Хмельницький національний університет

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Квіта Владислава Леонідовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2201101.01.07.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Владислав КВІТ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник асистент _____ Анастасія ДЬОМІНА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер доцент _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

10 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Рівень вищої освіти Перший(бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Квіту Владиславу Леонідовичу
Прізвище, ім'я, по батькові студента

. Тема роботи «Ігровий застосунок-головоломка з процедурною енерацією рівнів та адаптивною складністю»
Керівник роботи Дьоміна Анастасія Іванівна асистент
Прізвище, ім'я, по батькові науковий ступінь, вчене звання

атверджена наказом ректора університету від
. Строк подання студентом проекту на кафедру 01.06.2026
. Вихідні дані до проекту Матеріали переддипломної практики
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ослідження предметної області та постановка задач, проєктування
рограмного забезпечення, програмна реалізація та тестування застосунку.

Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення
Структурна схема архітектури системи
Схема залежностей між модулями
Схема ігрового процесу

слайди 14шт.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультатнта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина О. М., доцент	05.05.26	05.05.26
Антиплагіат	Форкун Ю. В., доцент	05.05.26	05.06.26

7. Дата видачі завдання «02» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів(розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальної теми кваліфікаційної роботи (КвР)	01.12-31.12.2025	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог.	01.01-20.02.2026	
3 Проєктування програмного забезпечення	21.02-20.03.2026	
4 Програмна реалізація з використанням відповідних засобів розроблення. Тестування ПЗ	21.03-30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05-25.05.2026	
6 Попередній захист КвР	Травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05-30.05.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Владислав КВІТ
підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Анастасія ДЬОМІНА
підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Кваліфікаційна робота: «Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю».

Автор роботи: Квіт Владислав Леонідович

Керівник роботи: Дьоміна Анастасія Іванівна

Пояснювальна записка: 75 с., 21 рис., 8 табл., 3 додатки, 35 джерел.

Графічна частина: 3 креслення, слайди 14 шт.

ГОЛОВОЛОМКА, ІГРОВИЙ ЗАСТОСУНОК, C#, UNITY,

Метою роботи є розроблення ігрового застосунку, що реалізує механіку процедурної генерації рівнів та адаптації складності ігрового процесу відповідно до поведінки користувача.

У межах кваліфікаційної роботи проведено аналіз предметної області ігрових застосунків із елементами процедурної генерації та адаптивного геймплею, визначено основні вимоги до подібних систем та обґрунтовано вибір компонентної архітектури програмного забезпечення. Розроблено загальну архітектуру ігрового застосунку, спроєктовано структуру основних модулів системи та визначено принципи їх взаємодії через інтерфейси.

Для реалізації ігрового застосунку використано ігровий рушій Unity та мову програмування C#, що забезпечило можливість реалізації фізичної взаємодії об'єктів, процедурної генерації рівнів та системи адаптації складності. Додатково використано вбудовані механізми рушія для реалізації ігрової фізики та користувацького інтерфейсу.

У результаті виконання роботи здійснено програмну реалізацію ігрового застосунку, який забезпечує динамічний ігровий процес за рахунок процедурно згенерованих рівнів та адаптивної зміни складності. Розроблена система може бути використана як основа для подальшого розвитку ігрових застосунків подібного типу та розширення їх функціональності.

01.06.26 (дата)

(підпис)

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2201101.01.07.ПЗ	Пояснювальна записка	75		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ. 2201101.01.07.Е8	Структурна схема	1		
			архітектури системи			
5	А3	КвРІПЗ. 2201101.01.07.Е8	Схема залежностей між	1		
			модулями			
6	А3	КвРІПЗ. 2201101.01.07.Е8	Схема ігрового процесу	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2201101.01.07.ВД		
Виконав		Квіт. В. Л.		26.05	Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю	Літ.	Арк. 1 / Аркушів 1
Керівник		Дьоміна А. І.		26.05			
Н. контр.		Яшина О.М.		26.05	Відомість документів	ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		25.05			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	База даних
СКБД	–	Система керування базами даних
ПЗ	–	програмне забезпечення
FPS	–	Кадрів за секунду
UI	–	Користувацький інтерфейс
UML	–	Уніфікована мова моделювання
RAM	–	Оперативна пам’ять

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням складності програмних систем, розширенням сфер їх застосування та підвищенням вимог до якості програмного забезпечення. Особливо це стосується індустрії цифрових розваг, яка сьогодні є однією з найбільш динамічних та інноваційних галузей. Відеоігри вже давно вийшли за межі простого засобу розваги та стали складними інтерактивними системами, що поєднують у собі елементи штучного інтелекту, фізичного моделювання, комп'ютерної графіки та алгоритмічного аналізу даних.

У цьому контексті особливого значення набуває підготовка фахівців у сфері програмної інженерії, які здатні не лише використовувати готові технологічні рішення, а й проєктувати та реалізовувати власні програмні системи. Сучасний розробник програмного забезпечення повинен володіти навичками аналізу предметної області, проєктування архітектури системи, реалізації алгоритмів різної складності та забезпечення стабільної роботи програмного продукту в умовах реального використання.

Окрему роль у розвитку програмної інженерії відіграє напрямок розробки ігрових застосунків. Ігрові системи є складними багатокомпонентними продуктами, які потребують чіткої архітектури, ефективної взаємодії модулів та оптимізації продуктивності. Сучасні ігри все частіше використовують процедурну генерацію контенту, адаптивні алгоритми складності та фізичне моделювання для створення більш реалістичного та унікального користувацького досвіду.

Тема даної кваліфікаційної роботи присвячена розробці адаптивної фізичної головоломки з процедурною генерацією рівнів. Актуальність обраної теми зумовлена зростанням популярності інтерактивних ігор, у яких гравець взаємодіє з динамічним середовищем, а ігрові рівні не є статичними, а формуються алгоритмічно. Такий підхід дозволяє значно підвищити

реіграбельність гри, забезпечити унікальний досвід для кожного користувача та адаптувати складність відповідно до його навичок.

Особливістю сучасних фізичних головоломок є поєднання логічних задач із реалістичною фізичною моделлю. У таких іграх поведінка об’єктів підпорядковується законам механіки, що дозволяє створювати складні інтерактивні сценарії. Додавання процедурної генерації рівнів дозволяє автоматично створювати нові ігрові ситуації без необхідності ручного проєктування кожного рівня, що значно розширює можливості ігрового застосунку.

Ще одним важливим елементом сучасних ігрових систем є адаптивна складність. Вона дозволяє змінювати параметри гри залежно від поведінки користувача, його успішності та швидкості проходження рівнів. Це забезпечує баланс між складністю та доступністю гри, що є критично важливим для утримання інтересу гравця та формування позитивного ігрового досвіду.

Процедурна генерація рівнів є одним із ключових технологічних підходів, що застосовується у сучасній розробці ігор. Вона базується на використанні алгоритмів, які автоматично формують ігровий контент відповідно до заданих параметрів. Такий підхід дозволяє створювати унікальні рівні при кожному запуску гри, зменшує обсяг ручної роботи розробника та підвищує варіативність проходження. Водночас процедурна генерація вимагає додаткових механізмів перевірки коректності рівнів, зокрема їх прохідності та збалансованості.

У межах даної кваліфікаційної роботи передбачається створення програмного застосунку, який поєднує три основні складові: фізичне моделювання ігрових об’єктів, процедурну генерацію рівнів та систему адаптації складності. Така комбінація дозволяє створити гнучку ігрову систему, яка може змінювати свою поведінку залежно від дій користувача та забезпечувати різноманітний ігровий процес.

Метою роботи є проєктування та реалізація програмного забезпечення у вигляді адаптивної фізичної головоломки з процедурною генерацією рівнів, що забезпечує динамічний та унікальний ігровий досвід для користувача.

Для досягнення поставленої мети необхідно вирішити такі завдання: провести аналіз предметної області та існуючих рішень; визначити функціональні та нефункціональні вимоги до системи; спроєктувати архітектуру програмного забезпечення; реалізувати основні ігрові механіки; розробити систему процедурної генерації рівнів; впровадити механізм адаптації складності; а також провести тестування розробленого програмного продукту.

Об’єктом дослідження є процес розробки ігрових застосунків із використанням процедурної генерації та адаптивних алгоритмів. Предметом дослідження є методи та засоби реалізації адаптивної фізичної головоломки в середовищі сучасних ігрових рушіїв.

Практичне значення роботи полягає у створенні програмного продукту, який може бути використаний як демонстраційна модель інтерактивної гри з процедурною генерацією рівнів та адаптивною складністю, а також як основа для подальших досліджень у сфері ігрової розробки.

Таким чином, дана кваліфікаційна робота спрямована на поєднання сучасних підходів до розробки програмного забезпечення та ігрових технологій, що дозволяє створити гнучку, масштабовану та інтерактивну систему, орієнтовану на користувача.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Сучасна індустрія цифрових ігор є однією з найбільш динамічних галузей інформаційних технологій, яка активно інтегрує досягнення програмної інженерії, комп'ютерної графіки, математичного моделювання та штучного інтелекту. Згідно з сучасними дослідженнями ринку, відеоігри сьогодні виступають не лише засобом розваги, але й платформою для навчання, розвитку когнітивних здібностей та моделювання складних систем. У цьому контексті особливий інтерес становлять ігри жанру головоломок, що спрямовані на розвиток логічного мислення, просторового уявлення та аналітичних навичок користувачів.

Ігровий жанр – це сукупність ігор, об'єднаних спільними характеристиками ігрового процесу (геймплея). Зазвичай, основним критерієм в його описі є дія, яку виконує гравець, на відміну від жанру, наприклад, літературного твору. В іграх це поняття прийнято називати сеттінгом [1].

Пазл(так ще називають жанр головоломок) – це тип ігор, який фокусується на пошуку рішення, одночасно спонукаючи гравця критично мислити. Ці ігри часто вимагають від гравця використання логіки та навичок розв'язання проблем для виконання завдань або проходження рівнів. Головоломки перевіряють розпізнавання образів і базуються на візуальному сприйнятті, грі слів або їх комбінації [2].

Окремим піджанром є фізичні головоломки, у яких основою ігрового процесу виступає моделювання фізичних явищ. У таких іграх гравець взаємодіє з об'єктами, що підпорядковуються законам механіки: гравітації, інерції, імпульсу, пружності, тертя тощо. Прикладом успішної реалізації фізичних

механік є гра Angry Birds (рисунок 1.1). У зазначеному проєкті геймплей побудований навколо реалістичної взаємодії об’єктів під впливом фізичних сил.

Рисунок 1.1 – Angry Birds

Фізичний рушій у таких іграх виступає центральним компонентом архітектури, оскільки саме він відповідає за розрахунок траєкторій, обробку зіткнень, визначення стабільності конструкцій та симуляцію поведінки тіл у просторі. Від точності та продуктивності фізичного модуля безпосередньо залежить якість користувацького досвіду.

