рішення щодо балансування, зміни геймплею та структури рівнів. Таким чином, процес розробки 2D-ігор є ітеративним і передбачає постійне вдосконалення продукту на основі отриманих результатів[17].

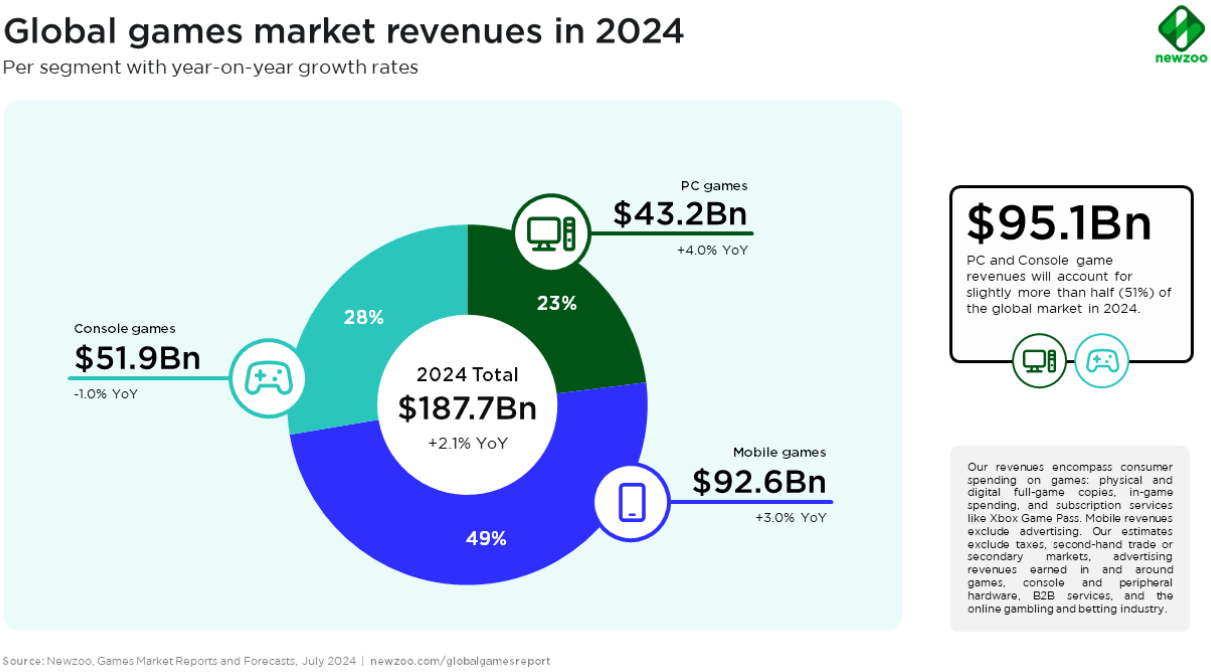

Рис 1.2 - доходи світового ринку ігор

Важливою ознакою поточної ситуації є також те, що життєвий цикл 2D-гри значно довший порівняно з традиційними програмними продуктами. Навіть після офіційного релізу гра продовжує активно підтримуватися через оновлення, виправлення помилок, додавання нового контенту та подальшу оптимізацію. Це зумовлює використання гнучких методологій розробки, таких як Agile, а також практик безперервної інтеграції та доставки (CI/CD).

Крім того, поточна ситуація в індустрії свідчить про тісний зв’язок між розробкою та маркетинговими активностями. Впровадження нових механік, сезонних подій або візуальних змін часто координується з рекламними кампаніями та роботою з ігровими спільнотами. Таким чином, розробка 2D-ігор виходить за межі суто технічного процесу і набуває міждисциплінарного характеру, поєднуючи програмування, геймдизайн, аналіз даних і маркетинг.

Маркетинг в ігровій сфері відрізняється від класичного бізнес-маркетингу, адже успіх продукту безпосередньо залежить від емоційної зацікавленості гравців. Трейлери, стрімінгові платформи, соціальні мережі та співпраця з інфлюенсерами формують образ гри та стимулюють бажання гравця стати частиною її світу. Приклади таких успіхів - Among Us та Minecraft, де популярність у значній мірі зростала завдяки спільнотам та користувацькому контенту, навіть коли на старті легальна дистрибуція була обмеженою[18].

Особливу роль у формуванні популярності продукту відіграє репутація студії. На відміну від корпоративного програмного забезпечення, негативні відгуки у геймдеві швидко стають публічними і впливають на загальне ставлення до бренду. Провальний реліз може знизити довіру та ускладнити майбутні проекти, тоді як успішні компанії, як Nintendo чи FromSoftware, здобувають лояльність гравців ще до виходу нових ігор.

Піратство в ігровій індустрії має двоякий характер. Воно дійсно може завдавати фінансових збитків, особливо для невеликих студій, де кожен продаж критично важливий для виживання команди. Водночас піратські копії інколи сприяють популяризації гри у регіонах із обмеженим доступом до офіційної дистрибуції. Такий ефект може стимулювати майбутні легальні продажі та підвищувати впізнаваність бренду[19].

Розробники використовують різні стратегії боротьби з піратством, від жорстких DRM-систем до творчих методів. Наприклад, у Game Dev Tycoon піратська версія гри отримувала спеціально запрограмований «баг», що демонстрував негативні наслідки піратства гравцеві у грі. Подібні підходи показують, що тема піратства у геймдеві є багатовимірною: вона поєднує фінансові ризики з маркетинговими можливостями та потребує нестандартних рішень, замість однозначного засудження.

Таким чином, хоча піратство може створювати труднощі, воно також підкреслює особливості ігрової індустрії: гнучкість, креативність та можливість використовувати будь-який канал розповсюдження для популяризації продукту. Для деяких студій правильний баланс між боротьбою з піратством і маркетинговою вигодою стає ключовим фактором успіху.

1.3 Порівняння існуючих рішень

Порівняння існуючих рішень у сфері 2D-ігор та традиційного програмного забезпечення дозволяє виявити фундаментальні відмінності не лише у функціональному призначенні продуктів, але й у підходах до їх розробки, життєвого циклу, підтримки та взаємодії з кінцевими користувачами.

Незважаючи на спільну технічну основу, ігрові продукти за своєю природою значно відрізняються від класичних програмних систем.

Традиційні програмні рішення - веб-застосунки, інформаційні системи, корпоративне програмне забезпечення - зазвичай розробляються відповідно до класичного життєвого циклу програмного забезпечення (Default Software Lifecycle). Такий підхід передбачає послідовні етапи: аналіз вимог, проектування, реалізацію, тестування, розгортання та подальшу підтримку. Вимоги до системи формуються на початковому етапі та залишаються відносно стабільними протягом усього життєвого циклу продукту. Основна мета таких рішень полягає у забезпеченні коректного виконання бізнес-функцій, стабільності, безпеки та передбачуваної роботи[20].

Рисунок 1.3 – lifecycle традиційного ПО

На відміну від цього, життєвий цикл розробки відеоігор (Video Game Development Lifecycle) має значно більш ітеративний і нелінійний характер, оскільки кінцева якість продукту безпосередньо залежить не лише від технічної реалізації, а й від суб’єктивного сприйняття ігрового досвіду користувачем. Розробка гри, як правило, починається з формування загальної концепції,

базової ігрової ідеї та створення первинних прототипів ключових механік. На цьому етапі основна увага зосереджується не на повноті функціоналу чи архітектурній завершеності системи, а на перевірці базового ігрового процесу, його динаміки, зрозумілості та привабливості для потенційного гравця[8].

Прототипування відіграє критично важливу роль, оскільки саме воно дозволяє на ранніх стадіях виявити сильні та слабкі сторони ігрової ідеї. Прототипи можуть багаторазово змінюватися, доопрацьовуватися або навіть повністю відкидатися у випадках, коли вони не викликають очікуваного емоційного відгуку або не забезпечують необхідного рівня залученості гравця. Такий підхід суттєво відрізняється від традиційної розробки програмного забезпечення, де зміна фундаментальних рішень на пізніх етапах є небажаною або економічно невигідною.

Рисунок 1.4 – video-game lifecycle

Подальші етапи розробки ігор включають активну фазу виробництва контенту, розширене плейтестування, балансування ігрових механік та постійну взаємодію з тестувальниками або спільнотою гравців. На відміну від традиційного ПЗ, де тестування спрямоване переважно на виявлення помилок, у відеоіграх значну роль відіграє оцінка складності, динаміки, зручності керування та загального ігрового досвіду[11]. Таким чином, тестування в ігровому життєвому циклі інтегроване в усі етапи розробки і не завершується навіть після релізу.

Суттєва відмінність також спостерігається на етапі післярелізної підтримки. У класичному програмному забезпеченні цей етап зводиться до технічного обслуговування, виправлення помилок та оновлень безпеки. У відеоіграх реліз часто є лише початком активної фази розвитку продукту. Оновлення контенту, сезонні події, додаткові рівні або механіки спрямовані на утримання гравців і підтримку їх інтересу впродовж тривалого часу.

Порівняння існуючих рішень також показує відмінності в організації командної роботи[4]. У традиційному програмному забезпеченні домінує інженерний підхід, де ключову роль відіграють програмісти та аналітики. У 2D-іграх команда має міждисциплінарний характер і включає геймдизайнерів, художників, аніматорів, звукорежисерів, тестувальників і фахівців з аналітики поведінки гравців. Така структура безпосередньо пов’язана з життєвим циклом гри, у межах якого технічні та творчі рішення постійно впливають одне на одне.

Окрему увагу варто приділити гнучкості процесу розробки. Якщо класичний життєвий цикл ПЗ орієнтований на мінімізацію змін після етапу реалізації, то ігровий життєвий цикл, навпаки, передбачає їх постійне внесення. Зміна ігрових механік, балансу або візуального стилю може відбуватися навіть на пізніх стадіях розробки чи після релізу, що робить гнучкі методології, такі як

Agile, CI/CD та швидке прототипування, особливо актуальними для ігрової індустрії.

Таким чином, порівняння життєвих циклів традиційного програмного забезпечення та відеоігор демонструє принципову різницю у підходах до створення програмних продуктів[7]. Розробка 2D-ігор характеризується ітеративністю, високою залежністю від користувацького досвіду, міждисциплінарною співпрацею та безперервним розвитком після релізу. Саме ці особливості визначають специфіку існуючих ігрових рішень та обґрунтовують необхідність їх окремого дослідження в межах даної кваліфікаційної роботи.

1.4 Недоліки і можливості для вдосконалення

Незважаючи на активний розвиток індустрії 2D-ігор та значну кількість наявних інструментів і методологій, процес їх розробки й просування все ще супроводжується низкою проблем і обмежень. Аналіз сучасної практики створення 2D-ігор дозволяє виокремити як типові недоліки, так і перспективні напрями для подальшого вдосконалення підходів до розробки та маркетингу ігрових продуктів.

Одним із ключових недоліків є високий рівень невизначеності кінцевого результату. На відміну від традиційного програмного забезпечення, де успіх продукту можна досить точно прогнозувати на основі бізнес-вимог і технічних показників, у 2D-іграх вирішальну роль відіграє суб’єктивне сприйняття гравців. Навіть технічно якісно реалізована гра може не отримати очікуваного відгуку через слабку ігрову ідею, невдалий баланс або недостатню емоційну

залученість. Це ускладнює планування ресурсів і підвищує ризики на всіх етапах життєвого циклу продукту.

Ще однією проблемою є обмеженість стандартних методологій розробки. Класичні підходи до управління проєктами не завжди ефективні в умовах постійних змін концепції гри та необхідності частого перегляду ігрових механік. Навіть гнучкі методології потребують адаптації під специфіку ігрової розробки, оскільки важко формалізувати такі параметри, як “цікавість”, “динамічність” або “емоційний відгук”. Це може призводити до комунікаційних труднощів у команді та до збільшення кількості ітерацій.

Також варто відзначити високі вимоги до міждисциплінарної взаємодії. У 2D-іграх результат значною мірою залежить від синергії між програмуванням, геймдизайном, візуальним стилем, звуковим оформленням і сценарною частиною. Недостатня узгодженість між цими складовими може негативно вплинути на цілісність ігрового досвіду та сприйняття продукту користувачами. Особливо це актуально для невеликих команд або незалежних розробників, де обмежені ресурси ускладнюють залучення фахівців з усіх необхідних напрямів.

Рисунок 1.5 – narrative pipeline

У великих ігрових проєктах навіть один департамент може мати складну багаторівневу структуру робочих процесів. Наприклад, Narrative Pipeline у розробці гри включає численні етапи: від формування концепції гри та жанру, визначення цільової аудиторії і ключових механік, до створення сценарію, розробки персонажів та їх мотивацій, побудови сюжетної структури, діалогів, інтеграції наративу у гру, тестування, локалізації для різних ринків і навіть запису голосів та motion capture.

Разом із тим, зазначені недоліки відкривають широкі можливості для вдосконалення процесів розробки 2D-ігор. Одним із перспективних напрямів є

активніше використання аналітики даних і поведінкових метрик гравців. Збір і аналіз інформації про дії користувачів дозволяє більш обґрунтовано приймати рішення щодо балансу, складності та структури ігрового процесу, зменшуючи рівень суб’єктивності.

Важливим напрямом розвитку є також удосконалення прототипування та плейтестингу на ранніх етапах. Швидке створення і перевірка прототипів дозволяє зменшити витрати часу й ресурсів на неефективні ідеї та зосередитися на найбільш перспективних механіках. Використання сучасних ігрових рушіїв і готових інструментів значно спрощує цей процес і робить його доступним навіть для невеликих команд.

Окрему увагу слід приділити інтеграції маркетингових підходів у процес розробки. Ранній аналіз цільової аудиторії, тестування ігрових концепцій та використання зворотного зв’язку з потенційними гравцями дозволяють не лише покращити сам продукт, а й підвищити його шанси на успіх на ринку.

Отже, виявлені недоліки сучасних підходів до розробки 2D-ігор водночас формують базу для їх подальшого вдосконалення. Використання ітеративних методів, поглибленої аналітики, міждисциплінарної взаємодії та орієнтації на потреби гравця створює умови для формування більш ефективних і стійких ігрових продуктів у сучасному цифровому середовищі.

1.5 Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено комплексне дослідження та аналіз предметної області розробки 2D-ігор, що дозволило сформувати цілісне уявлення про специфіку даної сфери та її принципові відмінності від традиційних підходів до створення програмного забезпечення.

У межах аналітичного огляду джерел було визначено, що розробка 2D-ігор орієнтована не лише на реалізацію функціональних вимог, а насамперед на створення емоційного та інтерактивного користувацького досвіду. Це зумовлює необхідність постійного врахування поведінки гравців, їх очікувань та ринкових тенденцій, а також поєднання технічних, творчих і аналітичних підходів у процесі розробки.