Штучний інтелект (ШІ) стає все більш важливим аспектом відеоігор, що впливає на все – від поведінки ворогів до генерації світу. У той час як традиційні ігри з ШІ покладаються на запрограмовані реакції та дерева прийняття рішень, адаптивний ШІ робить крок далі. Він дозволяє ігровим об’єктам, часто неігровим персонажам (NPC), навчатися й адаптувати свою поведінку на основі взаємодії з гравцями. Це створює більш динамічний та захопливий ігровий процес. Основна мета адаптивного ШІ – підтримувати оптимальний рівень складності, гарантуючи, що гра не буде ані занадто легкою, ані надто складною [3].

Одним із ключових інструментів реалізації адаптивності в іграх є процедурна генерація контенту. Процедурна генерація – потужний інструмент в арсеналі розробників ігор. Використовуючи алгоритми для динамічного створення контенту, процедурна генерація може створювати величезні, складні та різноманітні ігрові світи, які було б неможливо створити вручну. Ця техніка

не лише покращує реіграбельність, але й гарантує унікальність досвіду кожного гравця [4].

У фізичних головоломках процедурна генерація має додаткову складність, оскільки необхідно гарантувати розв'язуваність згенерованого рівня. Недостатньо лише створити структуру з об'єктів – потрібно перевірити її стабільність, можливість досягнення цільового стану та відповідність заданому рівню складності. Це вимагає використання алгоритмів валідації, симуляційного тестування та аналізу можливих сценаріїв розвитку подій.

З точки зору структурної організації, предметна область розробки адаптивної фізичної головоломки включає декілька взаємопов'язаних компонентів. Центральним елементом є ігровий рушій, який забезпечує життєвий цикл застосунку, обробку подій, рендеринг графіки та управління ресурсами. Поверх нього функціонує фізична підсистема, відповідальна за моделювання взаємодії об'єктів у режимі реального часу.

Окремим модулем виступає система процедурної генерації рівнів, яка формує конфігурацію сцени відповідно до заданих параметрів складності. Після генерації рівень передається до модуля перевірки, де здійснюється симуляція та оцінка коректності. Паралельно функціонує система адаптації складності, яка аналізує ігрову статистику (час проходження, кількість спроб, точність дій) та коригує параметри наступних рівнів.

Користувацький інтерфейс забезпечує взаємодію гравця з системою, відображення стану гри, результатів та підказок. Важливою складовою є модуль збору статистики, який акумулює дані про поведінку користувача для подальшого аналізу. Саме на основі цих даних формується профіль гравця, що використовується адаптивною системою.

У сучасній практиці розробки інтерактивних застосунків широко використовується середовище Unity, яке підтримує 2D- та 3D-графіку, фізичні симуляції та програмування мовою C#. Згідно з офіційною документацією Unity Technologies, фізична система рушія базується на інтеграції бібліотек для

моделювання твердих тіл та зіткнень, що дозволяє реалізовувати складні сценарії взаємодії об’єктів у режимі реального часу [5].

Функціонально система адаптивної фізичної головоломки повинна реалізовувати такі основні процеси: ініціалізацію ігрової сесії, генерацію рівня відповідно до профілю гравця, симуляцію фізичних процесів, обробку дій користувача, оцінювання результату та оновлення параметрів складності. Важливою вимогою є забезпечення стабільності роботи фізичного рушія при великій кількості об’єктів та мінімізація затримок обробки.

Таким чином, предметна область розробки адаптивної фізичної головоломки характеризується складною багаторівневою структурою та інтеграцією декількох технологічних напрямів: фізичного моделювання, алгоритмічної генерації контенту, аналізу поведінки користувача та програмної архітектури ігрових систем. Її особливістю є необхідність забезпечення балансу між реалістичністю фізики, коректністю генерації рівнів та індивідуалізацією складності. Саме поєднання цих компонентів формує функціональну основу проєкту та визначає його науково-практичну цінність.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

У предметній області на сучасному ринку існує кілька відомих реалізацій, які використовують елементи фізичного моделювання, адаптивної складності та автоматичного створення контенту. Аналіз цих проєктів дозволяє визначити їх сильні та слабкі сторони, а також окреслити ті можливості, які будуть реалізовані в межах даної кваліфікаційної роботи.

Одним із найвідоміших представників фізичних головоломок є гра Angry Birds. Angry Birds Classic – це гра, яка кидає виклик запустити різних птахів проти фортів і конструкцій, зроблених маленькими свинями. Мета полягає в тому, щоб знищити всіх свиней і викликати якомога більше руйнувань в

процесі. Доведеться добре прицілитися, розрахувавши силу пострілу, а потім запустити птаха. Завдяки гравітації він впаде на конструкцію і знесе деяких ворожих свиней.

У Angry Birds Classic є різні типи птахів, яких потрібно запустити, в яких є свої здібності, що допоможуть в проходженні рівня. Класичний червоний не має нічого особливого, але чорний птах вибухне, зелений птах може повернутися, як бумеранг тощо.

Angry Birds Classic пропонує сотні різних рівнів, в яких треба використовувати свій мозок і навички. Оцінка заснована на тризірковій системі, яка змушує грати на тих же рівнях, доки гравець не отримає всі три зірки [6]. На рисунку 1.2 зображено приклад рівня з цієї гри.

Рисунок 1.2 – приклад рівня з Angry Birds

Гра має такі позитивні аспекти:

– фізична симуляція об’єктів реалізована на високому рівні, що дозволяє отримувати передбачувані результати взаємодії;

– рівні побудовані таким чином, що гравець поступово опановує нові механіки.

Втім Angry Birds має суттєві обмеження щодо предметної області цієї роботи:

– усі рівні є жорстко спроєктованими вручну (немає процедурної генерації);

– не реалізовано механізм адаптації складності залежно від результатів гравця;

– процес гри не змінюється залежно від стилю гри користувача.

Таким чином, Angry Birds демонструє ефективне фізичне моделювання, але не реалізує ті елементи, які є ключовими для даного проєкту — процедурну генерацію та адаптивність.

Human Fall Flat – це весела та захоплююча гра-платформер, заснована на законах фізики. У цій грі до 4 людей можуть грати в фантастичних локаціях, а захоплену спільноту гравців постійно радують нові безкоштовні рівні. Кожний фантастичний рівень — це новий простір для пригод, від особняків, замків і земель Ацтеків до засніжених гір, похмурих нічних пейзажів и промислових зон. Безліч різних шляхів і захоплюючі пазли роблять дослідження рівнів ще цікавішим [8]. На рисунку 1.3 зображено приклад рівнів цієї гри.

Рисунок 1.3 – приклад рівнів у Human Fall Flat

Переваги:

– повністю 3D-фізика, яка інтерактивно реагує на дії гравця;

– висока варіативність рішень проблемних ситуацій.

Проте Human: Fall Flat не повністю відповідає предметній області цієї роботи:

– відсутня алгоритмічна генерація рівнів;

– адаптація геймплею базується на сценарних рішеннях, а не на поведінці користувача;

– гра сфокусована більше на кооперативному мультиплеєрі, ніж на індивідуальній адаптації складності.

На основі розглянутих прикладів можна зробити такі висновки у вигляді таблиці(Таблиця 1.1):

Таблиця 1.1 – Висновки на основі розглянутих ігор

Гра	Фізика	Процедурна генерація	Адаптивність	Логічна головоломка
Angry Birds	Так	Ні	Ні	Так
Human: Fall Flat	Так	Ні	Ні	частково

Як видно з таблиці, жодна з цих ігор не поєднує усі три ключові характеристики, які визначені предметною областю:

– Фізичне моделювання, що реалізується через відповідний рушій;

– Процедурна генерація рівнів;

– Адаптивна складність, що підлаштовується під поведінку гравця.

Проєкти, які розглядались, мають значні сильні сторони в окремих аспектах (наприклад, фізика або головоломка), але не реалізують повністю інтегровану систему, яка поєднувала б фізику, генерацію та адаптацію. Це створює підґрунтя для обґрунтування створення нового рішення – адаптивної фізичної головоломки з процедурною генерацією рівнів.

Обґрунтування переваг проєкту

Запропонований проєкт, на відміну від розглянутих ігор, має такі вагомі переваги:

– Інтеграція адаптивної складності. Система аналізує поведінку гравця та автоматично налаштовує параметри рівнів, що дозволяє підтримувати оптимальний рівень виклику.

– Автоматична генерація рівнів. Кожний рівень створюється алгоритмічно, що збільшує реіграбельність та індивідуальність досвіду.

– Баланс фізичних і логічних механік. Проєкт поєднує реалістичну фізику з логічними задачами, що створює новий формат головоломок.

– Використання Unity та C#. Ці технології забезпечують гнучкість реалізації, масштабованість та можливість подальшого розвитку проєкту.

1.3 Аналіз вимог до ігрового застосунку

Аналіз вимог до ігрового застосунку є одним із ключових етапів проєктування програмного забезпечення. Він визначає, що саме повинна робити система, яким чином вона взаємодіє з користувачем, які функції та обмеження має підтримувати. Для ігор це особливо важливо, оскільки вони поєднують інтерактивність, рівні складності, фізичні взаємодії та адаптацію поведінки під конкретного гравця. У контексті розробки адаптивної фізичної головоломки з процедурною генерацією рівнів вимоги до застосунку можуть бути розподілені на функціональні та нефункціональні (які визначають якість, продуктивність, масштабованість та зручність).

Функціональні вимоги описують поведінку системи, – перелік дій, які система має виконувати при виконанні певних умов, у відповідь на запити користувачів або при взаємодії з іншими системами. Дуже просто: «система робить дію X при виконанні умови Y».

У ширшому сенсі під функціональними вимогами розуміють набір можливостей системи, необхідних для підтримки бізнес-процесів. У цьому випадку фокус зміщується з окремих дій на сценарії використання та бізнес-функції, які система забезпечує [8].

До таких вимог належать:

– можливість завантаження рівня, згенерованого алгоритмом;

– підтримка механіки фізичного моделювання (гравітація, колізії, сили, інерція);

– система адаптації складності на основі поведінки гравця (аналіз часу проходження, кількості спроб, успішності);

– інтерфейс для вибору рівнів та відображення статистики прогресу;

– збереження стану гри та результатів гравця.

Ці вимоги узгоджуються з сучасними підходами до розробки ігор, де геймплей має бути гнучким та персоналізованим під користувача.

Однією з головних функціональних вимог є алгоритм процедурної генерації рівнів. Рівень повинен створюватися автоматично з урахуванням набору параметрів складності та фізичних характеристик об’єктів, а не бути статично прописаним у коді. Це дозволяє значно збільшити реіграбельність та варіативність ігрового процесу. Процедурна генерація рівнів є стандартною практикою у багатьох жанрах, включно з roguelike іграми, де кожна нова сесія має бути унікальною

Другий клас вимог стосується нефункціональних аспектів, які впливають на якість системи. Нефункціональні вимоги або NFR (non-functional requirements) – це набір специфікацій, які описують робочі можливості та обмеження системи та намагаються покращити її функціональність. Це в основному вимоги, які визначають, наскільки добре працюватиме продукт якщо врахувати, наприклад, швидкість, безпеку, надійність, цілісність даних тощо [9].

До них належать:

– продуктивність – гра повинна працювати без затримок при мінімальних системних вимогах;

– плавність фізичних симуляцій – фізичний рушій має забезпечувати стабільні та передбачувані результати;

– масштабованість – система має бути здатна адаптуватися до зміни параметрів складності та розміру рівня;

– надійність – відсутність критичних помилок під час проходження та збереження прогресу;

– зручність інтерфейсу – інтуїтивно зрозуміле меню, пояснення ходів, індикація помилок. Якість взаємодії з користувачем є критичною для підтримки інтересу та ігрового досвіду.

Далі важливо проаналізувати зв’язки між вимогами та компонентами системи. Для цього широко використовується UML-діаграма випадків використання (Use Case Diagram).

Діаграми варіантів використання допомагають фіксувати вимоги до системи та відображають поведінку системи в UML. Область застосування та функції високого рівня системи описані на діаграмах варіантів використання. Взаємодія між акторами та системами також зображена на цих діаграмах. Діаграми прецедентів показують, що робить система і як актори її використовують, але вони не показують, як система працює всередині. Крім того, контекст і вимоги до всієї системи або критичних компонентів системи проілюстровано та визначено за допомогою діаграм варіантів використання. Одна діаграма варіантів використання може представляти складну систему або кілька діаграм варіантів використання можуть представляти її компоненти. Діаграми варіантів використання часто створюються на ранніх стадіях проекту та використовуються як посилання протягом усього процесу розробки [10]. Така діаграма зображена на рисунку 1.4.

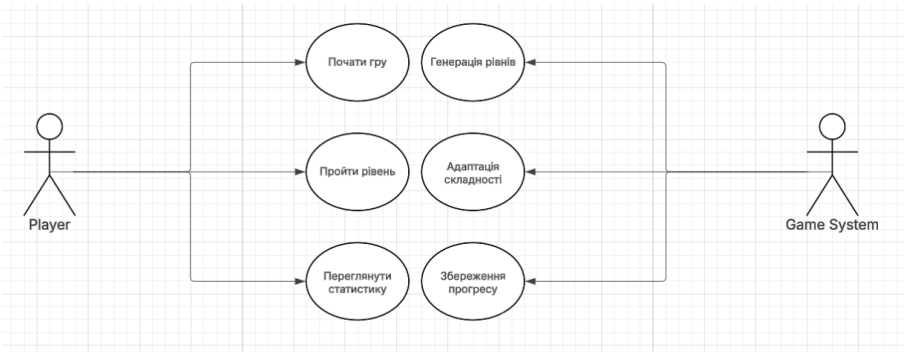


Рисунок 1.4 – UML-діаграма варіантів використання

На діаграмі зображено, як користувач(у ролі гравця) може взаємодіяти із застосунком, та як ігрова система може діяти для забезпечення правильної роботи застосунку. Гравець може почати гру, після чого система звернеться до необхідних компонентів для створення рівня. Також гравець може проходити самі згенеровані рівні, адаптованої складності або переглянути статистику, в яку записуються дані на основі проходження рівнів гравцем.

Іншим важливим елементом моделювання є UML-діаграма послідовності. Діаграма послідовності (Sequence Diagram) – використовуються для моделювання логіки сценаріїв використання, показуючи інформацію, що передається між об’єктами в системі під час виконання сценарію.

Діаграма послідовності (Sequence Diagrams) представляє інформацію в горизонтальному і вертикальному вирівнюванні. Об’єкти, які надсилають повідомлення один одному, зображуються у вигляді блоків, вирівняних у верхній частині сторінки зліва направо, причому кожен об’єкт займає стовпчик простору на сторінці, обмежений вертикальною лінією.

Повідомлення, які надсилаються від одного об’єкта до іншого, зображені у вигляді горизонтальних стрілок. Порядок повідомлень представлений у послідовності зверху вниз і зліва направо, починаючи з першого повідомлення у верхньому лівому кутку сторінки, а наступні повідомлення розташовуються праворуч і нижче [11].

Таку діаграму зображено на рисунку 1.5.

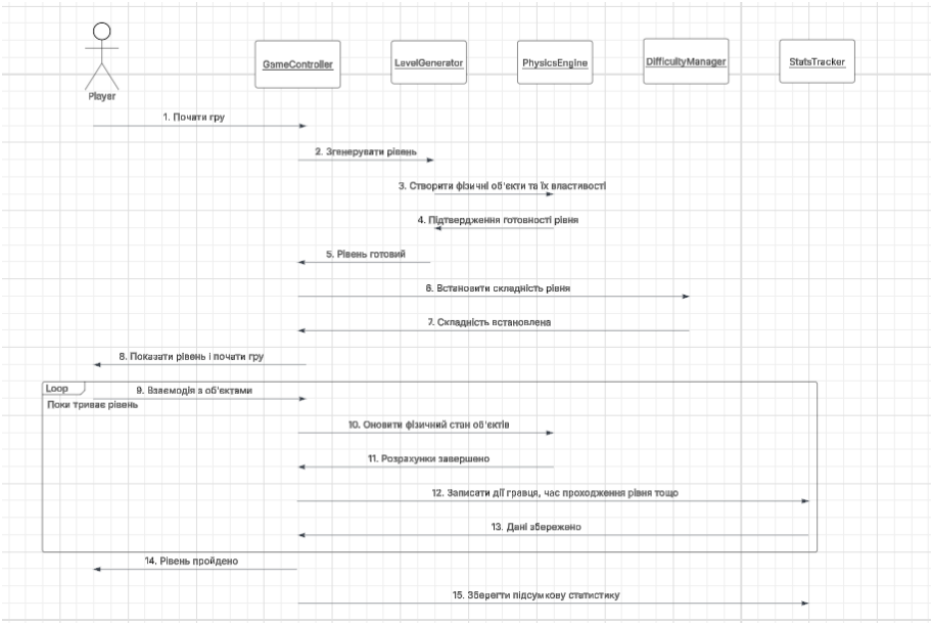


Рисунок 1.5 – UML-діаграма послідовності процесу гри

На діаграмі зображено сам процес гри. Користувач натискає «Почати гру», після чого генератор рівнів виконує поставлене йому завдання а контролер фізики створює об’єкти з фізичними властивостями. Також менеджер складностей встановлює рівень складності згенерованого рівня, після чого створений рівень відображається користувачу. Під час рівня можна взаємодіяти з об’єктами для досягнення цілі, тобто для виконання поставленого гравцеві завдання. При взаємодії з об’єктами оновлюється їх фізичний стан, який відображається в грі, а також дії гравця записуються для статистики та адаптації складності наступних рівнів. Після виконання гравцем головного завдання рівня буде повідомлення про те, що рівень пройдено, а також збереження підсумкової статистики такої, як час проходження рівня, кроки(дії гравця та їх кількість) тощо. Після цього все це повторюється, якщо гравець вирішує грати ще один рівень.

Ці діаграми дозволяють чітко побачити логіку, яку повинен реалізувати застосунок, та взаємозв’язки між елементами, що забезпечують відповідність вимогам.

Окремо слід звернути увагу на вимоги до моніторингу та логування подій. Оскільки система адаптації залежить від даних про поведінку гравця,

потрібно забезпечити збір та збереження таких даних, як: час проходження рівня, кількість спроб, найуспішніші рішення, кількість невдалих дій. Ці дані дозволять побудувати профіль гравця для подальших коригувань.

З огляду на технологічну платформу вибір середовища розробки – Unity, накладає додаткові вимоги. Unity забезпечує інтеграцію з фізичними рушіями, підтримку скриптових мов (C#), інструменти для роботи з анімаціями та UI. Вимоги до фізичного рушія включають: обробку колізій у реальному часі, підтримку різних типів матеріалів (пружні, жорсткі, ковзкі), можливість накладення сил (гравітація, імпульси) – усе це має бути реалізовано без суттєвих втрат продуктивності.

Таким чином, аналіз вимог показує, що ігровий застосунок має бути не лише логічно коректним, але й адаптивним, масштабованим, ефективним і зручним у використанні. Вимоги, сформульовані вище, стають основою для подальшого проєктування архітектури, вибору технологічних рішень та оцінювання відповідності реалізації очікуваному функціоналу.

1.4 Висновки. Постановка задачі

1.4.1 Висновки

У результаті проведеного аналізу предметної області, існуючих підходів до розробки ігрових застосунків, а також дослідження вимог до сучасних інтерактивних систем, можна зробити висновок, що розробка адаптивної фізичної головоломки з процедурною генерацією рівнів є актуальним та практично значущим завданням у сфері ігрової індустрії.

Сучасні комп’ютерні ігри дедалі частіше виходять за межі класичних сценаріїв проходження рівнів, пропонуючи користувачам динамічний, непередбачуваний та персоналізований досвід. Особливо це стосується жанру головоломок, де важливу роль відіграє не лише правильність побудови ігрових

механік, але й здатність системи адаптуватися до рівня підготовки гравця. У зв'язку з цим поєднання фізичного моделювання, процедурної генерації контенту та адаптивної складності формує новий клас ігрових застосунків, які забезпечують значно вищу реіграбельність та залученість користувача.

Проведений аналіз показав, що існуючі рішення на ринку, хоча й можуть містити окремі елементи фізики або логічного геймплею, не забезпечують повноцінної інтеграції всіх необхідних компонентів. Зокрема, більшість популярних ігор використовують заздалегідь створені рівні, що обмежує варіативність проходження та знижує унікальність ігрового досвіду. Також у них відсутній або реалізований у спрощеній формі механізм адаптації складності, який враховує індивідуальні особливості гравця.

Окрему увагу слід приділити процедурній генерації рівнів, яка в сучасних умовах є одним із ключових напрямів розвитку ігрових технологій. Її використання дозволяє створювати нескінченну кількість варіацій ігрових сцен без необхідності ручного проєктування кожного рівня. Однак у випадку фізичних головоломок дана технологія потребує додаткових механізмів перевірки коректності, оскільки кожен рівень повинен бути не лише унікальним, але й розв'язуваним з урахуванням фізичних законів.

Як показав аналіз, важливим компонентом сучасних ігрових систем є також модуль збору та аналізу статистики користувача. Саме на основі таких даних система може визначати стиль гри користувача, його швидкість прийняття рішень, кількість помилок, середній час проходження рівня та інші параметри. Ці показники є основою для реалізації адаптивної складності, яка дозволяє динамічно змінювати параметри наступних рівнів.

У межах даної кваліфікаційної роботи передбачається розробка програмного застосунку, який об'єднує три ключові компоненти: фізичну модель взаємодії об'єктів, систему процедурної генерації рівнів та систему адаптації складності на основі аналізу ігрової статистики. Такий підхід

дозволяє створити гнучку архітектуру, яка забезпечує масштабованість і можливість подальшого розширення функціоналу.

Технологічною основою реалізації проєкту обрано ігровий рушій Unity, який є одним із найпоширеніших інструментів для створення як 2D, так і 3D ігор. Вибір даного середовища обумовлений його широкими можливостями у сфері фізичного моделювання, підтримкою скриптової мови C#, а також наявністю вбудованих інструментів для роботи з графікою, анімаціями, інтерфейсом користувача та системами частинок. Згідно з офіційною документацією Unity Technologies, рушій підтримує інтеграцію фізичних компонентів, що дозволяє реалізовувати складні взаємодії між об’єктами в режимі реального часу.