Аналіз поточної ситуації показав, що ігрові проєкти мають динамічний характер розвитку, що проявляється у частих ітераціях, активному використанні прототипування, постійному тестуванні та адаптації контенту. На відміну від традиційних програмних систем, життєвий цикл 2D-ігор є нелінійним та гнучким, а зміни можуть вноситися навіть на завершальних етапах або після офіційного релізу продукту.

Порівняння існуючих рішень дозволило виокремити ключові відмінності між класичним Software Development Lifecycle та Video Game Development Lifecycle. Було встановлено, що ігрова розробка характеризується більшою ітеративністю, активною міждисциплінарною взаємодією та орієнтацією на суб’єктивні показники якості, такі як залученість гравця, баланс ігрових механік та загальне враження від продукту.

Окрему увагу було приділено недолікам сучасних підходів та можливостям для їх удосконалення. Зокрема, визначено високі ризики, пов’язані зі складністю координації між різними департаментами, що є характерною проблемою для ігрових проєктів. Показовим прикладом цього є багаторівневий Narrative Pipeline у геймдеві, який включає велику кількість послідовних і паралельних етапів - від формування ідеї та сценарію до інтеграції наративу в ігрові механіки й фінального тестування. Масштабування таких процесів суттєво ускладнює управління проєктом, особливо для невеликих команд або незалежних розробників, де обмежені ресурси та

відсутність вузькоспеціалізованих фахівців можуть призводити до втрати цілісності ігрового бачення. Водночас ці виклики відкривають додаткові можливості для вдосконалення процесів розробки, зокрема шляхом оптимізації внутрішніх пайплайнів, автоматизації рутинних завдань, чіткого розмежування ролей у команді та впровадження гнучких методологій управління, що дозволяють ефективніше адаптуватися до змін і знижувати ризики на всіх етапах створення.

Таким чином, результати дослідження підтверджують, що розробка 2D-ігор є складним, багатокомпонентним і креативно-орієнтованим процесом, який потребує спеціальних підходів до планування, реалізації та підтримки продукту. Отримані висновки створюють теоретичне підґрунтя для подальшого проєктування та реалізації власного ігрового продукту в межах наступних розділів кваліфікаційної роботи.

РОЗДІЛ 2
ОСНОВИ МЕТОДОЛОГІЇ РОЗРОБКИ ДОДАТКУ

2.1 Архітектура і дизайн додатку

Архітектура розроблюваного 2D-платформера побудована на компонентному підході, який є базовим принципом організації програмного коду в більшості сучасних ігрових рушіїв, зокрема в середовищі Unity. В основі цього підходу лежить ідея представлення кожного ігрового об’єкта не як монолітної сутності, а як сукупності незалежних, логічно завершених компонентів. У Unity таким базовим елементом є GameObject, який сам по собі не має поведінки, а набуває функціональності лише після додавання до нього відповідних компонентів (скриптів, трансформів, колайдерів, рендерерів тощо). Саме комбінація компонентів визначає, як об’єкт виглядає, як він рухається та як реагує на дії користувача або навколишнього середовища.

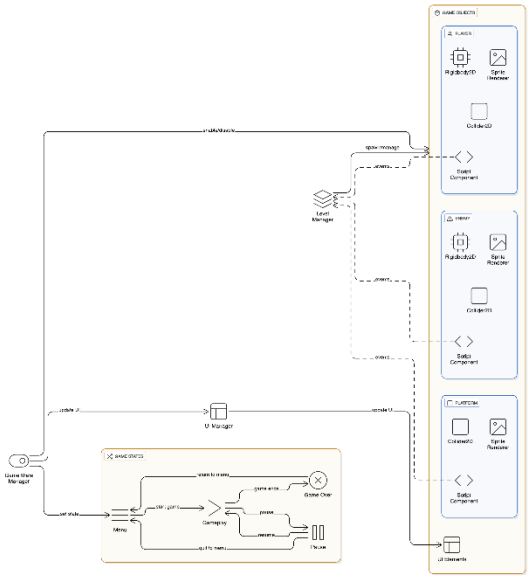

Рисунок 2.1 – загальна архітектура розроблювального додатку

Кожен компонент відповідає за окремий аспект поведінки або візуального відображення об’єкта: рух, фізичні взаємодії, відображення спрайтів, анімацію, обробку вводу користувача чи реакцію на події ігрового середовища. Такий підхід реалізує принцип composition over inheritance (композиція замість наслідування), коли функціональність формується шляхом додавання або вилучення компонентів, а не через складні ієрархії класів. Це робить структуру проєкту більш зрозумілою навіть для початківців, оскільки логіка поведінки розподіляється між окремими модулями з чітко визначеними обов’язками.

Використання компонентної моделі також спрощує процес налаштування та експериментування з ігровими механіками без потреби глибокого втручання в програмний код. Розробник може змінювати параметри компонентів безпосередньо в редакторі Unity, спостерігаючи результат у реальному часі. У підсумку такий архітектурний підхід дозволяє чітко розмежувати відповідальність між підсистемами гри, підвищує зрозумілість коду, а також значно спрощує супровід і масштабування проєкту на подальших етапах розробки.

Використання компонентної архітектури забезпечує високу гнучкість під час розробки. Одні й ті самі компоненти можуть повторно застосовуватись для різних об’єктів, що скорочує обсяг коду та зменшує ймовірність помилок. Крім того, зміни в поведінці окремого компонента не потребують втручання в загальну структуру гри, що позитивно впливає на стабільність та швидкість внесення оновлень. Такий принцип особливо важливий у контексті ігрової розробки, де механіки часто допрацьовуються та змінюються в процесі тестування.

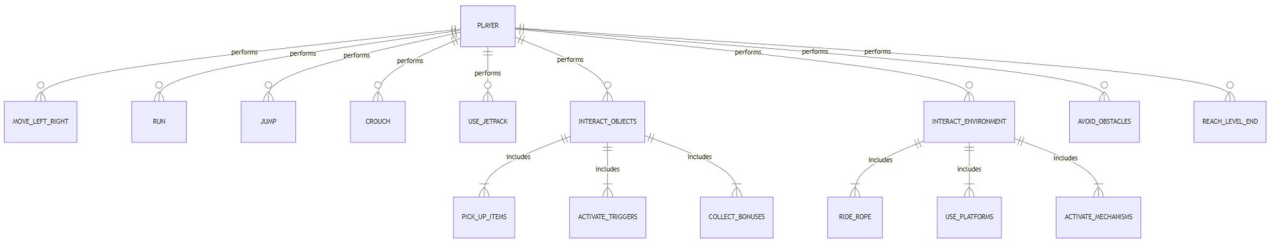

Рисунок 2.2 – use-case діаграма

Важливу роль в архітектурі додатку відіграють керуючі модулі, які забезпечують узгоджену роботу всіх систем гри. До них належать менеджери рівнів, система керування станами гри, менеджер ресурсів, а також підсистема взаємодії ігрових об’єктів між собою. Менеджери рівнів відповідають за послідовне завантаження сцен, ініціалізацію ігрового середовища, підключення необхідних об’єктів та їх налаштування відповідно до конкретного рівня. Як видно з узагальненої схеми логіки гри, такий підхід дозволяє чітко розділити етапи підготовки та виконання ігрового процесу.

Система управління станами гри забезпечує контроль переходів між основними режимами - головним меню, геймплеєм, паузою, екранами перемоги або поразки. Це дозволяє централізовано керувати логікою гри та уникати дублювання коду в окремих модулях. Як показано на узагальненому представленні взаємодії систем, використання єдиного механізму станів сприяє передбачуваній та стабільній поведінці додатку.

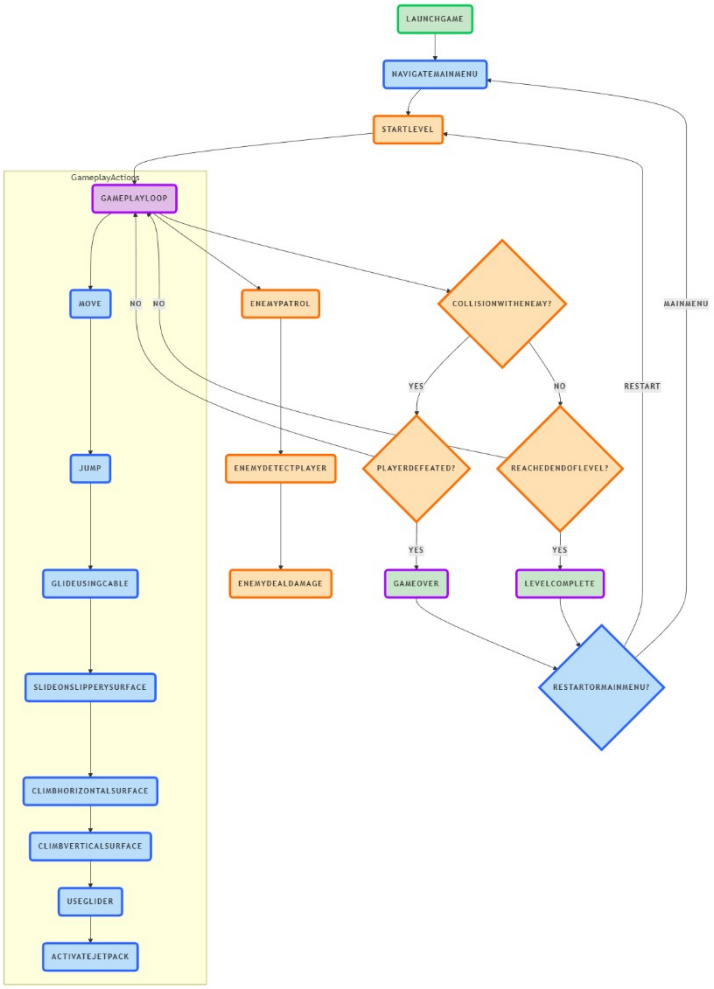

Рисунок 2.3 – діаграма активності

Для підвищення ефективності роботи з ресурсами використовується централізований менеджер, який координує процеси завантаження, кешування, повторного використання та вивільнення графічних, звукових і текстових даних. Такий підхід є критично важливим для 2D-проєктів, де обсяг дрібних, але численних активів - спрайтів, фреймів анімацій, тайлсетів, ефектів частинок - може стрімко накопичуватися та створювати відчутне навантаження на систему. Централізований менеджер дозволяє уникати дублювання даних і контролює життєвий цикл кожного ресурсу, завдяки чому значно зменшується кількість одночасно активних об’єктів у пам’яті. Оптимізована схема потоків даних забезпечує баланс між продуктивністю та стабільністю, дозволяючи грі

підтримувати плавний FPS навіть у сценах з підвищеною насиченістю графічних елементів які можуть бути на дисплеї.

Особливу роль відіграє логіка взаємодії між системами, яка стає зрозумілішою завдяки секвенційним діаграмам. На відповідній Sequence Diagram, створеній під час проєктування, відображено типовий сценарій, у якому користувач натискає кнопку для виконання дії - наприклад, стрибка, активації реактивного ранця, ковзання по поверхні або глайдингу на канаті. Діаграма демонструє, як сигнал від клавіатури потрапляє в Input System, потім передається до Player Controller, де перетворюється на команду поведінки, після чого перевіряється Game Logic на відповідність визначеним правилам геймплею. Подальша обробка включає взаємодію з Physics System, яка розв’язує колізії та визначає результуючий рух, а також з Enemy AI System, що реагує на наближення або дії гравця. Завершальний етап - оновлення візуального стану сцени та його передача до Rendering System, яка формує кадр, що гравець бачить на екрані.

У цьому контексті діаграма не лише описує ланцюжок викликів, але й дозволяє уявити загальну структуру внутрішніх взаємодій підсистем гри. Вона демонструє, що навіть одна проста дія гравця запускає каскад процесів, які координуються між собою через централізовані менеджери та системи обробки даних. Це підкреслює важливість оптимізованої архітектури, де кожен компонент виконує власну, чітко визначену функцію, а взаємодія між ними залишається структурованою, прозорою та передбачуваною.

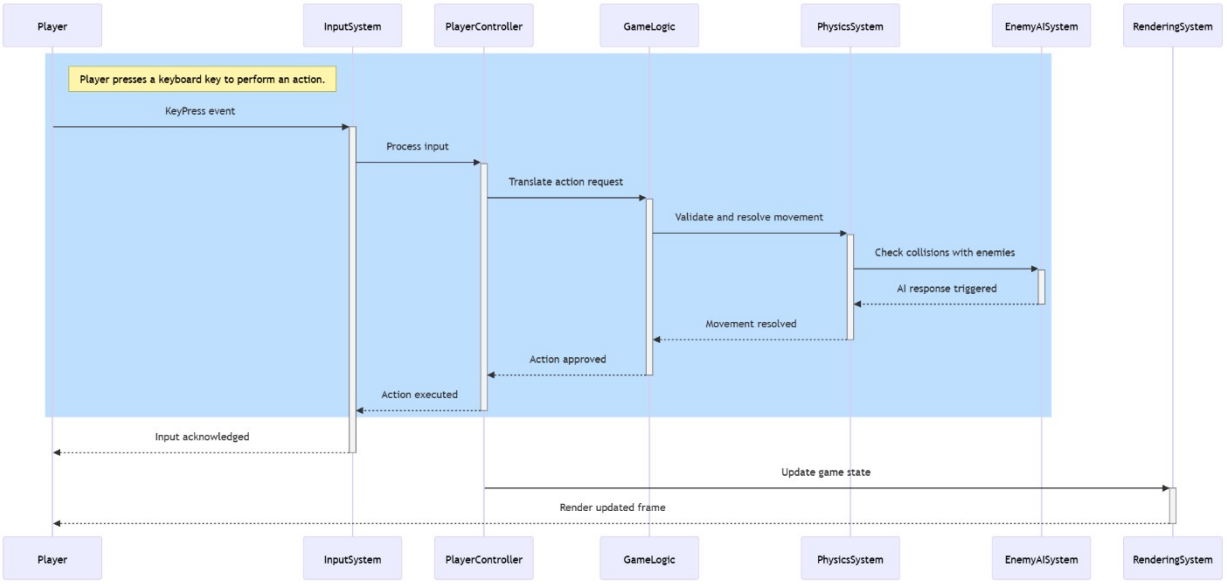

Рисунок 2.4 – діаграма послідовностей

З точки зору проєктування логіки та візуального представлення додатку застосовано принципи шаблону MVC (Model-View-Controller). Такий підхід дозволяє чітко відокремити ігрову логіку та внутрішні дані від механізмів їхнього відображення на екрані. Завдяки цьому будь-які зміни у візуальних елементах, стилях, структурі UI або спосіб взаємодії користувача не впливають на роботу базових механік та внутрішніх систем гри. Це суттєво спрощує підтримку проєкту, полегшує додавання нових функцій та підвищує масштабованість. Візуальна складова реалізується за допомогою стандартних інструментів інтерфейсу Unity - Canvas, EventSystem, UI Prefabs - що дозволяє зберігати єдину стилістичну концепцію, підтримувати адаптивність інтерфейсу та забезпечувати інтуїтивну взаємодію користувача із системою. Така структурована організація також полегшує тестування окремих частин UI без необхідності залучати всю ігрову логіку.

Окрему важливу роль у проєктуванні відіграє компонентна архітектура, яка була відображена в UML Component Diagram. У рамках Unity кожна сутність складається з набору модульних компонентів, і саме на цій концепції побудовано внутрішню структуру гри. Діаграма демонструє, як ключові підсистеми - такі як Game Manager, Game State Manager, Level Manager, Player Controller, Enemy AI Controller, Physics and Collision System, Movement Mechanics System, UI та Rendering System - взаємодіють одна з одною, формуючи цілісний робочий цикл гри. Компонент Game Manager координує глобальні процеси, тоді як Game State Manager відповідає за переключення між станами гри (меню, пауза, активна гра). Level Manager керує послідовністю рівнів та їхнім завантаженням. Player Controller отримує дані з Input System та передає їх у Movement Mechanics System і Physics System для обробки руху та колізій. Enemy AI Controller базується на простих поведінкових моделях та реагує на зміни стану гравця чи фізичного середовища. Rendering System завершує цикл, відображаючи актуальний стан сцени на екрані. Усі ці компоненти з’єднані чіткими залежностями та каналами передачі даних, що забезпечує зрозумілу модульну структуру, високу гнучкість і можливість повторного використання як окремих компонентів, так і цілих підсистем.

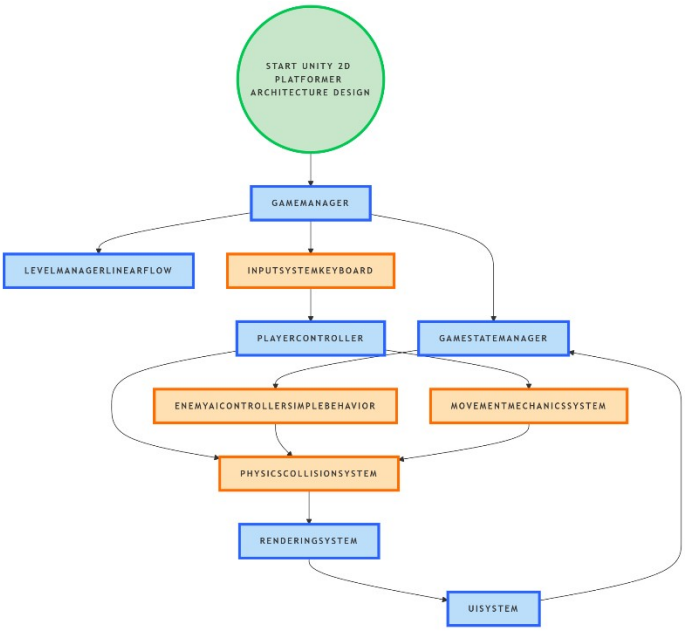

Рисунок 2.5 – діаграма компонентів

Узагальнюючи, архітектура та дизайн розроблюваного додатку орієнтовані на модульність, розширюваність та зручність супроводу, що є ключовими характеристиками якісної програмної системи у сфері ігрової розробки. Запропоновані підходи дозволяють легко адаптувати проєкт до нових вимог, інтегрувати додаткові ігрові механіки чи альтернативні режими проходження, а також розширювати функціонал без порушення логічної цілісності поточних систем. Структура взаємодії між основними підсистемами залишається прозорою та передбачуваною, що значно полегшує подальшу підтримку й оптимізацію гри. Ці принципи та рішення підтверджуються відповідними діаграмами, узагальненими моделями та структурними схемами, представленими у розділі, що наочно демонструють цілісний підхід до побудови внутрішньої логіки додатку.

2.2 Використані інструменти і технології

Основним інструментом розробки для створення 2D-платформера було обрано Unity Engine - один із найбільш розповсюджених рушіїв для інді-та комерційних проєктів. Unity поєднує інтуїтивний візуальний редактор, модульність побудови сцени та широкі можливості для програмування на C#, що робить його придатним як для швидкого прототипування, так і для побудови масштабних систем із чіткою архітектурою. Завдяки вбудованій підтримці 2D-графіки, системи анімацій, фізичних взаємодій, управління ресурсами та UI-фреймворку, рушій забезпечує усі необхідні інструменти для реалізації повноцінного ігрового циклу.

Для програмної частини проєкту ключову роль відіграє C#, який використовується для створення ігрових механік, логіки взаємодій об’єктів, контролю руху персонажа та роботи внутрішніх менеджерів. Використання компонентної архітектури Unity дозволило будувати код у вигляді автономних модулів, що легко тестуються, повторно використовуються та адаптуються до майбутніх змін у проєкті.

Особливу увагу приділено системі анімацій. Для цього застосовано Animator Controller, який дає змогу будувати розгалужені графи станів (біг, стрибок, планування, падіння, ковзання, атака, смерть тощо) із чітко визначеними переходами на основі параметрів і тригерів. Завдяки цьому поведінка персонажів та їхня візуальна реакція на події стає плавною, природною та керованою.

Візуальне відображення об’єктів реалізовано через Sprite Renderer, який відповідає за відтворення 2D-спрайтів та анімаційних кадрів, забезпечуючи коректне накладання шарів, освітлення та зміну кадрів у процесі руху персонажа або роботи анімацій. Фізичну модель гри підтримують компоненти Rigidbody2D та Collider2D, які є базовими інструментами Unity для створення поведінки, що підкоряється законам фізики. Rigidbody2D відповідає за фізичні

властивості об’єкта - вагу, інерцію, прискорення, взаємодію з гравітацією та здатність реагувати на сили, що дозволяє гравцю природно стрибати, падати чи ковзати поверхнями. Collider2D, у свою чергу, визначає форму невидимої «оболонки» навколо об’єкта, яка використовується для виявлення зіткнень із платформами, ворогами, тригерами та іншими елементами оточення. Це забезпечує коректну взаємодію персонажа з рівнем, включно з приземленням, зачепленням за стіни, ковзанням по слизьких поверхнях чи активацією подій на дотик. Використання вбудованого фізичного рушія Unity суттєво спрощує розробку, дозволяючи зосередитися на створенні ігрової логіки та поведінки об’єктів, а не на ручному розв’язанні низькорівневих фізичних задач.

Для збереження прогресу та конфігурацій гри використовувався PlayerPrefs, який дозволяє швидко записувати та зчитувати дані у форматі ключ-значення (наприклад: найвищий рекорд, налаштування звуку, відкриті рівні). У майбутньому планується перехід до більш масштабованої системи збережень на основі JSON-файлів, ScriptableObjects або власної системи серіалізації, що дозволить зберігати складніші структури даних.

Важливою частиною робочого процесу був контроль версій, який забезпечувався за допомогою Git. Це дозволяло відстежувати зміни, повертатися до стабільних версій проєкту та ефективно організовувати командну роботу. Основним середовищем розробки виступала Visual Studio, що забезпечувала зручний інструментарій для роботи з C#, налагодження коду та інтеграцію з Unity Engine.

Для створення графічних активів застосовувалися Adobe Photoshop та Aseprite - перший для підготовки складніших спрайтів і фону, другий - для покадрової анімації та піксель-арту. Звукові ефекти створювалися в інструменті Bfxr, який дозволяє швидко генерувати характерні 8-bit/retro-звуки, що ідеально підходить для стилістики 2D-платформерів.

Сукупність використаних технологій сформувала цілісну й гнучку екосистему, що охоплює всі етапи розробки - від проєктування архітектури та створення ігрових механік до підготовки ресурсів, тестування, оптимізації та налаштування фінальної збірки гри. Таке поєднання інструментів забезпечило ефективний робочий процес та можливість подальшого масштабування проєкту.

2.3 Опис структури проекту

Організація структури проєкту підпорядковується принципам чіткої ієрархії та логічного групування файлів, що є основою для підтримуваності та масштабованості сучасних ігрових застосунків. У межах Unity-проєкту структура папок була сформована з урахуванням кращих практик індустрії, що дозволяє легко орієнтуватися в ресурсах, сценаріях, анімаціях та сценах навіть на пізніх етапах розробки. Основна директорія поділена на функціональні модулі, що відображають логічну організацію проєкту: у папці Scripts містяться програмні класи, які реалізують поведінку персонажа, ворогів, UI-елементів, системи керування станами та менеджерів сцен; папка Prefabs зберігає уніфіковані об’єкти багаторазового використання, такі як стандартні вороги, платформні блоки, інтерактивні елементи та окремі частини інтерфейсу; у директорії Scenes містяться всі ігрові сцени - від головного меню та геймплейних рівнів до проміжних сервісних сцен на кшталт меню поразки чи тестових зон; папка Resources слугує сховищем для графічних активів, аудіофайлів, шрифтів і частинкових ефектів, що можуть завантажуватися динамічно за потреби; директорія Animations містить контролери станів, анімаційні графи та відповідні набори спрайтів, що забезпечують плавну візуальну реакцію об’єктів.

Усередині кожної сцени об’єкти додатково організовані за категоріями, що полегшує роботу розробника під час налагодження або внесення змін. Головні групи включають Environment для статичних та інтерактивних елементів світу, Enemies для ворожих NPC, Player для основного ігрового персонажа, UI для інтерфейсних панелей, кнопок і текстових віджетів, а також Triggers, де зберігаються об’єкти, відповідальні за скриптові події, переходи між зонами або запуск кат-сцен. Такий підхід дозволяє швидко знаходити потрібні елементи та знижує ризик помилок при модифікації сцени, оскільки логічна структура дає чітке розмежування відповідальностей.

Особливу увагу під час проєктування було приділено модульності. Кожна сцена створювалася як самодостатній модуль, який можна завантажувати, тестувати або змінювати ізольовано від решти проєкту. Переходи між сценами реалізовані через спеціалізований SceneManager, який керує процесом завантаження, коректним очищенням пам’яті та за потреби збереженням даних гравця. Такий підхід забезпечує гнучкість під час розширення гри: додавання нових рівнів, режимів чи меню не потребує перегляду базової структури, оскільки всі модулі взаємодіють через чітко визначені точки входу.

Для оптимізації роботи з пам’яттю застосовується динамічне завантаження ресурсів, яке дозволяє гнучко керувати обсягом активів, що перебувають у пам’яті в конкретний момент часу. Завдяки цьому гра може тримати в оперативній пам’яті лише ті елементи, які безпосередньо задіяні на поточному рівні або екрані. Активи - анімації ворогів, спрайтові набори, ефекти частинок, аудіофайли, тайлсети та інші - підтягуються під час ініціалізації сцени або виклику відповідної події, а після закінчення використання можуть бути безпечно вивантажені. Це значно знижує загальне споживання ресурсів і запобігає накопиченню невикористовуваних об’єктів у пам’яті, що часто є причиною лагів, падіння FPS або навіть аварійного завершення роботи.

Подібний підхід особливо важливий для платформ з обмеженими обчислювальними можливостями. На мобільних пристроях, де обсяг оперативної пам’яті значно нижчий, ніж на ПК, навіть кілька зайвих мегабайт невивантажених текстур можуть призвести до автоматичного завершення процесу операційною системою. У браузерних середовищах ситуація ще суворіша: обмеження WebGL, швидкість GC та поведінка вкладок роблять керування пам’яттю критично важливим для плавної роботи гри. Тому динамічне завантаження дозволяє забезпечити стабільний FPS, зменшити час завантаження сцен, а також підвищити загальну швидкодію, що напряму впливає на користувацький досвід.

Крім того, така система відкриває можливості для адаптивної оптимізації. Наприклад, залежно від рівня деталізації, який визначається продуктивністю устройства, можна завантажувати легші текстури або спрощені анімації. Це робить гру більш універсальною та дозволяє охопити ширший спектр платформ. Динамічне завантаження ресурсів також спрощує оновлення гри: нові активи можуть інтегруватися в окремі сцени без ризику порушити роботу всього застосунку. У підсумку такий підхід формує гнучку та ефективну систему управління ресурсами, яка гарантує стабільність, продуктивність та масштабованість проєкту на всіх етапах розробки й експлуатації.

Така структуризація створює чітку та передбачувану систему навігації в проєкті, яка підходить як для індивідуальної роботи, так і для командної взаємодії. Завдяки цьому проєкт легко масштабувати: можна додавати нові типи ворогів, сцени, механіки або елементи UI без ризику порушити цілісність базових модулів. У результаті структура папок і внутрішня організація сцен формують стабільний фундамент для подальшого розвитку гри, її тестування, оптимізації та підготовки до публікації.

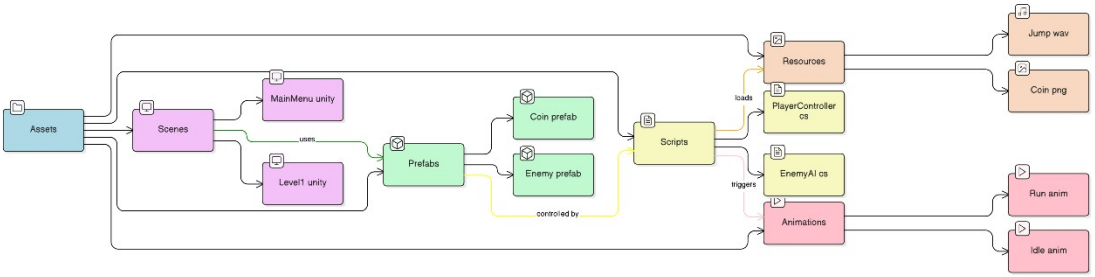

Рисунок 2.6 – зображення структури відео-гри

2.4 Декомпозиція програмного додатку

Декомпозиція програмного додатку є одним із ключових етапів у розробці будь-якої складної системи, зокрема відеоігор. Вона дозволяє розділити великий проєкт на менші функціональні блоки, які мають чітко визначені задачі, межі відповідальності та власні внутрішні механізми. Такий підхід робить процес створення гри значно прозорішим і більш керованим, адже кожен модуль можна проєктувати, розробляти, тестувати й оптимізувати окремо, не зачіпаючи роботу інших частин системи. Декомпозиція також полегшує командну роботу: різні учасники можуть паралельно працювати над окремими підсистемами, мінімізуючи конфлікти та залежності. Крім того, вона забезпечує високу адаптивність коду - зміни в одному модулі зазвичай не вимагають повного перегляду проєкту, а додавання нових механік або ігрових елементів стає значно простішим у майбутньому.