Додатково слід зазначити, що використання C# як основної мови програмування забезпечує високу продуктивність розробки та можливість реалізації складної логіки гри у структурованому вигляді. Завдяки об’єктно-орієнтованому підходу можна ефективно розділити систему на окремі модулі, такі як керування грою, генерація рівнів, фізичний рушій, адаптація складності та збір статистики.

У результаті проведеного аналізу було визначено, що основними недоліками існуючих рішень є:

– відсутність повноцінної процедурної генерації рівнів у фізичних головоломках;

– недостатній рівень адаптації складності під конкретного користувача;

– обмежена варіативність ігрового процесу через статичну структуру рівнів;

– недостатня інтеграція систем збору та аналізу ігрової статистики;

– слабка персоналізація ігрового досвіду.

У той же час визначено, що основними перевагами розроблюваного застосунку будуть:

– генерація унікальних рівнів для кожної нової ігрової сесії;

– адаптація складності відповідно до поведінки гравця;

– інтеграція фізичної симуляції як ключового елементу геймплею;

– можливість розширення системи новими механіками без зміни базової архітектури;

– накопичення статистичних даних для подальшого аналізу та оптимізації гри.

Таким чином, розроблювана система має на меті поєднати одразу декілька сучасних підходів до створення ігрових застосунків, що дозволяє отримати продукт нового покоління, орієнтований на адаптивність, реіграбельність та індивідуальний досвід користувача.

1.4.2 Постановка задачі

Метою кваліфікаційної роботи є розробка адаптивної фізичної головоломки з використанням процедурної генерації рівнів на основі ігрового рушія Unity та мови програмування C#.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз предметної області та існуючих аналогів ігрових застосунків;

– дослідити принципи роботи фізичних рушіїв у сучасних ігрових системах;

– проаналізувати методи процедурної генерації ігрового контенту;

– дослідити підходи до реалізації адаптивної складності в ігрових системах;

– сформувати вимоги до програмного забезпечення;

– розробити архітектуру ігрового застосунку з виділенням основних модулів;

– реалізувати систему генерації рівнів на основі заданих параметрів;

– реалізувати фізичну модель взаємодії ігрових об’єктів;

– розробити механізм збору та аналізу статистики гравця;

– реалізувати систему адаптації складності на основі отриманих даних;

– провести тестування розробленого програмного продукту;

– оцінити ефективність запропонованого підходу.

Об’єктом дослідження є процеси створення інтерактивних ігрових застосунків.

Предметом дослідження є методи реалізації адаптивної складності, процедурної генерації рівнів та фізичного моделювання в ігрових системах.

Результатом виконання кваліфікаційної роботи повинен стати програмний застосунок, що забезпечує динамічний ігровий процес із можливістю адаптації складності та автоматичної генерації рівнів, реалізований у середовищі Unity з використанням мови програмування C#.

2 ПРОЕКТУВАННЯ ПЗ

2.1 Обґрунтування вибору архітектури програмного забезпечення

Архітектура розробки програмного забезпечення – це розроблена структура додатка, яка включає визначення взаємодії компонентів інтерфейсу з внутрішніми процесами програми. Простіше кажучи, це своєрідний підхід, який визначає які функції за що відповідають та як вони взаємодіють між собою [12].

Розробка сучасних ігрових застосунків передбачає використання продуманої архітектури програмного забезпечення, яка забезпечує масштабованість, гнучкість та зручність подальшого супроводу системи. У контексті створення адаптивної фізичної головоломки з процедурною генерацією рівнів особливо важливим є правильний вибір архітектурного підходу, оскільки система повинна обробляти ігрову логіку, фізичні взаємодії, генерацію контенту та аналіз поведінки користувача в реальному часі.

У сучасній практиці розробки ігор на базі рушія Unity широко використовується компонентна архітектура. Компонентна архітектура є фундаментальною концепцією, яка формує ядро Unity розробки ігор. Він обертається навколо ідеї компонування складних ігрових об’єктів шляхом приєднання та комбінування багаторазових компонентів. Компоненти – це модульні функціональні елементи, які можна приєднати до GameObjects. Вони представляють поведінку, властивості або характеристики GameObject. Кожен компонент інкапсулює певну функціональність, таку як візуалізація, фізика, сценарії, аудіо або введення. Щоб розширити GameObject за допомогою певної функціональності, до нього можна приєднати компоненти. Unity надає широкий спектр вбудованих компонентів, які охоплюють різні аспекти розробки ігор. Наприклад, компонент Transform визначає положення, обертання та масштаб GameObject, тоді як компонент Rigidbody додає до об’єкта фізичну симуляцію. Unity також дозволяє створення спеціальних

компонентів за допомогою сценаріїв. Їх часто називають компонентами сценарію або компонентами MonoBehaviour [13]. Такий підхід дозволяє гнучко комбінувати функціональність, а також повторно використовувати вже реалізовані елементи.

У межах даного проєкту обрано компонентну архітектуру як основну, оскільки вона найбільш повно відповідає специфіці ігрових застосунків. Вона дозволяє розділити систему на логічні модулі, кожен з яких відповідає за окремий аспект роботи програми. Це, у свою чергу, спрощує процес розробки, тестування та подальшої модифікації системи.

Архітектура застосунку передбачає наявність центрального керуючого модуля – GameController, який відповідає за загальний життєвий цикл гри. Даний компонент координує роботу інших модулів, зокрема генератора рівнів, системи фізики, модуля адаптації складності та системи збору статистики. Такий підхід відповідає принципу централізованого керування, що дозволяє уникнути хаотичної взаємодії між компонентами.

Окрім базової архітектури, у процесі проєктування були використані відомі зразки (патерни) проєктування програмного забезпечення. Патерни проектування – це перевірені часом рішення, які допомагають розробникам створювати надійне, гнучке та масштабоване ПЗ. Вони являють собою шаблони, засновані на загальних принципах і архітектурних рішеннях, які можна використовувати в різних ситуаціях. Їхня роль полягає в тому, що вони допомагають нам проектувати програми більш організовано, роблять код більш перевикористовуваним і спрощують підтримку системи [14]. Застосування патернів дозволяє підвищити якість коду, зробити його більш зрозумілим та забезпечити можливість повторного використання рішень.

Одним із ключових патернів, що використовується у системі, є патерн Singleton. Він застосовується для реалізації таких компонентів, як GameController та DifficultyManager. Singleton – це шаблон розробки програмного забезпечення, який обмежує створення об’єкта класу окремим

екземпляром. Клас має лише один екземпляр, що забезпечує глобальну точку доступу до цього екземпляра [15].

Ще одним важливим патерном є Observer (спостерігач). Спостерігач — це поведінковий шаблон розробки програмного забезпечення, який дає змогу вказати механізм підписки для сповіщення кількох об’єктів про будь-які події, які впливають на об’єкт, за яким вони спостерігають. Шаблон програмування спостерігача встановлює зв’язок «один до багатьох» між об’єктом (суб’єктом) і залежними особами (спостерігачами). Піддослідний інформується щоразу, коли один із спостерігачів змінюється [15]. Наприклад, система збору статистики може підписуватись на події, що виникають під час гри (завершення рівня, дії гравця, зміна стану об’єктів). Таким чином, StatTracker отримує необхідні дані без прямого втручання в логіку інших модулів.

Для реалізації створення ігрових об’єктів використовується патерн Factory. Фабричний метод (Factory Method) – це патерн, який визначає інтерфейс до створення об’єктів деякого класу, але безпосереднє рішення у тому, об’єкт якого класу створювати відбувається у підкласах. Тобто патерн передбачає, що базовий клас делегує створення об’єктів класам-спадкоємцям [16]. Таким чином, генератор рівнів створює різні типи об’єктів (перешкоди, цілі, допоміжні елементи) на основі заданих параметрів. Використання фабричного методу дозволяє абстрагувати процес створення об’єктів і спростити додавання нових типів елементів у майбутньому.

Також у системі застосовується патерн State, який дозволяє описати різні стани гри (меню, активна гра, пауза, завершення рівня). Це дозволяє чітко розмежувати логіку поведінки системи залежно від поточного стану та уникнути складних умовних конструкцій у коді.

Окремо варто відзначити використання принципів SOLID, які є основою сучасного об’єктно-орієнтованого програмування. SOLID – це набір принципів об’єктно-орієнтованого програмування, які представив Роберт Мартін (дядько Боб) у 1995 році. Їхня ідея в тому, що треба уникати залежностей між

компонентами коду. Якщо є велика кількість залежностей, такий код важко підтримувати (спагеті-код). Кожен клас повинен виконувати лише один обов’язок. Це не означає, що в нього має бути тільки один метод. Це означає, що всі методи класу мають бути сфокусовані на виконання одного спільного завдання. Якщо є методи, які не відповідають меті існування класу, їх треба винести за його межі [17]. Наприклад, LevelGenerator відповідає виключно за генерацію рівнів, а PhysicsEngine — за фізичні розрахунки.

Принцип відкритості/закритості (Open/Closed Principle) можна сформулювати так: Сутність програми повинна бути відкрита для розширення, але закрита для зміни. Суть цього принципу полягає в тому, що система повинна бути побудована таким чином, що її подальші зміни повинні бути реалізовані за допомогою додавання нового коду, а не зміни вже існуючого [18]. Принцип відкритості/закритості (Open/Closed Principle) забезпечує можливість розширення функціональності системи без зміни існуючого коду. Це особливо важливо для ігрових застосунків, де можуть додаватися нові механіки, типи рівнів або поведінка об’єктів.

Вибір саме такої архітектури та набору патернів обумовлений необхідністю забезпечення гнучкості та масштабованості системи. Оскільки проєкт передбачає інтеграцію декількох складних підсистем (фізика, генерація, адаптація), важливо забезпечити їх незалежність та можливість окремого розвитку.

Таким чином, обрана архітектура поєднує в собі компонентний підхід, централізоване керування та використання перевірених патернів проєктування. Це дозволяє створити структуровану, розширювану та ефективну програмну систему, що відповідає сучасним вимогам до розробки ігрових застосунків.

2.2 Опис декомпозиції

Декомпозиція – це потужний інструмент, який допомагає спростити складну задачу і зробити її більш зрозумілою для розробників. Це призводить до поліпшення якості коду та зниження витрат на розробку й тестування [19]. Такий підхід дозволяє спростити розробку, підвищити зрозумілість структури програмного забезпечення та забезпечити можливість незалежної розробки й тестування окремих модулів. У випадку розробки адаптивної фізичної головоломки декомпозиція є особливо важливою, оскільки система поєднує кілька складних підсистем: ігрову логіку, фізичне моделювання, генерацію рівнів, адаптацію складності та збір статистики.