У межах розробленого 2D-платформера програмна система поділена на низку логічно незалежних модулів, що охоплюють роботу гравця, ворогів, фізики, інтерфейсу, ресурсів та ігрових станів, створюючи гнучку та масштабовану архітектуру.

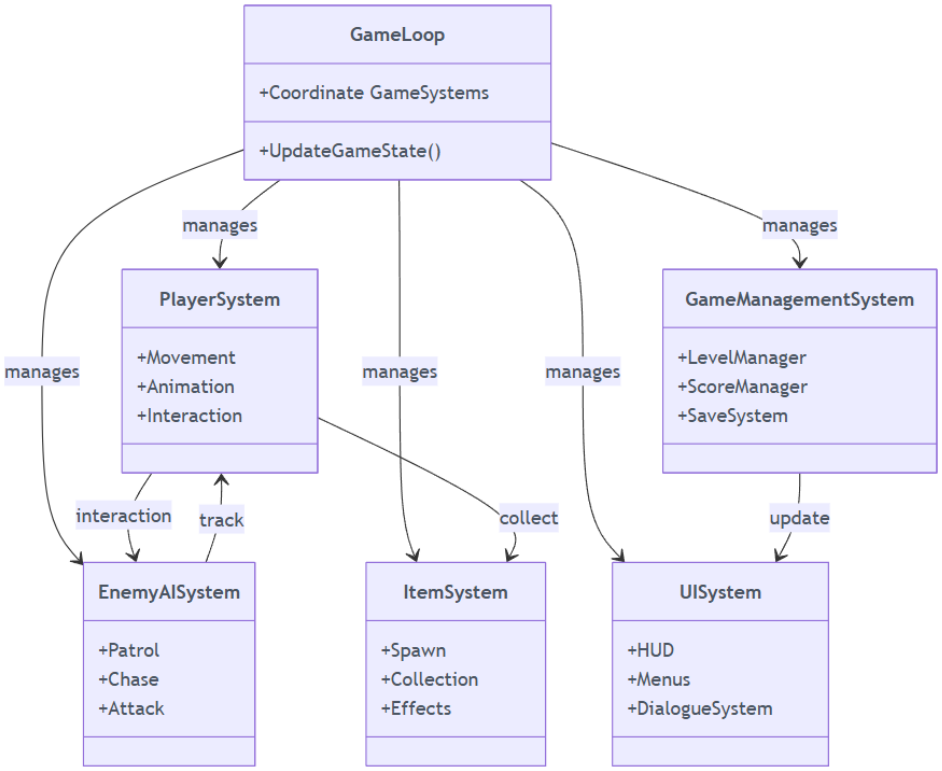

Рисунок 2.7 – загальна декомпозиція системи

Основні з даних модулів включають Player System - комплексну систему управління персонажем, яка відповідає за рух, інтерпретацію вводу користувача, роботу з фізичним середовищем та всі взаємодії між гравцем і об’єктами рівня. Це одна з центральних частин архітектури гри, адже саме вона забезпечує чутливість управління, плавність анімацій та передбачувану поведінку персонажа. Player System побудована за принципом модульності, тому складається з кількох внутрішніх підсистем, кожна з яких виконує вузькоспеціалізовану роль, але всі вони взаємодіють через чітко визначені інтерфейси.

Movement Controller є підмодулем, який відповідає за обчислення руху персонажа, керування швидкістю, обробку стрибків, падіння та інших параметрів фізичної мобільності. Він працює у зв’язці з фізичним рушієм Unity та використовує дані від компонентів Rigidbody2D і Collider2D, щоб коректно відтворювати поведінку героя у просторі. Саме Movement Controller забезпечує

плавну динаміку керування, коригує траєкторію руху, контролює прискорення та сповільнення, а також відтворює такі механіки, як ковзання по схилах чи відштовхування від стін. Ретельно налаштовані параметри цього модуля впливають на відчуття гри, роблячи управління інтуїтивним та приємним для майбутнього користувача.

Animation Handler виконує завдання синхронізації всіх дій персонажа з відповідними анімаційними станами. Він відстежує зміни у фізичній та логічній поведінці героя - наприклад, початок бігу, фазу стрибка, момент приземлення або активацію атаки - і відповідно перемикає стани Animator Controller. Це дозволяє досягти плавності й природності візуального відображення рухів. Модуль також керує переходами між анімаціями, визначаючи їх тривалість і пріоритет, що є критично важливим для створення реалістичної та візуально приємної реакції персонажа на дії гравця.

Interaction Module відповідає за всі форми взаємодії персонажа з об’єктами середовища. Він оцінює, які об’єкти доступні для активної дії, визначає їх тип - наприклад, предмети для збору, монети, важелі, двері або тригери - і викликає відповідні події гри. Завдяки цьому модулю персонаж може впливати на світ, отримувати ресурси, відкривати нові зони або запускати механізми. Interaction Module також забезпечує розширюваність: до системи можна легко інтегрувати нові типи взаємодій, не змінюючи базові механіки Player System. Це робить структуру системи гнучкою, придатною для масштабування та зручною для майбутнього тестування.

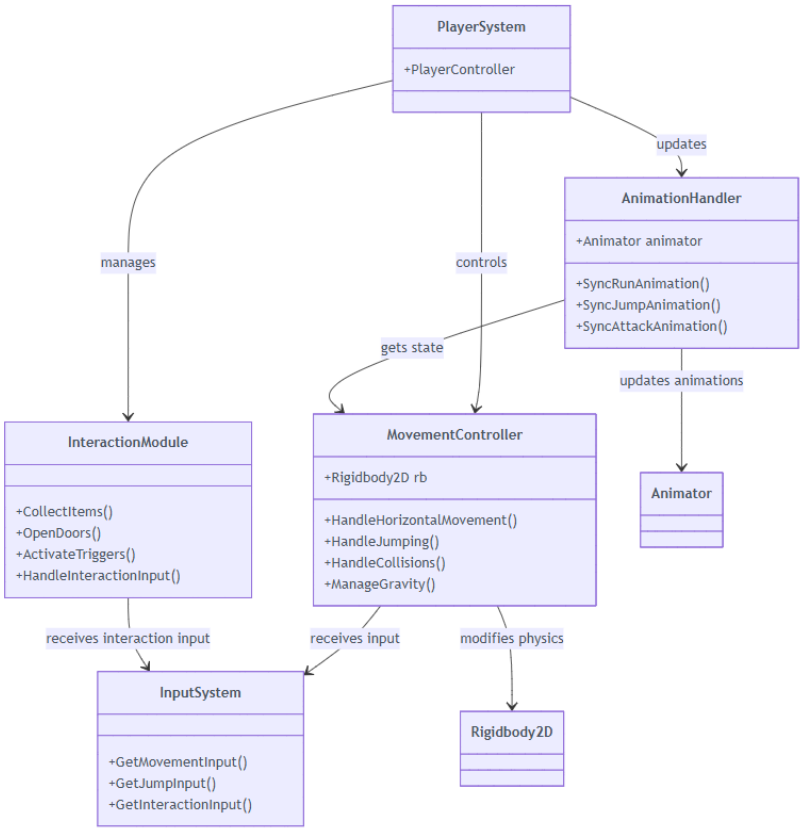

Рисунок 2.8 – декомпозиція системи гравця

Enemy AI System (система штучного інтелекту ворогів) відповідає за керування поведінкою ворогів у грі, забезпечуючи динамічну та передбачувану реакцію на дії гравця та зміни у середовищі. Вона реалізує поведінку ворогів за допомогою відносно простих алгоритмів, що дозволяє створювати відчуття живого світу без надмірного навантаження на систему. Основні механіки включають патрулювання території, реагування на наближення гравця, переслідування та атаки, що робить гру більш цікавою та викликає у гравця необхідність стратегічно підходити до проходження рівнів. Для кожного типу ворога передбачено декілька станів поведінки, таких як Idle (очікування), Patrol (патрулювання), Chase (переслідування) та Attack (атака), між якими відбуваються плавні переходи залежно від умов навколишнього середовища, таких як видимість гравця, відстань до нього або наявність перешкод.

Система також враховує взаємодію ворогів із іншими елементами гри, наприклад, зі статичними об’єктами середовища чи іншими NPC, що дозволяє реалізувати групову поведінку та уникати колізій. AI-модуль налаштований так, щоб поведінка ворогів здавалася правдоподібною, створюючи виклики для гравця і підтримуючи темп геймплею. Крім того, Enemy AI System побудована модульно, що дає змогу додавати нові типи ворогів або змінювати алгоритми їх поведінки без необхідності переписувати всю систему. Це підвищує гнучкість розробки, спрощує тестування та оптимізацію ігрового процесу, а також дозволяє легко масштабувати складність ворогів залежно від рівня.

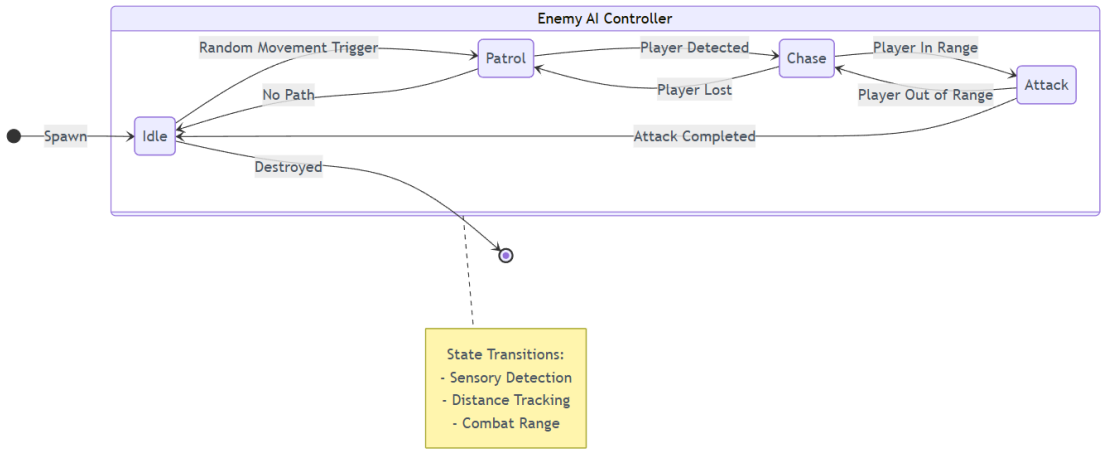

Рисунок 2.9 – декомпозиція системи ворогів

Item System (система предметів) включає логіку збору та обробки різноманітних елементів геймплею, таких як монети, бонуси, тимчасові покращення або спеціальні предмети, що впливають на характеристики персонажа. Вона відповідає не лише за реєстрацію того, які предмети були зібрані гравцем, а й за оновлення їхнього стану протягом гри та збереження цих даних для подальшого відновлення прогресу. Система також забезпечує інтерактивне відображення змін у користувацькому інтерфейсі, наприклад, збільшення рахунку, показ іконок зібраних бонусів чи оновлення шкали здоров’я. Крім того, Item System може тригерити певні події у грі: від запуску

анімацій і звукових ефектів до активації спеціальних механік, таких як тимчасове підвищення швидкості, захист від ворогів чи відновлення ресурсів персонажа. Такий модульний підхід дозволяє легко масштабувати систему, додавати нові типи предметів і взаємодії, а також гнучко інтегрувати їх у загальний геймплей без необхідності суттєвих змін у інших підсистемах.

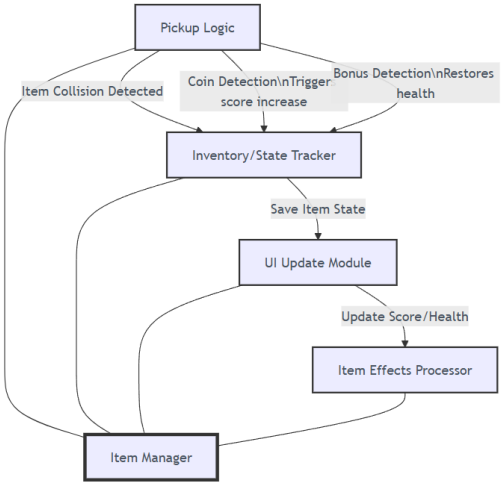

Рисунок 2.10 – декомпозиція системи предметів

Game Management System (система управління ігровим процесом) координує взаємодію між усіма іншими підсистемами, забезпечуючи узгоджену роботу компонентів та цілісність ігрового досвіду. В її основі знаходиться модуль GameManager, який відповідає за підтримку поточного стану гри, контролюючи переходи між різними режимами, такими як головне меню, геймплей, пауза або завершення рівня. Крім того, GameManager здійснює управління рівнями, відстежує прогрес гравця, підраховує очки, життєві ресурси та інші ключові показники, що впливають на взаємодію користувача з грою. Цей модуль також ініціює процес збереження прогресу за допомогою PlayerPrefs, що дозволяє зберігати дані про досягнення гравця та відновлювати їх при наступному запуску. Під час переходів між сценами GameManager готує необхідні дані для завантаження нових елементів, забезпечує синхронізацію станів усіх активних об’єктів і контролює коректну ініціалізацію ресурсів, що

особливо важливо для підтримки плавного ігрового процесу та уникнення помилок при інтеграції нових механік. Завдяки такій архітектурі Game Management System виступає центральною ланкою, яка забезпечує стабільність, масштабованість і керованість всього проєкту.

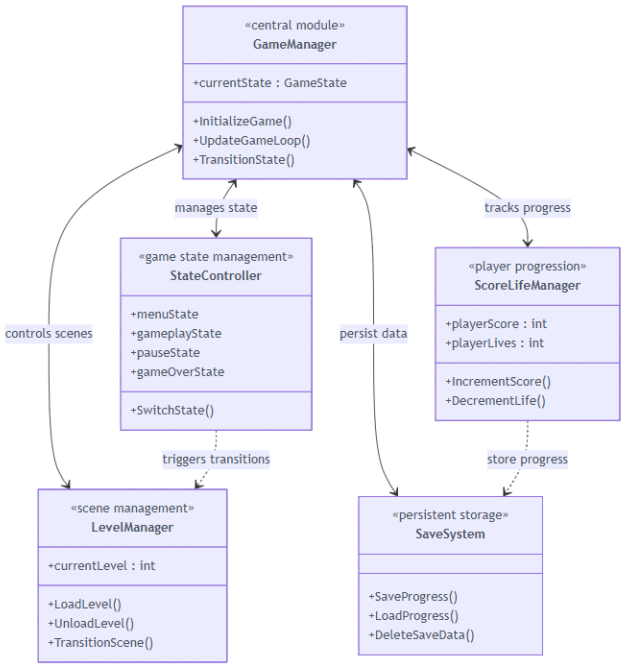

Рисунок 2.11 – декомпозиція Game Management System

UI System (система інтерфейсу користувача) охоплює всі елементи, що забезпечують взаємодію гравця з грою, включаючи кнопки меню, лічильники очок, шкалу здоров’я, таймери, повідомлення про виграш чи поразку та підказки під час проходження рівнів. Структура UI побудована на Canvas-компонентах Unity, що дозволяє організувати візуальні елементи у ієрархічну структуру, забезпечуючи гнучке управління їхнім відображенням та поведінкою. Для коректного відображення на різних роздільностях екранів використовується адаптивне масштабування та прив’язка елементів до певних точок екрану, що гарантує однаковий користувацький досвід на моніторах, мобільних пристроях та планшетах. Крім того, UI System відповідає за динамічне оновлення інформації в реальному часі, наприклад, зміни очок, стану

здоров’я або появу нових підказок, що робить взаємодію гравця більш інтуїтивною та інформативною. Така організація інтерфейсу забезпечує зручність, читабельність і швидку адаптацію під нові функції або додаткові ігрові механіки, що значно спрощує підтримку та масштабування гри.

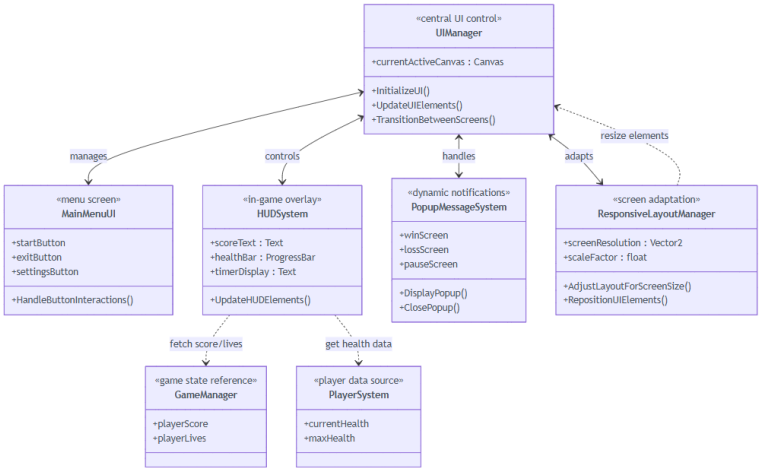

Рисунок 2.12 – декомпозиція UI System

Кожна система реалізована як окремий модуль із чітко визначеними інтерфейсами (API), що дозволяє інтегрувати нові функції без необхідності втручання в існуючий код або змін у інших компонентах проєкту. Такий підхід значно підвищує гнучкість розробки, оскільки будь-який модуль може бути оновлений або замінений незалежно від інших, при цьому основний функціонал гри залишається стабільним. Це особливо важливо для ітераційної розробки, коли нові ігрові механіки або елементи потрібно додавати швидко і без ризику порушити роботу вже реалізованих систем. Для читачів, які не знайомі з програмуванням, можна пояснити, що API виступає як своєрідний «контракт» між компонентами - він визначає, як модулі повинні взаємодіяти, без необхідності знати внутрішню реалізацію кожного з них.

Взаємодія між модулями реалізована за допомогою подійної системи UnityEvent та делегатів, що дозволяє компонентам «спілкуватися» один з одним без жорсткого зв’язку. Такий підхід знижує залежність між системами, роблячи

код більш гнучким та легким для розуміння навіть для нових учасників команди. Наприклад, зміна логіки роботи ворога не потребує змін у системі керування гравцем або інтерфейсі, бо вони реагують на події незалежно. Подібна архітектура сприяє масштабуванню проєкту - нові функції можуть додаватися без ризику порушення існуючого функціоналу, що є ключовим аспектом професійної розробки сучасних ігор.

Декомпозиція також забезпечує можливість паралельної розробки. Це означає, що члени команди можуть одночасно працювати над різними системами - один розробляє поведінку ворогів, інший - створює інтерфейс користувача, а третій - реалізовує механіку предметів та взаємодію з ними. Такий підхід не лише прискорює загальний процес створення гри, а й зменшує кількість конфліктів під час об’єднання гілок у системі контролю версій Git. Для новачків можна пояснити, що Git дозволяє кільком людям працювати над одними файлами одночасно, а правильна декомпозиція зменшує ризик, що їхні зміни «зіштовхнуться» або зіпсують одну і ту ж частину коду.

Таким чином, реалізована декомпозиція забезпечує високу підтримуваність, розширюваність та тестованість проєкту. Вона створює основу для ефективної ітераційної розробки, коли нові механіки, рівні або функції можна додавати поступово, без ризику порушення роботи гри. Крім того, такий підхід дозволяє проводити глибоке тестування окремих модулів, виявляти помилки на ранніх етапах та швидко їх виправляти, що в кінцевому підсумку покращує якість продукту і робить процес розробки більш передбачуваним та контрольованим.

2.5 Процес тестування та налагодження

Процес тестування є невід’ємною складовою розробки програмного забезпечення і набуває особливого значення у створенні відеоігор, де

стабільність роботи, продуктивність системи та якість геймплею безпосередньо впливають на загальне користувацьке враження та задоволення від продукту. У випадку 2D-платформера процес тестування був структурованим та поетапним, що дозволяло систематично оцінювати функціональність, взаємодію компонентів і поведінку гри в цілому. Для забезпечення комплексного контролю якості застосовувалися різні типи перевірок: модульне тестування, інтеграційне тестування, смуок (Smoke Testing), регресійне тестування та ігрове тестування (Playtesting), кожне з яких виконувало чітко визначену роль у процесі валідації та оптимізації гри.

На початкових етапах розробки проводилося модульне тестування, яке дозволяло перевірити правильність роботи окремих компонентів системи. Зокрема, тестувалися механізми обробки колізій між персонажем та ворогами, коректність взаємодії гравця з платформами та предметами, а також правильність нарахування очок при зборі монет чи бонусів. Модульне тестування дозволяло виявляти помилки на рівні окремих підсистем, що суттєво скорочувало час на їхнє виправлення, забезпечуючи стабільну роботу базових механік ще до інтеграції всіх елементів гри.

Далі проводилось інтеграційне тестування, спрямоване на перевірку взаємодії між різними модулями. Наприклад, оцінювалось, наскільки коректно система управління персонажем реагує на сигнали від GameManager, чи правильно оновлюється інтерфейс користувача після втрати життя або збору предметів, та як взаємодіють між собою фізична модель і логіка штучного інтелекту ворогів. Такий підхід дозволяв своєчасно виявляти проблеми на межі компонентів і уникати каскадного ефекту помилок, коли збій одного модуля може призвести до некоректної роботи інших.

Після інтеграції ключових елементів виконувалося смуок-тестування (Smoke Testing) - швидка перевірка базової працездатності гри після кожної

нової збірки. Це включало запуск гри, перевірку роботи меню, завантаження рівнів, базовий контроль фізики, руху персонажа, стрибків і колізій. Смоук-тести дозволяли оперативно переконатися у відсутності критичних помилок, що забезпечувало стабільність для подальшого детального тестування та ітерацій розробки відеогри.

Наступним кроком проводилося регресійне тестування, спрямоване на перевірку впливу внесених змін на вже стабільні частини гри. Наприклад, після коригування фізики стрибка або налаштувань ворогів перевірялося, чи не порушена взаємодія персонажа з платформами, чи правильно відображаються ефекти та UI, та чи не виникають нові помилки у вже перевірених механіках. Регресійне тестування дозволяло гарантувати, що внесені поліпшення або нові функції не створюють нових проблем і не впливають негативно на існуючий геймплей відеогри.

Окремий напрям тестування був присвячений оцінці ігрового балансу, юзабіліті та загального користувацького досвіду. Під час playtesting-сесій гравці перевіряли складність рівнів, інтуїтивність керування, комфортність інтерфейсу, відчуття швидкості та плавність анімацій. Зібрані відгуки використовувалися для корекції параметрів ворогів, швидкості руху персонажа, розташування платформ, а також для балансування ігрових механік, щоб забезпечити одночасно виклик та задоволення від проходження рівнів. Такий підхід дозволяв отримати не лише технічну, але й ігрову валідність продукту.

Для налагодження та аналізу використовувалися вбудовані інструменти Unity, такі як Console для відстеження помилок і логів, Profiler для аналізу продуктивності та виявлення вузьких місць у роботі рушія, а також Frame Debugger для оптимізації процесу рендерингу. Крім того, була реалізована власна система логування подій, де кожна дія гравця або помилка системи записувалася із часовою міткою, що дозволяло локалізувати проблеми навіть

без необхідності запуску сцени у редакторі, прискорюючи процес відлагодження і підвищуючи ефективність тестування.

Оцінка продуктивності проводилася на різних конфігураціях обладнання для визначення «вузьких місць» і забезпечення стабільного FPS. Використовуючи Profiler, аналізували споживання пам’яті, кількість викликів рендерингу, навантаження на фізичний рушій та частоту кадрів. На основі отриманих даних були проведені оптимізації спрайтів, скорочено кількість фізичних об’єктів у сцені, застосовано техніку об’єднання (Batching) для зменшення навантаження на процесор і підвищення плавності гри. Такі заходи дозволили забезпечити комфортний ігровий процес навіть на середньопродуктивних пристроях і створили основу для подальшого масштабування гри.

Завдяки комплексному поетапному тестуванню, систематичному налагодженню та оптимізації продуктивності вдалося досягти стабільної роботи гри, відсутності критичних збоїв та приємного користувацького досвіду. Це створює міцну основу для подальшого розвитку продукту: розробки нових рівнів, додавання складніших механік, вдосконалення інтерфейсу та оптимізації для різних платформ, що робить проєкт більш конкурентоспроможним та готовим до випуску на ринок.

2.6 Висновки до розділу 2

Другий розділ присвячений комплексному аналізу методології розробки 2D-платформера, що включає архітектуру додатку, дизайн, використані інструменти та технології, організацію структури проєкту, декомпозицію програмного коду та процес тестування й налагодження. У цьому розділі показано, як системний підхід до розробки забезпечує не лише стабільність і

функціональність гри, але й її масштабованість, зручність супроводу та можливості для подальшого розвитку.

Архітектура додатку, базується на компонентному підході та шаблоні MVC, що дозволяє розділити логіку гри, її візуальне представлення та взаємодію з користувачем. Такий підхід забезпечує чітке визначення відповідальності між підсистемами та дозволяє інтегрувати нові механіки без втручання у вже стабільний код. Використання діаграм активності, послідовностей, компонентів і потоків даних допомагає наочно уявити взаємодію гравця із системами гри, управління станами, фізичними об’єктами та ворогами, а також відображення результатів у UI. Завдяки цьому читач може оцінити логіку обробки дій гравця і роль кожного компоненту у загальній архітектурі.

Було детально описано вибір технологій та інструментів розробки. Основним рушієм обрано Unity, який поєднує візуальне середовище для дизайну рівнів із потужними можливостями програмування на C#. Використання Animator Controller, Sprite Renderer, Rigidbody2D і Collider2D дозволяє створювати природну поведінку персонажів, забезпечує реалістичні фізичні взаємодії та підтримує плавність анімацій. Python-подібні принципи організації коду та модульність, адаптовані до Unity, сприяють гнучкості розробки та швидкому прототипуванню нових механік. Допоміжні інструменти, такі як Git для контролю версій та Photoshop/Aseprite для підготовки графічних активів, забезпечують ефективну командну роботу та високу якість кінцевого продукту.

Продемонстровано логіку організації структури проєкту, де чітка ієрархія папок та модульність сцен дозволяє легко управляти ресурсами, підключати нові рівні або персонажі та забезпечує автономність тестування окремих частин. Динамічне завантаження ресурсів, централізований менеджер даних та

адаптивний UI сприяють оптимізації продуктивності, зменшують навантаження на пам’ять та дозволяють підтримувати стабільність гри навіть на пристроях з обмеженими ресурсами.

У підрозділі «Декомпозиція програмного додатку» було показано розподіл основних підсистем гри: Player System, Enemy AI System, Item System, Game Management System та UI System. Кожен модуль розділений на підсистеми, що реалізують конкретну функціональність, мають чітко визначені інтерфейси та взаємодіють через події і делегати. Така структура не лише спрощує розробку та тестування, а й дозволяє паралельну роботу над різними аспектами гри, прискорюючи процес інтеграції та зменшуючи конфлікти у коді.

Підрозділ «Процес тестування та налагодження» показав важливість системного підходу до контролю якості. Модульне тестування забезпечило правильну роботу окремих компонентів, інтеграційне тестування перевірило взаємодію підсистем, а smoke- та регресійні тести дозволили оперативно відстежувати критичні помилки після змін у коді. Playtesting дав змогу оцінити ігровий баланс, інтуїтивність інтерфейсу та загальну зручність геймплею. Вбудовані інструменти Unity - Console, Profiler та Frame Debugger - спільно з власною системою логування подій забезпечили швидке локалізування та усунення помилок, а також оптимізацію продуктивності, включно з використанням batching та динамічного завантаження ресурсів.

Узагальнюючи, проведений аналіз показує, що реалізовані архітектура, дизайн, інструменти, модульна структура та методи тестування формують надійну базу для подальшого розвитку проєкту. Вони забезпечують гнучкість, масштабованість та ефективність розробки, дозволяють швидко адаптувати гру до змінних вимог користувачів, інтегрувати нові механіки та режими, підтримувати стабільність і високу якість користувацького досвіду.

РОЗДІЛ 3
РОЗРОБКА ТА ЗАСТОСУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Результати розробки програмного продукту

В результаті проведеної роботи було створено повноцінний 2D-платформер, який поєднує сучасні підходи до розробки відеоігор з акцентом на технічну стабільність, естетичну привабливість і зручність користувацького досвіду. Розробка показала ефективність обраних архітектурних рішень, використання компонентного підходу для організації ігрових систем, а також можливості рушія Unity у створенні інтерактивних, візуально насичених і динамічних середовищ.

Основна мета проекту полягала у створенні гри, яка не лише реалізує базові механіки платформера, такі як рух, стрибки, колізії та збір предметів, а й формує цілісний користувацький досвід. Гра передбачає поступове ускладнення ігрового процесу, де кожний наступний рівень додає нові елементи та виклики, що стимулюють гравця до застосування стратегічного мислення і точності дій. Такий підхід забезпечує баланс між технічною реалізацією та емоційним задоволенням від взаємодії з ігровим середовищем.