Відповідно до обраної компонентної архітектури, програмна система поділяється на окремі функціональні модулі, кожен із яких виконує чітко визначену роль. Такий підхід відповідає сучасним принципам розробки програмного забезпечення, зокрема принципу єдиної відповідальності, що передбачає мінімізацію функціонального навантаження кожного окремого компонента.

У межах даного проєкту було виділено такі основні компоненти системи: GameController, LevelGenerator, PhysicsEngine, DifficultyManager, StatTracker та модуль користувацького інтерфейсу. Кожен із цих компонентів виконує окрему функцію, але разом вони утворюють єдину узгоджену систему.

Центральним елементом системи є GameController. Даний компонент відповідає за загальне керування ігровим процесом та координує взаємодію між усіма іншими модулями. Він забезпечує ініціалізацію гри, запуск нових рівнів, обробку завершення ігрової сесії, а також керування переходами між різними станами системи. Фактично, GameController виконує роль керівника, який визначає послідовність виконання основних процесів. Такий підхід дозволяє централізувати управління логікою гри та уникнути дублювання функціоналу в інших модулях.

Наступним важливим компонентом є LevelGenerator, який відповідає за процедурну генерацію рівнів. Його основною функцією є створення конфігурації ігрового простору на основі заданих параметрів складності. Генерація включає розміщення об’єктів, визначення їх властивостей та формування структури рівня. У процесі роботи LevelGenerator використовує алгоритми випадкової генерації, а також враховує обмеження, пов’язані з фізичною коректністю та розв’язуваністю рівня. Таким чином, цей модуль забезпечує унікальність кожної ігрової сесії та підвищує реіграбельність ігрового застосунку.

Компонент PhysicsEngine відповідає за моделювання фізичних процесів у грі. У середовищі Unity фізичний рушій реалізується за допомогою вбудованих механізмів, таких як Rigidbody та Collider. Rigidbody дозволяє ігровим об’єктам взаємодіяти з фізикою. Для реалістичного руху об’єктів на них впливають сила обертання та інші сили. Будь-який ігровий об’єкт повинен містити Rigidbody, щоб бути сприйнятливим до гравітації, діяти відповідно до призначеного шляху написання скриптів сил або взаємодіяти з іншими об’єктами через фізичний рушій NVIDIA PhysX [20]. Компоненти колайдера визначають форму ігрового об’єкта (GameObject) для цілей фізичних зіткнень [21]. Даний модуль забезпечує обробку зіткнень, розрахунок траєкторій руху, вплив гравітації та інших сил. Важливою особливістю є те, що PhysicsEngine працює в режимі реального часу, що накладає додаткові вимоги до продуктивності та стабільності системи.

Модуль DifficultyManager реалізує механізм адаптації складності гри. Його основною задачею є аналіз поведінки користувача та коригування параметрів наступних рівнів. Для цього використовуються дані, отримані від StatTracker, такі як час проходження рівня, кількість спроб, точність дій та інші показники. На основі цих даних DifficultyManager визначає, чи необхідно підвищити або знизити складність. Такий підхід відповідає концепції

адаптивного геймплею, яка спрямована на підтримання оптимального рівня виклику для гравця.

StatTracker є модулем, що відповідає за збір та аналіз ігрової статистики. Він фіксує всі ключові події, що відбуваються під час гри, включаючи дії користувача, результати проходження рівнів, кількість помилок та інші параметри. Зібрані дані використовуються не лише для адаптації складності, але й можуть бути застосовані для подальшого аналізу ефективності ігрових механік. Важливою характеристикою цього модуля є його незалежність від інших компонентів, що дозволяє змінювати або розширювати функціональність збору даних без впливу на основну логіку гри.

Окремим компонентом системи є модуль користувацького інтерфейсу (UI). Він забезпечує взаємодію гравця із системою, відображає інформацію про стан гри, результати, підказки та інші елементи. UI включає головне меню, екран гри, екран результатів та інші інтерфейсні компоненти. Важливою вимогою до цього модуля є забезпечення інтуїтивної зрозумілості та зручності використання.

Декомпозиція системи також передбачає розділення функціональності всередині кожного модуля. Наприклад, у межах LevelGenerator можуть бути виділені підкомпоненти, відповідальні за генерацію структури рівня, розміщення об’єктів та перевірку їх коректності. Аналогічно, у модулі StatTracker можна виділити підсистеми збору даних, їх збереження та аналізу.

Важливою характеристикою запропонованої декомпозиції є слабка зв’язаність між компонентами. Це означає, що модулі взаємодіють між собою через чітко визначені інтерфейси, не залежачи від внутрішньої реалізації один одного. Такий підхід дозволяє змінювати або замінювати окремі компоненти без необхідності модифікації всієї системи. Наприклад, алгоритм генерації рівнів може бути змінений без впливу на роботу фізичного рушія або системи адаптації складності.

Ще однією важливою перевагою декомпозиції є можливість паралельної розробки. Оскільки кожен модуль має чітко визначену відповідальність, різні частини системи можуть розроблятися незалежно. Це дозволяє скоротити час розробки та підвищити ефективність роботи.

Скрипт взаємодіє з внутрішніми механізмами Unity, створюючи клас, успадкований від вбудованого класу під назвою MonoBehaviour [22]. Це дозволяє інтегрувати модулі безпосередньо у структуру сцени та використовувати стандартні механізми рушія для їх взаємодії.

Для наочності структури системи доцільно представити її у вигляді схеми (рисунок 2.1), на якій відображено основні компоненти та зв’язки між ними.

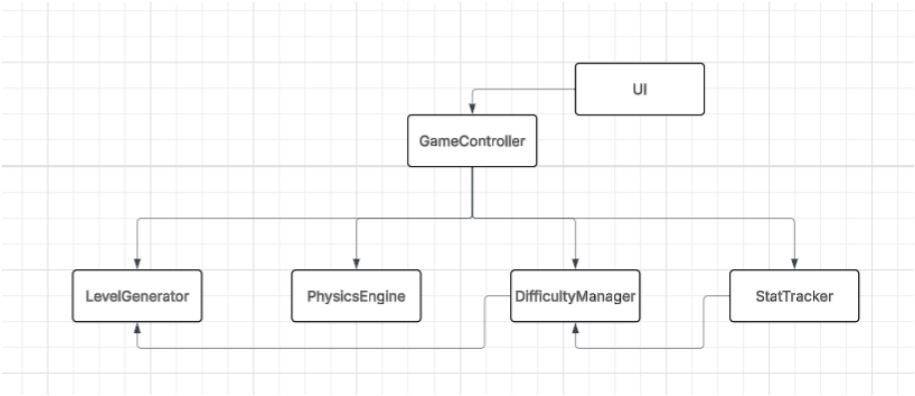


Рисунок 2.1 – Структурна схема декомпозиції програмної системи

Таким чином, проведена декомпозиція дозволяє представити програмну систему як сукупність взаємопов’язаних, але незалежних компонентів, кожен із яких виконує окрему функцію. Такий підхід забезпечує гнучкість, масштабованість та зручність супроводу програмного забезпечення. Крім того, він створює основу для подальшого опису залежностей між модулями.

2.3 Опис залежностей

У процесі проєктування програмного забезпечення одним із ключових аспектів є визначення залежностей між компонентами системи. Залежності описують, яким чином окремі модулі взаємодіють між собою, які дані передаються між ними та як зміни в одному компоненті можуть впливати на інші. Грамотно спроєктовані залежності дозволяють забезпечити гнучкість, масштабованість та зручність супроводу програмного продукту.

У контексті розробки адаптивної фізичної головоломки залежності між компонентами відіграють особливо важливу роль, оскільки система включає декілька взаємопов’язаних підсистем: ігрову логіку, фізичне моделювання, генерацію рівнів, адаптацію складності та збір статистики. Неправильна організація залежностей може призвести до високої зв’язаності (coupling), що ускладнить модифікацію системи та її подальший розвиток.

Основною метою при проєктуванні залежностей є досягнення слабкої зв’язаності між компонентами та високої згуртованості (cohesion) всередині кожного модуля. Слабка зв’язаність означає, що зміни в одному компоненті не потребують змін в інших, тоді як висока згуртованість означає, що всі елементи модуля виконують логічно пов’язані функції.

У розробленій системі центральним компонентом, який має найбільшу кількість залежностей, є GameController. Саме він координує роботу інших модулів і виступає основною точкою взаємодії між ними. GameController має залежності від таких компонентів, як LevelGenerator, PhysicsEngine, DifficultyManager та StatTracker. Водночас інші модулі не мають прямих залежностей один від одного, що дозволяє зменшити загальну зв’язаність системи.

Залежність між GameController та LevelGenerator полягає у тому, що контролер ініціює процес створення нового рівня. GameController передає до генератора параметри складності, які визначають структуру майбутнього рівня.

У відповідь LevelGenerator повертає сформований набір об’єктів та їх властивостей. Важливо, що GameController не залежить від конкретної реалізації генератора, а взаємодіє з ним через абстрактний інтерфейс. Це дозволяє змінювати алгоритм генерації без впливу на інші частини системи.

LevelGenerator, у свою чергу, має залежність від PhysicsEngine, але вона є непрямою. Генератор рівнів повинен враховувати фізичні властивості об’єктів (масу, розміри, типи колайдерів), щоб створений рівень був коректним і розв’язуваним. Однак сам генератор не виконує фізичні розрахунки, а лише формує дані, які передаються до фізичного рушія. Таким чином, залежність між цими компонентами є логічною, але не жорсткою.

PhysicsEngine є відносно незалежним компонентом, який отримує дані про об’єкти та виконує їх фізичну симуляцію. Його основною залежністю є ігрова сцена, у межах якої відбувається обробка фізичних процесів. Він не залежить від логіки генерації або адаптації складності, що відповідає принципу розділення відповідальностей. Водночас інші модулі можуть отримувати результати роботи фізичного рушія, наприклад інформацію про зіткнення або зміну стану об’єктів.

Важливу роль у системі відіграє модуль DifficultyManager, який відповідає за адаптацію складності. Його основною залежністю є StatTracker, оскільки саме цей модуль надає дані про поведінку гравця. DifficultyManager аналізує отриману статистику та формує параметри складності, які передаються до LevelGenerator. Таким чином, між цими трьома компонентами формується логічний ланцюг залежностей: StatTracker → DifficultyManager → LevelGenerator.

StatTracker, у свою чергу, реалізує залежність від подій, що відбуваються у системі. Він не викликає інші модулі напряму, а підписується на події, які генеруються GameController та PhysicsEngine. Такий підхід реалізується за допомогою патерна Observer і дозволяє уникнути жорстких залежностей.

Наприклад, при завершенні рівня GameController генерує подію, яку перехоплює StatTracker і зберігає відповідні дані.

Залежності між компонентами також можна класифікувати за типами. У даній системі використовуються такі основні типи залежностей:

– функціональні залежності, які визначають порядок виклику модулів;

– інформаційні залежності, що описують передачу даних між компонентами;

– часові залежності, які визначають послідовність виконання операцій;

– подієві залежності, які базуються на механізмі обробки подій.

Функціональні залежності проявляються у взаємодії GameController з іншими модулями. Наприклад, перед початком гри необхідно згенерувати рівень, після чого активувати фізичний рушій. Це означає, що LevelGenerator має бути викликаний раніше за PhysicsEngine.