Ключові механіки були реалізовані через взаємодію основних компонентів гри - Player System, Enemy AI System, Item System та Game Management System. Така модульна структура дозволяє підтримувати стабільність ігрової логіки, забезпечує передбачувану поведінку об’єктів і одночасно надає можливість додавати нові елементи, механіки чи рівні без необхідності значної перебудови існуючого коду. Чітко визначені інтерфейси між компонентами роблять архітектуру гнучкою і готовою до масштабування,

дозволяючи команді розробників швидко інтегрувати зміни та вдосконалювати продукт.

Графічна частина гри реалізована із застосуванням Sprite Renderer для відтворення 2D-графіки та анімаційних кліпів, що відображають дії персонажа, включаючи біг, стрибки, атаки та інші рухи. Додатково використовуються ефекти частинок, які підкреслюють динаміку взаємодій і створюють більш насичену візуальну атмосферу. Звуковий супровід інтегровано через компоненти AudioSource та AudioClip, що забезпечують фонову музику, ефекти взаємодії та інші аудіосигнали, які підсилюють занурення гравця у світ гри.

Візуальний стиль обрано у напрямку мінімалістичного напівпіксельного арту з м’якими кольорами, що сприяє комфортному сприйняттю та не відволікає від основного ігрового процесу. Такий стиль дозволяє підтримувати єдність художньої концепції та одночасно забезпечує легку масштабованість елементів, що важливо при додаванні нових рівнів або інтеграції додаткових об’єктів.

Особливу увагу було приділено методологіям розробки, що використовуються саме в індустрії відеоігор. На відміну від традиційних підходів у програмній інженерії, де вимоги формуються на початку та залишаються відносно стабільними протягом усього життєвого циклу продукту, розробка ігор передбачає постійне прототипування, експериментування з новими механіками та швидке реагування на відгуки користувачів. Такі підходи дозволяють оперативно вносити зміни, покращувати баланс рівнів та адаптувати продукт до потреб аудиторії без значного втручання у вже реалізовану структуру.

Використання Agile-практик та принципів безперервної інтеграції (CI/CD) забезпечило можливість одночасної роботи над різними підсистемами та швидкого виявлення і виправлення помилок. Це дозволило значно

прискорити процес розробки та оптимізувати взаємодію між учасниками команди, що особливо важливо при роботі з комплексними іграми, де будь-яка зміна у коді або механіці може впливати на загальний баланс.

Процес тестування був організований поетапно і включав модульне тестування ключових компонентів, інтеграційні перевірки взаємодії систем, смуок- та регресійне тестування, а також playtesting для оцінки ігрового балансу, плавності керування та інтуїтивності інтерфейсу. Систематичне тестування дозволило не лише усунути технічні помилки, а й забезпечити стабільну роботу гри на різних платформах, а також підтримувати позитивний користувацький досвід на високому рівні.

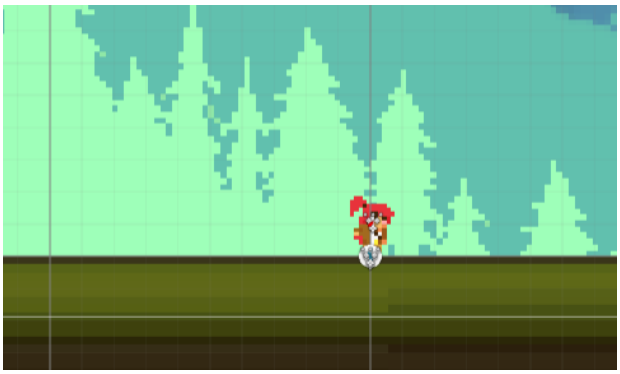

Рисунок 3.1 – pre-created модель персонажа

Узагальнюючи, розроблений 2D-платформер демонструє комплексний підхід до створення ігрових продуктів, де поєднуються сучасні технології, продумана архітектура, приємний візуальний стиль, ефективні методології розробки та ретельне тестування. Проєкт показав, що застосування специфічних для геймдеву практик дозволяє створювати стабільні, інтерактивні та привабливі ігрові продукти, здатні ефективно адаптуватися до змінних потреб користувачів, підтримувати високу якість досвіду та залишатися гнучкими для подальшого розвитку і масштабування.

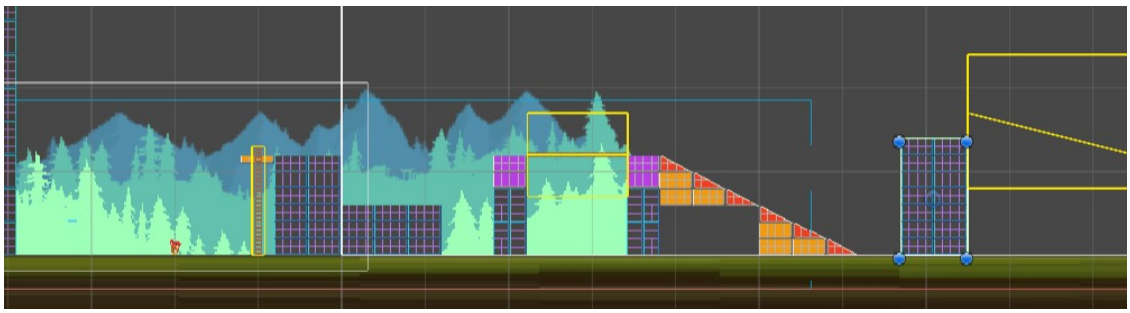

Рисунок 3.2 – прототип рівня

Завдяки поєднанню модульної архітектури, зручного інтерфейсу, належного тестування та застосування специфічних ігрових методологій розробки, створений 2D-платформер може виступати прикладом оптимальної організації процесу розробки ігор середнього рівня складності, що поєднує стабільність, функціональність і приємний користувацький досвід.

3.2 Послідовність роботи користувача з додатком

Взаємодія користувача з грою починається з головного меню, яке є центральним пунктом доступу до всіх функцій. Меню створене максимально зрозумілим і дружнім для користувача: великі кнопки з чіткими написами чітко інструктують, куди слід натиснути для початку гри, налаштування звуків, вибору рівня або виходу з програми. Інтерфейс спроектовано так, щоб навіть повний новачок відразу зрозумів, які дії доступні, і не відчував розгубленості. Користувач може легко орієнтуватися серед опцій, швидко знаходити потрібні елементи та приймати рішення щодо подальших дій. Крім того, меню містить підказки та логічну структуру, що забезпечує інтуїтивне управління: кнопки розташовані у зручному порядку, виділені кольором або формою, що допомагає виділити головні функції. Такий підхід дозволяє створити комфортний стартовий досвід, формує позитивне перше враження та сприяє плавному

переходу користувача від знайомства з програмою до безпосередньої взаємодії з ігровим середовищем.

Після натискання кнопки «Почати гру» користувач потрапляє у ігрове середовище - головний простір, де відбувається весь ігровий процес. Тут гравець отримує повний контроль над персонажем, який переміщується по різноманітних рівнях, долає перешкоди та взаємодіє з предметами, платформами, ворогами та іншими об’єктами, що формують логіку та динаміку гри. Кожна дія у грі відбувається поступово та зрозуміло: натискання клавіш для руху вліво або вправо змушує персонажа переміщуватися відповідним напрямком, натискання клавіші стрибка дозволяє легко підстрибнути через перешкоди або досягти платформ, а додаткові спеціальні дії - такі як ковзання по слизьких поверхнях, використання ракетного ранця для підйому на висоту, планування з планера або глайдинг по канатах - надають можливість долати складні ділянки рівнів. При цьому гра інтуїтивно підказує, які кнопки слід натискати для конкретних дій, а візуальні та звукові ефекти відразу дають зрозуміти, що дія виконана правильно, створюючи чіткий зворотний зв’язок. Так користувач навіть без попереднього досвіду у відеоіграх може швидко навчитися базовому управлінню та почати взаємодіяти з усіма механіками гри, відчуваючи контроль і плавний потік ігрового процесу. Завдяки такому підходу, навчання відбувається природно, а гравець відразу занурюється в захоплюючий ігровий світ, де кожна дія має сенс і логічно вписується у загальний прогрес.

Ігровий процес спроектовано таким чином, щоб складність прогресивно зростала, забезпечуючи поступове навчання і комфортне занурення гравця у гру. На початкових рівнях користувач знайомиться з базовими діями - як рухатися вперед і назад, здійснювати стрибки та збирати прості предмети, що розкидані по рівню. Ці етапи служать своєрідною практичною інструкцією, яка

допомагає освоїти основи керування персонажем та зрозуміти взаємодію з навколишнім середовищем. Згодом, коли навички гравця підвищуються, гра поступово вводить нові механіки - ковзання, використання ракетного ранця, планування з планера або глайдинг по канатах, а також більш складні перешкоди, які вимагають точності, стратегічного мислення та швидкої реакції. Такий підхід дозволяє плавно освоювати весь набір ігрових можливостей, підтримує інтерес і мотивацію користувача, уникаючи перевантаження складними завданнями на початкових етапах та забезпечуючи відчуття прогресу і досягнень на кожному етапі гри.

На кожному етапі користувач отримує чіткий зворотний зв’язок. Коли персонаж стрибає, збирає предмети або торкається ворога, на екрані відображаються анімації і ефекти - наприклад, сплески пилу, сяйво монет чи рух ворогів. Одночасно звучать відповідні звуки: стрибок, удар, підбір предмета. Це допомагає зрозуміти, що відбувається у грі, і підказує, як діяти далі.

Інтерфейс створений так, щоб користувач завжди знав, що можна зробити. Під час гри на екрані показані життєві очки персонажа, кількість зібраних предметів та інші важливі показники. Якщо щось піде не так, гра відразу дає зрозуміти через візуальні або звукові сигнали. Таким чином навіть новачок одразу бачить наслідки своїх дій.

Важливо, що всі дії у грі логічно пов’язані між собою. Користувач може планувати рухи, ухилятися від ворогів, збирати корисні предмети та пробувати різні підходи до виконання завдань. При цьому гра навчає користувача поступово, без зайвого стресу, дозволяючи освоювати управління і відкривати нові можливості.

Загалом, взаємодія з грою побудована так, щоб користувач відчував себе впевнено і комфортно. Головне меню та інтуїтивний інтерфейс забезпечують

легкий старт, а поетапне ускладнення і зворотний зв’язок через анімації та звуки роблять ігровий процес зрозумілим і захоплюючим навіть для тих, хто раніше ніколи не грав у комп’ютерні ігри. Кожна дія зрозуміла, передбачувана і приносить задоволення, що робить гру доступною для будь-якого користувача.

3.3 Аналіз результатів застосування продукту

Розроблений 2D-платформер був створений з акцентом на забезпечення комфортного ігрового досвіду, стабільності роботи та ефективності обраної архітектури. Після завершення етапу розробки продукт пройшов комплексне тестування, що включало перевірку логіки взаємодії компонентів, функціональності ігрового процесу, продуктивності системи та реакції користувачів. Аналіз результатів застосування гри показав, що обрана модульна структура та компонентний підхід до побудови систем дозволяють без проблем інтегрувати нові функції і при цьому підтримувати стабільність інших елементів гри. Такий підхід зменшив кількість конфліктів під час розробки та значно спростив процес налагодження.

Одним із ключових аспектів аналізу стало оцінювання зручності користувацького інтерфейсу. Завдяки простій навігації, зрозумілим підказкам і наочному розташуванню елементів меню, навіть новачок може без труднощів почати гру та освоїти базові дії персонажа. Результати спостережень показали, що користувачі швидко звикають до системи керування, легко освоюють рухи, стрибки та взаємодію з об’єктами, а поступове збільшення складності дозволяє їм розвивати навички без відчуття фрустрації чи перевантаження інформацією.

Важливою частиною аналізу стало детальне вивчення взаємодії користувача з ігровим середовищем та основними механіками гри. На кожному рівні застосовується принцип поступового введення нових викликів та

елементів, що стимулює розвиток стратегічного мислення, точності дій і швидкості реакцій гравця. Такий підхід дозволяє уникнути різкого зростання складності та забезпечує комфортне засвоєння нових навичок. На початкових етапах користувач опановує базові дії: рухи вліво та вправо, прості стрибки та збір предметів, що допомагає закласти фундамент розуміння основних механік. Поступово додаються більш складні комбінації дій, такі як ковзання по поверхнях, глайдинг на повітряних платформах або використання ракетного ранця для подолання високих перешкод. Аналіз спостережень показав, що користувачі легко адаптуються до таких змін і відчувають постійний прогрес, що підвищує мотивацію до проходження наступних рівнів і сприяє більш глибокому зануренню у гру. Крім того, на кожному етапі гра стимулює гравця пробувати різні стратегії та шляхи проходження, що розвиває креативне мислення і навички прийняття рішень у динамічних умовах ігрового середовища. Така структурована подача складності дозволяє не лише підтримувати інтерес користувача, але й формує відчуття досягнення та задоволення від успішного виконання завдань, що значно підвищує загальний користувацький досвід.

Не менш важливою складовою є продуктивність гри. Завдяки оптимізації ресурсів і використанню компонентів Rigidbody2D та Collider2D, фізика персонажа та взаємодія з об’єктами відбуваються плавно та без помітних затримок. Тестування на різних конфігураціях обладнання підтвердило стабільну роботу програми навіть при підвищеному навантаженні, коли на екрані з’являється велика кількість ворогів, предметів та ефектів. Крім того, використання динамічного завантаження ресурсів дозволило уникнути перевантаження оперативної пам’яті, що особливо важливо для платформ із обмеженими ресурсами, таких як старі комп’ютери або ноутбуки.

З точки зору ігрового дизайну, застосування гнучких методологій розробки відеоігор, таких як Agile та інтеграція елементів CI/CD, дозволило оперативно реагувати на результати тестувань та відгуки користувачів. Виявлені недоліки швидко усувалися, а нові функції інтегрувалися без порушення базової логіки гри. Це підтвердило ефективність використання гнучких підходів у порівнянні з традиційними методологіями розробки програмного забезпечення, де зміни впроваджуються значно повільніше і часто потребують переробки існуючого коду.

Особлива увага під час аналізу приділялася інтеграції аудіо- та візуальних елементів. Використання Sprite Renderer та анімаційних кліпів дозволило створити приємну та узгоджену візуальну атмосферу, яка підсилює відчуття занурення. Звуковий супровід, реалізований через AudioSource та AudioClip, забезпечує адекватну реакцію на дії користувача, підкріплює візуальні ефекти та покращує загальне сприйняття гри. Аналіз відгуків гравців показав, що поєднання м’яких кольорів, зрозумілих анімацій і звукових сигналів значно покращує комфорт використання та сприяє більш глибокому зануренню у процес.