Інформаційні залежності виникають при передачі даних між компонентами. Наприклад, DifficultyManager передає параметри складності до LevelGenerator, а StatTracker передає статистичні дані до модуля адаптації. Важливо, що передача даних здійснюється через чітко визначені інтерфейси, що дозволяє уникнути прямого доступу до внутрішніх структур модулів.

Часові залежності визначають порядок виконання процесів у системі. Наприклад, аналіз статистики має відбуватися після завершення рівня, а не під час його генерації. Неправильне визначення часових залежностей може призвести до некоректної роботи системи.

Подієві залежності реалізуються через механізм підписки на події. Вони дозволяють компонентам реагувати на зміни стану системи без прямого виклику один одного. Наприклад, при зміні стану гри (перехід у паузу або завершення рівня) відповідні модулі отримують повідомлення і виконують необхідні дії.

Важливим аспектом є управління залежностями за допомогою принципу інверсії залежностей (Dependency Inversion Principle). Принцип інверсії

залежностей (DIP) – це один з основоположних принципів SOLID, який стверджує, що класи та модулі не повинні залежати від конкретних реалізацій інших класів. Залежності повинні бути абстраговані за допомогою інтерфейсів.

Переваги DIP:

– Зниження зв’язків: Зменшується зв’язок між класами, роблячи код більш модульним.

– Підвищення гнучкості: Легше змінювати та розширювати код, не ламаючи інші частини.

– Збільшення тестованості: Легше тестувати код, який не залежить від конкретних реалізацій.

– Підвищення надійності: Знижується ризик помилок, пов’язаних з залежностями [23].

У даній системі це реалізується через використання інтерфейсів для взаємодії між компонентами.

Наприклад, GameController не залежить від конкретної реалізації LevelGenerator, а працює через інтерфейс генерації рівнів. Це дозволяє замінити алгоритм генерації без зміни логіки контролера. Аналогічно, DifficultyManager взаємодіє зі StatTracker через абстрактний інтерфейс отримання статистики.

Ще одним важливим підходом є використання ін’єкції залежностей (Dependency Injection). Dependecy Injection – це проєктувальний патерн, за яким об’єкт або функція отримує інші об’єкти або функції, від яких вони залежить. DI має на меті розділення процесів створення та використання об’єктів іншими класами, що призводить до певної децентралізації системи [24]. Це зменшує зв’язаність та полегшує тестування системи. У середовищі Unity цей підхід може реалізовуватися через серіалізовані поля або спеціалізовані фреймворки.

Окрім цього, у системі враховано можливість розширення залежностей. Наприклад, у майбутньому можна додати нові модулі, такі як система збереження прогресу або мережевий компонент. Завдяки слабкій зв’язаності їх інтеграція не потребуватиме значних змін у вже існуючій структурі.

Для наочності залежності між компонентами можуть бути представлені у вигляді діаграми залежностей, на якій відображено основні зв’язки між модулями (рисунок 2.2).

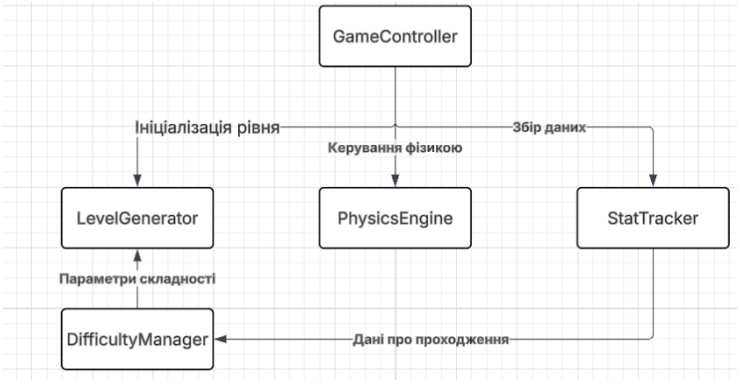


Рисунок 2.2 – Схема залежностей між компонентами системи

На схемі видно, що GameController є центральним вузлом, який взаємодіє з іншими компонентами. LevelGenerator отримує параметри від DifficultyManager, який, у свою чергу, використовує дані StatTracker. PhysicsEngine функціонує незалежно, але взаємодіє з GameController у процесі виконання гри.

Таким чином, запропонована структура залежностей забезпечує баланс між централізованим керуванням та незалежністю компонентів. Вона дозволяє зменшити зв’язаність системи, підвищити її гнучкість та спростити процес модифікації. Використання сучасних принципів проєктування, таких як інверсія залежностей, подієва взаємодія та слабка зв’язаність, забезпечує відповідність системи сучасним вимогам до розробки програмного забезпечення.

У результаті можна зробити висновок, що правильно організовані залежності є критично важливими для забезпечення ефективності, масштабованості та надійності ігрового застосунку. Саме вони формують основу для стабільної роботи системи та її подальшого розвитку.

2.4 Опис інтерфейсів

Інтерфейси програмної системи є критично важливим елементом проєктування, оскільки саме вони визначають способи взаємодії між компонентами системи та забезпечують зв’язок між внутрішньою логікою застосунку і користувачем. У сучасних ігрових застосунках інтерфейси виконують не лише роль механізму обміну даними, але й виступають одним із ключових факторів, що впливають на ефективність роботи системи, її масштабованість та зручність використання.

У контексті розробки адаптивної фізичної головоломки з процедурною генерацією рівнів інтерфейси мають забезпечувати узгоджену взаємодію між такими основними компонентами системи, як GameController, LevelGenerator, PhysicsEngine, DifficultyManager, StatTracker, а також модулем користувацького інтерфейсу. Кожен із зазначених компонентів виконує окрему функціональну роль, тому правильна організація інтерфейсів дозволяє досягти слабкої зв’язаності між ними, що є однією з основних вимог сучасної програмної інженерії.

Загалом інтерфейси системи можна умовно поділити на внутрішні (міжмодульні) та зовнішні (користувацькі). Внутрішні інтерфейси забезпечують взаємодію між компонентами системи, тоді як зовнішні відповідають за взаємодію користувача з програмним продуктом.

Внутрішні інтерфейси реалізуються у вигляді набору методів, подій та контрактів, які визначають спосіб взаємодії між модулями. Важливою вимогою до таких інтерфейсів є їх незалежність від конкретної реалізації компонентів, що дозволяє змінювати внутрішню логіку модулів без впливу на інші частини системи.

Центральним елементом взаємодії виступає GameController, який реалізує інтерфейс керування життєвим циклом гри. Через цей інтерфейс здійснюється ініціалізація ігрової сесії, запуск нового рівня, обробка завершення гри та

перехід між станами. GameController взаємодіє з іншими модулями через чітко визначені інтерфейси, що дозволяє уникнути прямої залежності між компонентами.

Інтерфейс взаємодії між GameController та LevelGenerator передбачає передачу параметрів генерації, зокрема рівня складності, типу рівня та обмежень щодо структури сцени. LevelGenerator, у свою чергу, повертає опис згенерованого рівня у вигляді структури даних, що містить інформацію про об’єкти, їх координати, фізичні властивості та взаємозв’язки. Такий підхід дозволяє відокремити логіку генерації від логіки виконання гри.

Особливістю інтерфейсу LevelGenerator є необхідність забезпечення детермінованості або контрольованої випадковості генерації. Це означає, що при однакових вхідних параметрах система може відтворити однаковий рівень, що є важливим для тестування та налагодження.

Інтерфейс взаємодії з PhysicsEngine забезпечує передачу команд для ініціалізації фізичних об’єктів, запуску симуляції та отримання результатів обробки фізичних процесів. PhysicsEngine працює на основі рушія Unity і використовує компоненти Rigidbody та Collider для моделювання фізичних явищ. Важливою вимогою до цього інтерфейсу є забезпечення високої продуктивності та мінімальних затримок, оскільки фізична симуляція виконується у режимі реального часу.

Крім того, інтерфейс PhysicsEngine повинен підтримувати обробку подій зіткнень, які можуть використовуватися іншими компонентами системи. Наприклад, StatTracker може фіксувати кількість зіткнень або силу ударів для подальшого аналізу поведінки гравця.

Інтерфейс взаємодії між GameController та DifficultyManager забезпечує передачу даних про результати ігрової сесії та отримання оновлених параметрів складності. DifficultyManager аналізує статистичні дані, такі як час проходження рівня, кількість спроб, точність дій та ефективність рішень. На основі цих даних формується рішення щодо зміни складності наступного рівня.

Важливою особливістю цього інтерфейсу є підтримка адаптивності в реальному часі. Це означає, що складність може змінюватися не лише між рівнями, але й під час їх проходження. Для цього інтерфейс повинен забезпечувати можливість динамічного оновлення параметрів гри.

Інтерфейс StatTracker базується на подієвій моделі взаємодії. Він підписується на події, що генеруються іншими компонентами системи, та фіксує їх для подальшого аналізу. Такий підхід дозволяє уникнути прямого виклику методів StatTracker з інших модулів, що забезпечує слабку зв’язаність системи.

StatTracker реалізує інтерфейси збору, збереження та аналізу даних. Він може взаємодіяти з локальним сховищем або базою даних для збереження інформації про ігрові сесії. Інтерфейс цього модуля повинен забезпечувати швидкий доступ до статистичних даних та можливість їх обробки у режимі реального часу.

Окрему увагу слід приділити інтерфейсу користувача, який є основним засобом взаємодії гравця із системою. Користувацький інтерфейс повинен бути інтуїтивно зрозумілим, зручним у використанні та відповідати сучасним вимогам до дизайну інтерактивних систем.

Основними складовими користувацького інтерфейсу є головне меню, екран гри, екран результатів та інтерфейс статистики. Головне меню забезпечує доступ до основних функцій застосунку, таких як початок гри, налаштування та перегляд статистики. Воно повинно бути простим і зрозумілим, щоб користувач міг швидко орієнтуватися у функціоналі системи.

Екран гри є центральним елементом інтерфейсу, на якому відбувається основна взаємодія користувача з системою. Він включає область відображення ігрового рівня, елементи керування та інформаційні панелі. Інтерфейс повинен забезпечувати чітке відображення об’єктів та їх стану, а також швидку реакцію на дії користувача.

Взаємодія користувача з ігровими об’єктами реалізується через механізми введення, такі як миша або сенсорний екран. Інтерфейс повинен забезпечувати точне визначення дій користувача та їх коректну передачу до системи. Важливою вимогою є мінімізація затримок між дією користувача та реакцією системи.

Екран результатів відображає інформацію про завершення рівня, включаючи час проходження, кількість спроб та оцінку ефективності. Він також надає можливість переходу до наступного рівня або повторного проходження поточного.

Інтерфейс статистики дозволяє користувачу переглядати накопичені дані про ігровий процес. Він може включати графіки, таблиці та інші візуальні елементи, що відображають прогрес гравця. Такий інтерфейс підвищує мотивацію користувача та сприяє більш глибокому залученню в ігровий процес.

Реалізація користувацького інтерфейсу здійснюється з використанням інструментів Unity UI, що дозволяє створювати адаптивні інтерфейси з урахуванням різних роздільних здатностей екрану. Використання компонентів Canvas, Button, Text та інших забезпечує гнучкість та зручність розробки.

Важливою характеристикою інтерфейсу є його адаптивність. Інтерфейс повинен коректно відображатися на різних пристроях та підтримувати різні формати екрану. Для цього використовуються механізми масштабування та адаптивного розміщення елементів.

Ще одним важливим аспектом є забезпечення зворотного зв’язку. Кожна дія користувача повинна супроводжуватися відповідною реакцією системи, що може бути реалізовано через візуальні або звукові ефекти. Це дозволяє підвищити рівень залученості користувача та зробити ігровий процес більш привабливим.

Інтерфейси також відіграють важливу роль у забезпеченні тестованості системи. Чітко визначені інтерфейси дозволяють проводити модульне та

інтеграційне тестування, що є необхідним для забезпечення якості програмного продукту.

Крім того, інтерфейси повинні бути задокументовані. Документація включає опис методів, параметрів та сценаріїв використання. Це спрощує процес розробки та дозволяє уникнути помилок під час інтеграції компонентів.

У контексті подальшого розвитку системи інтерфейси забезпечують можливість її розширення. Завдяки чітко визначеним контрактам можна додавати нові модулі або змінювати існуючі без значного впливу на інші компоненти.

Таким чином, інтерфейси програмної системи є основою її функціонування, забезпечуючи взаємодію між компонентами та користувачем. Їх правильне проєктування дозволяє створити гнучку та ефективну систему, що відповідає сучасним вимогам до розробки ігрових застосунків.

2.5 Висновки

У другому розділі кваліфікаційної роботи було виконано проєктування архітектури, структури та взаємодії компонентів ігрового застосунку, що реалізує адаптивну фізичну головоломку з процедурною генерацією рівнів.

На етапі обґрунтування архітектури було визначено доцільність використання компонентного підходу, який є базовим для рушія Unity. Така архітектура забезпечує модульність системи, спрощує розширення функціональності та дозволяє ефективно розділити відповідальність між окремими частинами програми. Додатково було обґрунтовано використання класичних патернів проєктування (Singleton, Observer, Factory, State), що підвищує структурованість коду та забезпечує його повторне використання.

У підрозділі декомпозиції було визначено основні модулі системи: GameController, LevelGenerator, PhysicsEngine, DifficultyManager, StatTracker та

модуль користувацького інтерфейсу. Кожен із них виконує чітко визначену функцію, що дозволило розділити складну систему на логічно незалежні частини. Це забезпечує можливість паралельної розробки, спрощує тестування та підвищує загальну підтримуваність програмного продукту.

Під час опису залежностей між компонентами було встановлено, що система побудована за принципом слабкої зв’язаності. Центральним елементом виступає GameController, який координує взаємодію між модулями. Взаємодія між компонентами реалізована через інтерфейси та подієву модель, що дозволяє мінімізувати прямі залежності та забезпечує гнучкість архітектури. Особливу увагу приділено принципам SOLID, інверсії залежностей та подієвій взаємодії, що сприяє масштабованості системи.

У підрозділі інтерфейсів було визначено внутрішні та зовнішні інтерфейси системи. Внутрішні інтерфейси забезпечують обмін даними між модулями, тоді як зовнішні відповідають за взаємодію користувача з ігровим застосунком. Описані інтерфейси дозволяють забезпечити чіткий розподіл відповідальностей, підтримку адаптивності складності, а також коректну взаємодію між системою генерації рівнів, фізичним рушієм та підсистемою збору статистики.

Таким чином, у розділі 2 було сформовано повну архітектурну модель ігрового застосунку, визначено його основні модулі, встановлено взаємозв’язки між ними та описано інтерфейси взаємодії. Отримані результати створюють основу для подальшої реалізації програмного продукту та забезпечують його структурну цілісність, масштабованість і зручність супроводу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Програмна реалізація модулів

Розроблення програмного забезпечення адаптивної фізичної головоломки передбачає реалізацію низки функціональних модулів, кожен із яких відповідає за окрему частину ігрового процесу. Відповідно до обраної архітектури, система побудована за модульним принципом, що забезпечує її гнучкість, масштабованість та зручність супроводу.

Основними модулями програмної системи є GameController, LevelGenerator, PhysicsEngine, DifficultyManager та StatTracker. Кожен із цих компонентів реалізований у вигляді окремого класу мовою програмування C# з використанням можливостей рушія Unity.

Центральним компонентом системи є GameController, який відповідає за керування життєвим циклом гри. Даний модуль ініціалізує ігрову сесію, запускає генерацію рівнів, координує роботу інших компонентів та обробляє завершення гри. GameController реалізований як Singleton, що забезпечує єдину точку доступу до нього з інших частин системи.

Приклад реалізації базової структури GameController наведено нижче:

```
public class GameController : MonoBehaviour
{
    public static GameController Instance;

    private void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    public void StartGame()
    {
```

```
        int difficulty =
DifficultyManager.Instance.GetDifficulty();
        LevelGenerator.Instance.GenerateLevel(difficulty);
    }

    public void EndLevel(bool success)
    {
        StatTracker.Instance.SaveResult(success);
        DifficultyManager.Instance.UpdateDifficulty();
    }
}
```

У наведеному фрагменті показано основні методи керування грою: запуск рівня та його завершення. GameController координує взаємодію між генератором рівнів, системою адаптації складності та модулем збору статистики.

Наступним важливим компонентом є LevelGenerator, який відповідає за процедурну генерацію рівнів. Даний модуль створює ігрову сцену на основі параметрів складності, що передаються від DifficultyManager. Генерація включає розміщення об’єктів, визначення їх властивостей та формування логіки рівня.

Приклад реалізації методу генерації рівня:

```
public class LevelGenerator : MonoBehaviour
{
    public static LevelGenerator Instance;

    public GameObject blockPrefab;

    public void GenerateLevel(int difficulty)
    {
        int objectCount = difficulty * 5;

        for (int i = 0; i < objectCount; i++)
        {
            Vector3 position = new Vector3(
                Random.Range(-5f, 5f),
                Random.Range(1f, 5f),
                0
            );

            Instantiate(blockPrefab, position,
Quaternion.identity);
```

```
        }
    }
}
```

У цьому прикладі використовується генерація об’єктів на основі випадкових координат. Кількість об’єктів залежить від рівня складності, що дозволяє забезпечити варіативність ігрового процесу.

Модуль PhysicsEngine у середовищі Unity реалізується за допомогою вбудованих компонентів фізики. Зокрема, для забезпечення взаємодії об’єктів використовуються компоненти Rigidbody та Collider. Програмна реалізація цього модуля полягає у налаштуванні фізичних властивостей об’єктів та обробці подій зіткнень.

Приклад обробки зіткнень:

```
public class PhysicsHandler : MonoBehaviour
{
    private void OnCollisionEnter(Collision collision)
    {
        Debug.Log(«Collision with: « + collision.gameObject.name);
        StatTracker.Instance.RegisterCollision();
    }
}
```

Цей фрагмент демонструє, як система реагує на фізичні події та передає інформацію до модуля збору статистики.

Модуль DifficultyManager відповідає за адаптацію складності гри. Він аналізує результати попередніх рівнів та визначає параметри для наступних. Реалізація базується на простих евристичних правилах, що враховують успішність гравця.

Приклад реалізації:

```
public class DifficultyManager : MonoBehaviour
{
    public static DifficultyManager Instance;
```

```
    private int difficulty = 1;

    public int GetDifficulty()
    {
        return difficulty;
    }

    public void UpdateDifficulty()
    {
        float successRate = StatTracker.Instance.GetSuccessRate();

        if (successRate > 0.8f)
            difficulty++;
        else if (successRate < 0.5f && difficulty > 1)
            difficulty--;
    }
}
```

У цьому прикладі складність змінюється залежно від успішності гравця, що забезпечує адаптивність системи.

Модуль StatTracker реалізує збір та аналіз статистичних даних. Він фіксує результати проходження рівнів, кількість дій та інші параметри, які використовуються для адаптації складності.

Приклад реалізації:

```
public class StatTracker : MonoBehaviour
{
    public static StatTracker Instance;

    private int totalAttempts = 0;
    private int successfulAttempts = 0;

    public void SaveResult(bool success)
```

```
    {
        totalAttempts++;
        if (success)
            successfulAttempts++;
    }

    public float GetSuccessRate()
    {
        if (totalAttempts == 0) return 0;
        return (float)successfulAttempts / totalAttempts;
    }

    public void RegisterCollision()
    {
        // логування події
    }
}
```

Цей модуль забезпечує накопичення даних, необхідних для аналізу поведінки гравця.

Важливою складовою реалізації є також інтеграція модулів між собою. Взаємодія здійснюється через виклики методів та обмін даними, що дозволяє забезпечити узгоджену роботу всієї системи. Використання патернів проєктування, таких як Singleton та Observer, дозволяє спростити взаємодію та зменшити залежності між компонентами.

Окрему увагу приділено реалізації користувацького інтерфейсу. Для цього використовується система Unity UI, яка дозволяє створювати інтерактивні елементи керування. Кнопки, текстові поля та інші елементи взаємодіють із GameController через події.

Приклад обробки натискання кнопки:

```
public void OnStartButtonClicked()
```

```
{
    GameController.Instance.StartGame();
}
```

Таким чином, користувач ініціює запуск гри через інтерфейс, а система обробляє цю дію відповідно до логіки застосунку.

У процесі реалізації особлива увага приділялася забезпеченню читабельності коду, його структурованості та відповідності принципам об’єктно-орієнтованого програмування. Кожен клас виконує чітко визначену функцію, що відповідає принципу єдиної відповідальності.

Таким чином, програмна реалізація модулів забезпечує повноцінне функціонування ігрового застосунку, включаючи генерацію рівнів, фізичне моделювання, адаптацію складності та збір статистики. Реалізована система є гнучкою, масштабованою та придатною для подальшого розвитку.

3.2 Розроблення бази даних

3.2.1 Загальний опис бази даних

База даних – це структурована колекція даних, яка зберігається в електронному форматі та доступна для операцій обробки, пошуку та аналізу. По своїй суті, база даних є цифровим сховищем, що містить інформацію про різні об’єкти, які можуть бути пов’язані між собою [25].

У процесі розроблення ігрового застосунку важливим етапом є проєктування та реалізація бази даних, яка забезпечує збереження інформації про користувача, результати проходження рівнів, статистичні показники та параметри адаптації складності. Незважаючи на те, що гра не потребує складної корпоративної СКБД, наявність структурованого сховища даних є необхідною умовою для реалізації механізмів аналізу поведінки гравця та адаптації ігрового процесу.

У межах даного проєкту база даних використовується для:

– збереження інформації про гравців;

– фіксації результатів проходження рівнів;

– накопичення статистики для системи адаптації складності;

– забезпечення можливості аналізу ігрових сесій.