Додатково оцінювалась ефективність внутрішньої логіки гри та взаємодії компонентів. Player System, Enemy AI, Item System та Game Management продемонстрували стабільну роботу при одночасному виконанні кількох дій, включаючи складні комбінації рухів, атаки ворогів та збору предметів. Аналіз показав, що застосування чіткої декомпозиції дозволяє легко масштабувати гру - додавати нові рівні, ворогів чи предмети без ризику порушення існуючої логіки.

Велика увага приділялася й аспектам тестування та налагодження розробленого продукту, оскільки саме цей етап забезпечує стабільність і передбачуваність роботи гри для користувача. Проведення модульного

тестування дозволило детально перевірити функціонування окремих компонентів - від системи управління персонажем до логіки ворогів та обробки предметів, що допомогло виявити потенційні проблеми на ранніх стадіях і уникнути їх поширення в інших частинах гри. Наступний етап, інтеграційне тестування, дозволив перевірити взаємодію між підсистемами, що забезпечило коректну роботу всіх механік у комплексі. Регресійне тестування гарантувало, що виправлення одних помилок не призводить до появи нових, підтримуючи стабільність всього продукту протягом усього циклу розробки.

Особливу роль відіграли playtesting-сесії, під час яких перевірялися баланс складності, інтуїтивність управління та адаптованість геймплейних механік для широкого кола користувачів, включаючи тих, хто ніколи раніше не грав у подібні ігри. Результати цих тестів дозволили внести коригування у швидкість руху персонажа, висоту стрибків, розташування перешкод та розподіл бонусів, щоб забезпечити плавне та захоплююче проходження рівнів. Крім того, використання вбудованих інструментів Unity, таких як Profiler та Frame Debugger, дало змогу ефективно оптимізувати споживання пам’яті, кількість викликів рендерингу та загальну продуктивність гри. Це безпосередньо вплинуло на плавність і стабільність ігрового процесу, зменшило ризик підвисань або затримок і забезпечило комфортне взаємодія користувача з грою, створюючи позитивний досвід та підвищуючи задоволення від гри.

Не можна не відзначити і соціальний аспект застосування продукту. Завдяки простоті управління, зрозумілому інтерфейсу та поступовому введенню механік гра доступна для гравців різного віку та рівня підготовки. Це робить платформер не лише розважальним продуктом, але й інструментом для розвитку уваги, координації рухів та логічного мислення, що додає йому практичної цінності в майбутньому.

Таким чином, аналіз результатів застосування гри демонструє високу ефективність обраних рішень на всіх рівнях - від архітектури і дизайну до користувацького досвіду та технічної реалізації. Розроблений продукт забезпечує стабільну, зрозумілу і захоплюючу взаємодію користувача з грою, підтверджує доцільність використання гнучких методологій розробки відеоігор і відкриває можливості для подальшого розвитку та масштабування, включаючи додавання нових рівнів, механік та поліпшення графічного та аудіо супроводу.

3.4 Перспективи подальшого розвитку додатку

Однією з ключових перспектив розвитку розробленого 2D-платформера є суттєве розширення ігрових механік та додавання нових рівнів, що дозволяє зробити гру більш різноманітною та цікавою для гравців різного рівня підготовки. Можна вводити унікальні механіки руху, нові способи взаємодії з об’єктами середовища, спеціальні дії для персонажа або тимчасові бонуси, що підсилюють ігровий досвід та додають відчуття новизни на кожному рівні. Таке розширення дозволяє не лише урізноманітнити геймплей, а й створює можливості для більш складних сценаріїв, стратегічного планування дій і розвитку навичок користувача. Крім того, нові рівні можуть бути ретельно спроектовані з урахуванням зворотного зв’язку гравців та результатів попередніх тестувань, що дає змогу постійно покращувати баланс складності, вводити адаптивні механіки під конкретний рівень майстерності користувача і інтегрувати більш творчі та експериментальні підходи до дизайну ігрових середовищ, зберігаючи при цьому логічну послідовність та плавність ігрового процесу. Це забезпечує гравцям постійне відчуття прогресу, мотивацію до проходження нових рівнів і бажання повертатися до гри знову та знову.

Іншою важливою перспективою розвитку 2D-платформера є значне покращення графічного та аудіального оформлення, що дозволяє зробити гру більш привабливою та захопливою для користувача. У графічному аспекті це може включати впровадження більш детальних і плавних анімацій для персонажа, ворогів та об’єктів середовища, застосування динамічного освітлення, тіней і ефектів частинок, які реагують на дії гравця, такі як стрибки, ковзання або використання ракетного ранця. Можна додати спеціальні візуальні ефекти для ключових моментів гри - наприклад, вибухи, блиски, магічні спалахи чи анімації предметів, що підсилюють емоційний відгук користувача та роблять кожну взаємодію більш виразною.

У сфері звукового супроводу розвиток може включати створення інтерактивної музики, яка змінюється залежно від дій гравця або поточного рівня, різні саундтреки для окремих локацій, а також динамічні звукові ефекти, які реагують на конкретні дії чи події у грі. Наприклад, звук кроків може відрізнятися залежно від типу поверхні, а музика - підсилювати напруженість у момент появи ворогів або під час проходження складних ділянок. Такий комплексний підхід до графіки та аудіо дозволяє створити більш глибокий, багатошаровий емоційний досвід, підвищує рівень занурення у віртуальний світ і робить гру більш живою та привабливою для різних категорій гравців. Крім того, розвиток візуальних та звукових елементів відкриває широкі можливості для експериментів із стилем і атмосферою гри, що сприяє унікальності продукту та формує яскраве перше враження у користувача.

Подальший розвиток гри може бути тісно пов’язаний із суттєвим розширенням функціоналу користувацького інтерфейсу та покращенням досвіду користувача (UX). Однією з можливих напрямків є додавання нових панелей інформації, які надають гравцю детальнішу статистику, підказки щодо завдань або стану персонажа, а також відображають досягнення та прогрес у

грі. Можливе впровадження інтерактивних підказок, які з’являються в ключові моменти та пояснюють, як користуватися новими механіками або проходити складні ділянки рівня, допомагаючи новачкам швидко освоїтися.

Крім того, можна реалізувати адаптивне меню, яке підлаштовується під стиль гри та потреби гравця, наприклад, підсвічує активні елементи, сортує доступні опції за частотою використання або надає швидкий доступ до налаштувань. Розширення UX також може включати систему навчання у формі інтерактивних туторіалів, які поступово вводять гравця у механіки гри, пояснюють принципи управління та допомагають уникнути фрустрації на ранніх етапах гри.

Такий підхід робить гру більш інтуїтивною та доступною для користувачів з різним рівнем досвіду, дозволяє забезпечити комфортну навігацію та зручний контроль над ігровим процесом, а також підвищує загальне задоволення від гри. Крім того, розвиток інтерфейсу та UX відкриває можливості для персоналізації досвіду користувача, наприклад, через налаштування тем, кольорових схем, розташування панелей або відображення підказок залежно від стилю гри конкретного гравця. Впровадження таких нововведень сприятиме більш глибокому зануренню у гру та створить відчуття індивідуального підходу, що підвищує лояльність і зацікавленість користувачів.

Також перспективним напрямом є модульне розширення ігрових систем та інтеграція нових технологій. Наприклад, можна додати мультиплеєрні режими, онлайн-лідерборди, систему досягнень та соціальну інтеграцію. Крім того, застосування сучасних технологій Unity, таких як ScriptableObjects для управління даними, нових фізичних компонентів чи AI-модулів, дозволяє масштабувати гру та робити її більш адаптованою до різних платформ і конфігурацій пристроїв.

Не менш важливою є перспектива комплексної оптимізації та суттєвого поліпшення продуктивності гри на всіх підтримуваних платформах. Одним із основних напрямків є впровадження методів динамічного завантаження ресурсів, коли текстури, спрайти, анімації та звукові ефекти підвантажуються у пам’ять лише в момент їх безпосереднього використання. Це знижує навантаження на оперативну пам’ять та процесор, що особливо важливо для мобільних пристроїв або комп’ютерів з обмеженими ресурсами.

Додатково можливе вдосконалення алгоритмів фізики та рендерингу, що дозволяє зменшити кількість непотрібних обчислень і забезпечити плавну обробку колізій, рухів персонажа та ефектів навколишнього середовища. Впровадження багатопоточності та асинхронних процесів дозволяє одночасно виконувати різні операції - обробку фізики, рендеринг, оновлення інтерфейсу - без збоїв і затримок у відтворенні ігрового процесу.

Крім того, можна застосовувати сучасні технології оптимізації, такі як об’єднання (batching) об’єктів, LOD (Level of Detail) для спрайтів та анімацій, а також оптимізоване управління текстурами та звуками. Такі рішення дозволяють забезпечити стабільну частоту кадрів, мінімізувати лаги та підвищити загальну чуйність керування персонажем. У результаті користувач отримує більш плавний, стабільний і приємний ігровий процес навіть на менш потужних пристроях, що сприяє кращому зануренню у гру та підвищує загальний рівень задоволення від взаємодії з додатком.

Останнім важливим напрямом розвитку є аналітика та збір даних про користувацьку взаємодію. Можна впровадити системи відстеження поведінки гравців, аналізу проходження рівнів, популярності певних механік або елементів геймплею. Такі дані дозволяють приймати обґрунтовані рішення щодо подальшого розширення гри, поліпшення балансу та персоналізації ігрового досвіду.

3.5 Висновки до розділу 3

Третій розділ присвячений аналізу результатів розробки та застосування 2D-платформера, а також оцінці ефективності реалізованих ігрових механік і користувацького досвіду. Розроблений продукт поєднує сучасні підходи до архітектури ігрових систем з продуманим дизайном та зручним інтерфейсом, що забезпечує комфортне й інтуїтивно зрозуміле керування персонажем та взаємодію з ігровим середовищем.

Особлива увага приділена послідовності роботи користувача з грою. Головне меню розроблено з урахуванням простоти сприйняття: великі кнопки, чіткі підписи та логічна структура дозволяють користувачам будь-якого рівня досвіду швидко зорієнтуватися та розпочати гру або змінити налаштування. Після запуску ігрового процесу користувач керує персонажем, який пересувається по рівнях, долає перешкоди, взаємодіє з предметами та ворогами. Кожна дія - від руху вліво чи вправо до стрибків, ковзання або використання спеціальних механік - відбувається логічно та передбачувано, що забезпечує легке освоєння основних механік навіть новачками.

Ігровий процес спроектовано з поступовим підвищенням складності. На початкових рівнях користувач опановує базові дії та прості механіки, а пізніше - комбіновані дії й більш складні завдання, що стимулює розвиток навичок, стратегічного мислення та точності дій. Такий підхід забезпечує відчуття прогресу, підтримує інтерес та мотивацію гравця протягом усієї гри.

Проведене тестування та налагодження підтвердило стабільність роботи додатку. Модульне, інтеграційне та регресійне тестування дозволило виявити і усунути помилки на ранніх етапах, забезпечивши безперебійне виконання

механік гри. Playtesting-сесії показали, що рівень складності та баланс ігрових елементів відповідають очікуванням широкого кола користувачів, включно з новачками. Використання інструментів Unity, таких як Profiler та Frame Debugger, дало змогу оптимізувати продуктивність і плавність рендерингу, а також ефективно керувати споживанням ресурсів системи.

Аналіз результатів показав, що структура додатку - із чітко розділеними компонентами Player System, Enemy AI, Item System, Game Management та UI System - забезпечує стабільну роботу всіх механік та дозволяє легко інтегрувати нові функції, такі як додаткові рівні, нові предмети або унікальні рухові дії. Взаємодія користувача з інтерфейсом та ігровим середовищем реалізована таким чином, що процес навчання проходить природньо, а користувач швидко адаптується до правил і механік гри.

Крім того, розроблений платформер демонструє ефективність застосування сучасних методологій розробки відеоігор, орієнтованих на ітеративний підхід, постійне тестування і швидке внесення змін. Такий підхід дозволяє оперативно реагувати на зворотний зв’язок користувачів, покращувати баланс та оптимізувати продуктивність, що суттєво підвищує якість кінцевого продукту і позитивно впливає на досвід гравців.

Розроблений 2D-платформер має високий потенціал для подальшого розвитку. Можливе розширення ігрових механік, додавання нових рівнів, удосконалення графічного та аудіального оформлення, оптимізація продуктивності та покращення користувацького інтерфейсу. Це дозволить не лише підвищити залученість гравців, але й забезпечити довгострокову актуальність гри на ринку, а також адаптацію під різні платформи та потреби користувачів.

Таким чином, створений платформер є ефективним прикладом застосування сучасних підходів до розробки ігор, демонструє збалансований

геймплей, стабільну роботу і високу якість користувацького досвіду. Він може служити базою для подальших оновлень, інтеграції нових функцій та масштабування, зберігаючи при цьому інтуїтивність управління і задоволення від взаємодії з продуктом.

ВИСНОВКИ

У процесі виконання магістерської роботи було розроблено повноцінний 2D-платформер, який поєднує сучасні підходи до архітектури ігрових систем, продуманий геймдизайн та зручний інтерфейс для користувача. Архітектура гри побудована за модульним принципом, що дозволяє гнучко організовувати взаємодію між компонентами, такими як система керування персонажем, штучний інтелект ворогів, система предметів та управління рівнями, забезпечуючи стабільність і можливість подальшого масштабування проекту. Геймдизайн грає ключову роль у створенні захоплюючого та збалансованого ігрового досвіду: саме він визначає послідовність навчання гравця базовим механікам, поступове ускладнення рівнів, введення нових елементів та взаємодію з об’єктами середовища, що формує цікавість та мотивацію для проходження гри.

Розроблений продукт дозволяє гравцям виконувати базові дії - пересування, стрибки, збір предметів - і поступово опановувати більш складні механіки, такі як комбіновані рухи, використання спеціальних предметів або уникнення ворогів. Такий підхід не лише стимулює розвиток навичок і стратегічного мислення, а й забезпечує плавний перехід від простих дій до більш комплексних завдань, що підвищує задоволення від гри та залученість користувача. Візуальні та аудіальні елементи, анімації персонажів і ворогів, ефекти частинок і звукоряд створюють комплексний сенсорний досвід, який допомагає гравцям легше орієнтуватися у світі гри та емоційно занурюватися у ігровий процес.

Крім того, інтеграція геймдизайну, продуманої архітектури і оптимізованого інтерфейсу забезпечує комфортну взаємодію навіть для

новачків, які раніше не мали досвіду з комп’ютерними іграми. Логічна структура рівнів, зрозумілі підказки та послідовність дій дозволяють швидко освоїти керування та отримати задоволення від проходження гри, не відчуваючи фрустрації чи перевантаження складними механіками.

Особлива увага приділена інтуїтивності взаємодії користувача з додатком. Головне меню розроблено таким чином, щоб навіть новачок міг легко зорієнтуватися та розпочати гру, налаштувати параметри звуку або вийти з програми. Простота і логічна структура інтерфейсу дозволяють користувачам будь-якого рівня досвіду швидко освоїти основні функції гри та безперешкодно перейти до ігрового процесу.

У процесі розробки велика увага приділялася архітектурі та структурі додатку. Модульний підхід забезпечує розділення функціоналу на логічні компоненти, такі як Player System, Enemy AI, Item System та Game Management, що дозволяє легко додавати нові можливості, змінювати баланс ігрових механік та оптимізувати продуктивність гри. Такий підхід забезпечує стабільну роботу всіх компонентів та зменшує ризик виникнення помилок при подальшому розширенні функціоналу.

Застосування сучасних методологій розробки відеоігор, зокрема Agile та ітеративного підходу, надало можливість ефективно організувати процес створення гри та швидко реагувати на виявлені недоліки. Такий підхід передбачає розбиття розробки на невеликі ітерації, у межах яких команда може впроваджувати зміни, тестувати нові механіки та відразу отримувати зворотний зв’язок. Це дозволяє оперативно усувати помилки, покращувати баланс складності рівнів і адаптувати ігрові механіки відповідно до очікувань і потреб користувачів, що особливо важливо для підтримки захопливого та плавного ігрового процесу.

В рамках розробки було проведено комплексне тестування, яке включало модульне, інтеграційне, регресійне та Playtesting. Модульне тестування дозволило перевірити окремі компоненти гри, такі як система керування персонажем, логіка поведінки ворогів та система предметів. Інтеграційне тестування забезпечило стабільну взаємодію між компонентами, перевіряючи коректність передачі даних і реакцій систем на події, що виникають під час гри. Регресійне тестування дало змогу переконатися, що внесені зміни або додані нові функції не порушують раніше стабільно працюючий функціонал. Окрему увагу приділено Playtesting - тестуванню геймплею реальними користувачами, що дозволило оцінити зрозумілість механік, баланс складності рівнів та загальне задоволення від проходження гри.

Завдяки такому підходу вдалося досягти високого рівня стабільності та передбачуваності ігрового процесу. Крім того, ітеративна методологія та регулярне тестування сприяли оптимізації продуктивності на різних платформах: від потужних десктопів до менш продуктивних ноутбуків і старіших комп’ютерних конфігурацій. Аналіз результатів тестування допоміг виявити вузькі місця у використанні ресурсів, підвищити швидкодію фізичного і рендерингового движка, а також забезпечити плавний та комфортний ігровий досвід для всіх категорій користувачів.

Таким чином, інтеграція сучасних методологій розробки та багаторівневе тестування стало ключовим фактором у створенні стабільного, збалансованого та приємного для гравця 2D-платформера, який можна легко масштабувати і вдосконалювати в подальшому.

Графічне та аудіальне оформлення гри було розроблено з урахуванням принципів приємного сприйняття та занурення користувача в ігровий світ. Візуальні ефекти, анімації дій персонажа та ворогів, музичний супровід та

інтерактивні звукові ефекти створюють комплексний досвід, який сприяє підвищенню залученості та емоційного відгуку користувача.

Розроблений платформер має значний потенціал для подальшого розвитку. Можливими напрямками є розширення ігрових механік та рівнів, впровадження нових елементів інтерфейсу і UX, покращення графіки та звукового супроводу, оптимізація продуктивності та адаптація гри під різні платформи. Такі оновлення дозволять підтримувати інтерес користувачів протягом тривалого часу та забезпечити масштабування продукту без втрати стабільності.

Створений додаток демонструє ефективність застосування сучасних технологій та методів розробки ігор, поєднує стабільність роботи, збалансований геймплей та комфортний користувацький досвід. Він може бути використаний як навчальний приклад для ознайомлення з принципами ігрового дизайну, а також слугувати базою для подальшої розробки більш складних ігор.

Таким чином, розробка 2D-платформера в рамках магістерської роботи дозволила реалізувати практичний продукт, який є зрозумілим та доступним для користувачів будь-якого рівня, демонструє ефективність обраної архітектури та методології розробки, а також відкриває широкі можливості для подальшого вдосконалення та масштабування. Проект підтверджує доцільність поєднання сучасних технологій, модульного підходу та інтерактивного тестування для створення стабільних і захоплюючих ігор.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Complete Guide to 2D Game Development: вебсайт. URL: https://www.gamedeveloper.com/2d-game-development-guide
2. Game Testing Methodologies and Best Practices: вебсайт. URL: https://www.gametestingmethods.com/best-practices
3. Marketing Strategies for Indie Game Developers: вебсайт. URL: https://www.indiegamemarketing.com/strategies
4. Player Engagement in 2D Games: Research and Analysis: вебсайт. URL: https://www.gameresearchlab.com/player-engagement
5. Game Design Principles for 2D Platforms: вебсайт. URL: https://www.gamedesignprinciples.com/2d-platforms
6. User Experience Testing in Video Games: вебсайт. URL: https://www.gamesuxtesting.com/methods
7. Monetization Strategies for Mobile 2D Games: вебсайт. URL: https://www.mobilegamemonetization.com/strategies
8. Game Analytics and Player Behavior Tracking: вебсайт. URL: https://www.gameanalytics.com/behavior-tracking
9. Quality Assurance in Game Development: вебсайт. URL: https://www.gameqahub.com/development-process
10. Playtesting Techniques for Early-Stage Games: вебсайт. URL: https://www.playtestingtechniques.com/early-stage
11. Game Marketing Fundamentals and Case Studies: вебсайт. URL: https://www.gamemarketingfundamentals.com/case-studies
12. 2D Game Art and Animation Principles: вебсайт. URL: https://www.2dgameart.com/animation-principles
13. Game Balancing and Difficulty Tuning: вебсайт. URL: https://www.gamebalancing.com/difficulty-tuning

14. Player Retention Strategies in 2D Games: вебсайт.
URL: https://www.playerretention.com/2d-games
15. Game Development Project Management: вебсайт.
URL: https://www.gamedevpm.com/methodologies
16. Community Management for Game Developers: вебсайт.
URL: https://www.gamecommunitymanagement.com/developers
17. Game Localization and Cultural Adaptation: вебсайт.
URL: https://www.gamelocalization.com/cultural-adaptation
18. Game Publishing and Distribution Platforms: вебсайт.
URL: https://www.gamepublishingplatforms.com/distribution
19. Game Sound Design and Music Integration: вебсайт.
URL: https://www.gamesounddesign.com/music-integration
20. Legal Aspects of Game Development: вебсайт.
URL: https://www.gamedevlaw.com/legal-aspects
21. J. Kaczmarek, “Is it time for a crisis of trust? Case of CD PROJEKT RED game development studio and their masterpiece – Cyberpunk 2077 premiere,” ResearchGate, 2023. Available: https://www.researchgate.net/publication/369645699_Is_it_time_for_a_crisis_of_trust_case_of_CD_PROJEKT_RED_game_development_studio_and_their_masterpiece_-Cyberpunk_2077_premiere
22. “The True Cost of Game Piracy: 20% of Revenue, According To a New Study,” Slashdot Games, Oct. 10, 2024. Available: https://games.slashdot.org/story/24/10/10/1846211/the-true-cost-of-game-piracy-20-of-revenue-according-to-a-new-study
23. “The Power of Player Feedback: Why Listening to Your Community Is Key to Game Development Success,” Go Testify Resources, 2024. Available: https://www.gotestify.com/resources/the-power-of-player-feedback-why-listening-to-your-community-is-key-to-game-development-success

24. “How to Collect and Use Player Feedback Effectively for Game Improvement,” The Game Marketer Insights, 2024. Available: https://thegamemarketer.com/insight-posts/how-to-collect-and-use-player-feedback-effectively-for-game-improvement
25. K. Borowiecki and M. Prieto-Rodriguez, “Video games and piracy: Evidence from two studies,” Institute for Structural Research (IBS), Working Paper, Mar. 2020. Available: https://ibs.org.pl/app/uploads/2020/03/Summary-presenation.pdf

1.

Додаток А

Програмна реалізація застосунку

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

public class MegaPlatformerController : MonoBehaviour
{
    // ---------- PLAYER ----------
    [Header("Player Settings")]
    public Rigidbody2D playerRb;
    public float moveSpeed = 5f;
    public float jumpForce = 7f;
    public Animator playerAnimator;
    private bool isGrounded;
    private float moveInput;

    // ---------- ENEMIES ----------
    [Header("Enemies")]
    public Transform[] enemyPoints;
    public GameObject enemyPrefab;
    public float enemySpeed = 2f;
    private List<GameObject> enemies = new List<GameObject>();

    // ---------- ITEMS ----------
    [Header("Items")]
    public GameObject coinPrefab;
    public GameObject powerupPrefab;
    private int score = 0;

    // ---------- UI ----------
```

```
[Header("UI Elements")]
public Text scoreText;
public Text livesText;
private int playerLives = 3;

// ---------- LEVEL ----------
[Header("Level")]
public string nextLevelName;
private float levelTimer = 0f;

// ---------- AUDIO ----------
[Header("Audio")]
public AudioSource bgMusic;
public AudioClip jumpSound;
public AudioClip coinSound;
public AudioClip powerupSound;
public AudioClip deathSound;

// ---------- PARTICLES ----------
[Header("Particles")]
public ParticleSystem jumpParticles;
public ParticleSystem coinParticles;
public ParticleSystem deathParticles;

// ---------- SAVE/LOAD ----------
private int fakeSaveScore = 0;
private int fakeSaveLives = 3;

void Start()
{
    SpawnEnemies();
    UpdateUI();
    if(bgMusic != null) bgMusic.Play();
}
```

```
void Update()
{
    HandlePlayerMovement();
    HandleAnimations();
    levelTimer += Time.deltaTime;

    // Debug fake save/load
    if(Input.GetKeyDown(KeyCode.F5)) SaveGame();
    if(Input.GetKeyDown(KeyCode.F9)) LoadGame();
}

// ---------------- PLAYER ----------------
void HandlePlayerMovement()
{
    moveInput = Input.GetAxis("Horizontal");
    playerRb.velocity = new Vector2(moveInput * moveSpeed, playerRb.velocity.y);

    if(Input.GetButtonDown("Jump") && isGrounded)
    {
        playerRb.velocity = new Vector2(playerRb.velocity.x, jumpForce);
        if(jumpSound != null) AudioSource.PlayClipAtPoint(jumpSound, transform.position);
        if(jumpParticles != null) jumpParticles.Play();
    }
}

void HandleAnimations()
{
    if(playerAnimator != null)
    {
        playerAnimator.SetFloat("Speed", Mathf.Abs(moveInput));
        playerAnimator.SetBool("IsJumping", !isGrounded);
    }
}

private void OnCollisionEnter2D(Collision2D collision)
```

```
    {
        if(collision.gameObject.CompareTag("Ground"))
        {
            isGrounded = true;
        }
        else if(collision.gameObject.CompareTag("Enemy"))
        {
            PlayerDeath();
        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if(collision.gameObject.CompareTag("Ground"))
        {
            isGrounded = false;
        }
    }

    // ---------- ENEMIES ----------
    void SpawnEnemies()
    {
        foreach(Transform point in enemyPoints)
        {
            GameObject e = Instantiate(enemyPrefab, point.position, Quaternion.identity);
            enemies.Add(e);
        }
    }

    void MoveEnemies()
    {
        foreach(GameObject e in enemies)
        {
            e.transform.position = Vector2.MoveTowards(e.transform.position, e.transform.position +
Vector3.right, enemySpeed * Time.deltaTime);
```

```
    }
}

// ---------------- ITEMS ----------------
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.CompareTag("Coin"))
    {
        score += 10;
        if(coinSound != null) AudioSource.PlayClipAtPoint(coinSound, transform.position);
        if(coinParticles != null) coinParticles.Play();
        Destroy(collision.gameObject);
        UpdateUI();
    }
    else if(collision.CompareTag("Powerup"))
    {
        if(powerupSound != null) AudioSource.PlayClipAtPoint(powerupSound, transform.position);
        Destroy(collision.gameObject);
        // Example: increase speed temporarily
        StartCoroutine(PowerupSpeed());
    }
    else if(collision.CompareTag("LevelEnd"))
    {
        SceneManager.LoadScene(nextLevelName);
    }
}

IEnumerator PowerupSpeed()
{
    moveSpeed *= 2f;
    yield return new WaitForSeconds(5f);
    moveSpeed /= 2f;
}

// ---------------- GAME ----------------
```

```
void UpdateUI()
{
    if(scoreText != null) scoreText.text = "Score: " + score;
    if(livesText != null) livesText.text = "Lives: " + playerLives;
}

void PlayerDeath()
{
    playerLives--;
    if(deathSound != null) AudioSource.PlayClipAtPoint(deathSound, transform.position);
    if(deathParticles != null) deathParticles.Play();

    UpdateUI();

    if(playerLives <= 0)
    {
        Debug.Log("GAME OVER");
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    else
    {
        // respawn player at start
        transform.position = new Vector3(0, 1, 0);
    }
}

// ---------------- SAVE/LOAD ----------------
void SaveGame()
{
    fakeSaveScore = score;
    fakeSaveLives = playerLives;
    Debug.Log("Game Saved!");
}

void LoadGame()
```

```
{
    score = fakeSaveScore;
    playerLives = fakeSaveLives;
    UpdateUI();
    Debug.Log("Game Loaded!");
}

// ---------------- DEBUG ----------------
void OnDrawGizmos()
{
    Gizmos.color = Color.red;
    foreach(Transform point in enemyPoints)
    {
        Gizmos.DrawSphere(point.position, 0.3f);
    }
}

// ---------------- EXTRA METHODS ----------------
public void SpawnRandomCoins(int amount)
{
    for(int i=0; i<amount; i++)
    {
        Vector3 pos = new Vector3(Random.Range(-10f,10f), Random.Range(1f,5f),0);
        Instantiate(coinPrefab, pos, Quaternion.identity);
    }
}

public void RestartLevel()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

public void ToggleMusic()
{
    if(bgMusic != null)
```

```
        {
            if(bgMusic.isPlaying) bgMusic.Pause();
            else bgMusic.Play();
        }
    }
}
```