Також було розглянуто різні типи баз даних. Реляційні бази даних використовуються з 1970-х років і зберігають дані у структурованих таблицях із рядками та стовпцями. Назва «реляційні» походить від способу взаємозв’язку таблиць на основі зв’язків між даними.

Система керування реляційними базами даних (РСКБД) – це програмне забезпечення, яке дозволяє створювати, оновлювати та керувати реляційними базами даних. Стандартною мовою для запитів, зміни й керування даними є SQL (Structured Query Language).

Ключові особливості

– Відповідність ACID: Гарантує надійність через атомарність, узгодженість, ізоляцію й довговічність.

– Підтримка структурованих даних: Найкраще підходить для структурованих даних із попередньо визначеними зв’язками.

– Висока надійність: Забезпечує узгодженість та цілісність, що робить її ідеальною для критичних застосунків.

Реляційні бази даних не завжди є найкращим вибором для великих обсягів неструктурованих або напівструктурованих даних [26].

Бази даних «ключ-значення» – найпростіший вид нереляційних БД. Такі бази працюють на основі моделі даних «ключ-значення». Кожен запис у базі містить ключ-ідентифікатор запису та значення, пов’язане з ключем. Основна ідея цієї бази – дані не мають залежностей і не належать до певних структур, як у випадку з реляційними БД. Тому їх легко масштабувати за потреби.

Доступ до даних відбувається за унікальним ключем, а це забезпечує швидкий пошук, що корисно для великої кількості даних. Тут можна зберігати інформацію різного формату, як, наприклад, JSON- чи XML-файли, рядки, числа і текст в одній БД [27].

Документоорієнтовані БД – У цих БД дані зберігаються в структурованих форматах — XML, JSON, BSON. При цьому зберігається адресний доступ до даних за ключем. Вміст документа може мати різний набір властивостей.

Ці бази даних добре підходять для швидкої розробки систем та сервісів, що працюють з по-різному структурованими даними. Вони легко масштабуються та змінюють структуру за потреби [28].

З огляду на особливості застосунку було обрано реляційну модель бази даних, оскільки вона забезпечує чітку структуру даних, цілісність та зручність обробки запитів. Як СКБД використовується SQLite, яка добре інтегрується з Unity та не потребує окремого серверного середовища.

Структура бази даних

У результаті аналізу вимог було визначено основні сутності предметної області:

– гравець;

– ігрова сесія;

– рівень;

– статистика проходження.

На основі цих сутностей сформовано структуру бази даних, яка складається з таких таблиць:

– Players – зберігає інформацію про користувачів;

– GameSessions – зберігає дані про ігрові сесії;

– Levels – містить інформацію про згенеровані рівні;

– Statistics – зберігає детальну статистику проходження.

Зв’язки між сутностями можна подати у вигляді ER-діаграми(Рисунок 3.1)

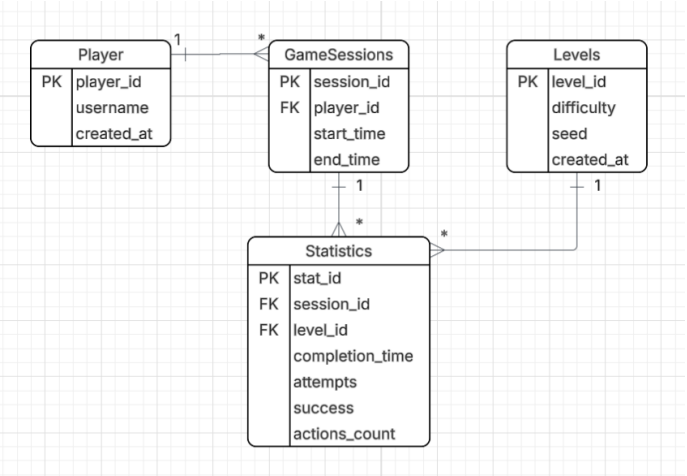


Рисунок 3.1 – ER-діаграма бази даних ігрового застосунку

3.2.2 Опис таблиць та SQL-реалізація

Таблиця Players

Дана таблиця призначена для збереження інформації про гравців.

```
CREATE TABLE Players (
    player_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Пояснення:

– player_id – унікальний ідентифікатор гравця

– username – ім’я користувача

– created_at – дата створення профілю

Таблиця GameSessions

Зберігає інформацію про окремі ігрові сесії.

```
CREATE TABLE GameSessions (
```

```
    session_id INTEGER PRIMARY KEY AUTOINCREMENT,
    player_id INTEGER,
    start_time DATETIME,
    end_time DATETIME,
    FOREIGN KEY (player_id) REFERENCES Players(player_id)
);
```

Пояснення:

– кожна сесія прив’язана до гравця

– дозволяє аналізувати тривалість гри

Таблиця Levels

Містить інформацію про згенеровані рівні.

```
CREATE TABLE Levels (
    level_id INTEGER PRIMARY KEY AUTOINCREMENT,
    difficulty INTEGER,
    seed INTEGER,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Пояснення:

– difficulty – рівень складності

– seed – значення генерації (для відтворення рівня)

Таблиця Statistics

Це ключова таблиця для системи адаптації складності.

```
CREATE TABLE Statistics (
    stat_id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id INTEGER,
    level_id INTEGER,
    completion_time REAL,
    attempts INTEGER,
```

```
    success BOOLEAN,
    actions_count INTEGER,
    FOREIGN KEY (session_id) REFERENCES GameSessions(session_id),
    FOREIGN KEY (level_id) REFERENCES Levels(level_id)
);
```

Пояснення:

– completion_time – час проходження

– attempts – кількість спроб

– success – чи пройдено рівень

– actions_count – кількість дій гравця

Також варто розуміти які запити будуть виконуватись під час роботи застосунку. Приклади таких запитів наведено нижче.

Додавання нового гравця

```
INSERT INTO Players (username)
VALUES ('Player1');
```

Створення нової сесії

```
INSERT INTO GameSessions (player_id, start_time)
VALUES (1, CURRENT_TIMESTAMP);
```

Додавання рівня

```
INSERT INTO Levels (difficulty, seed)
VALUES (3, 12345);
```

Запис статистики

```
INSERT INTO Statistics (session_id, level_id, completion_time,
attempts, success, actions_count)
VALUES (1, 1, 45.7, 2, 1, 30);
```

Запити для аналізу

Середній час проходження

```
SELECT AVG(completion_time)
FROM Statistics
WHERE success = 1;
```

Кількість невдалих спроб

```
SELECT COUNT(*)
FROM Statistics
WHERE success = 0;
```

Аналіз складності

```
SELECT difficulty, AVG(completion_time)
FROM Levels
JOIN Statistics USING(level_id)
GROUP BY difficulty;
```

У середовищі Unity база даних може підключатися через:

– SQLite бібліотеки (наприклад, System.Data.SQLite);

– або через локальні файли.

Алгоритм роботи:

1. При старті гри створюється або відкривається БД;

2. При початку рівня створюється запис Level;

3. Під час гри StatTracker накопичує дані;

4. Після завершення рівня дані записуються в Statistics;

5. DifficultyManager використовує ці дані;

Особливості реалізації

У даному проєкті база даних:

– не перевантажена складною логікою

– оптимізована під швидкі записи

– орієнтована на аналітику

Важливо, що:

– всі обчислення складності виконуються в коді

– БД використовується як джерело даних

Розроблена база даних забезпечує ефективне збереження та обробку інформації, необхідної для функціонування ігрового застосунку. Вона підтримує ключові механізми системи, зокрема адаптацію складності та аналіз поведінки користувача. Обрана структура є простою, масштабованою та добре інтегрується з середовищем Unity, що робить її оптимальним рішенням для даного проєкту.

3.3 Вимоги до технічних та програмних засобів

Для забезпечення стабільної роботи розробленого ігрового застосунку необхідно визначити вимоги до технічного та програмного середовища. Оскільки застосунок реалізує фізичне моделювання, процедурну генерацію рівнів та систему адаптації складності, він потребує достатніх обчислювальних ресурсів для обробки ігрової логіки в режимі реального часу.

Вимоги до системи можна умовно поділити на дві категорії: мінімальні та рекомендовані. Мінімальні вимоги визначають конфігурацію, за якої застосунок може працювати коректно, тоді як рекомендовані забезпечують комфортний ігровий процес без затримок. Вимоги наведено у таблиці 3.1.

Таблиця 3.1 – Мінімальні та рекомендовані системні вимоги

Параметр	Мінімальні вимоги	Рекомендовані
Процесор	Intel Core i3 або краще	Intel Core i5 або AMD Ryzen 5 або краще
RAM	4 ГБ	8ГБ
Графіка	Інтегрований графічний адаптер з підтримкою DirectX 10 або вище.	дискретна відеокарта початкового або середнього рівня з підтримкою сучасних графічних бібліотек
Вільного місця на диску	500 МБ	1 ГБ
Операційна система	Windows 10/11	

До програмних засобів, необхідних для розробки застосунку, належить середовище Unity, яке використовується як основна платформа для створення гри. Воно забезпечує інструменти для роботи з графікою, фізикою, анімаціями та обробкою подій. Мова програмування C# використовується для реалізації логіки застосунку, взаємодії між компонентами та обробки ігрових сценаріїв.

Для розробки та редагування програмного коду використовується інтегроване середовище розробки, таке як Visual Studio або Visual Studio Code. Ці інструменти забезпечують підтримку синтаксису, налагодження програм та аналіз помилок, що значно підвищує ефективність процесу розробки.

Для роботи з базою даних використовується вбудована або локальна СКБД SQLite, яка не потребує окремого серверного розгортання. Вона дозволяє зберігати інформацію про гравця, результати проходження рівнів та статистику для системи адаптації складності. Використання SQLite є доцільним у даному проєкті через її простоту, швидкість роботи та легку інтеграцію з Unity.

Важливою вимогою до програмного забезпечення є підтримка фізичного рушія, який забезпечує моделювання взаємодії об’єктів. У даному застосунку

використовується вбудований фізичний рушій Unity, що базується на бібліотеці PhysX. Він забезпечує обробку зіткнень, гравітації, сил та інших фізичних явищ у режимі реального часу.

Також необхідною є підтримка бібліотек для роботи з користувацьким інтерфейсом. Unity UI дозволяє створювати меню, панелі, кнопки та інші елементи взаємодії з користувачем. Це забезпечує зручність використання застосунку та зрозумілість інтерфейсу.

З огляду на те, що застосунок реалізує систему збору статистики, важливою є підтримка механізмів запису та обробки даних. Для цього використовуються стандартні засоби роботи з файлами та базами даних у середовищі C#. Вимогою є забезпечення швидкого запису даних без впливу на продуктивність ігрового процесу.

Особливу увагу слід приділити вимогам до продуктивності. Оскільки фізичний рушій виконує обчислення в режимі реального часу, система повинна забезпечувати стабільну частоту кадрів (не менше 30 FPS для мінімальних вимог і 60 FPS для рекомендованих). Це дозволяє уникнути затримок та забезпечує комфортний ігровий досвід.

Крім того, важливою вимогою є стабільність роботи застосунку. Програма повинна коректно обробляти помилки, уникати аварійного завершення та забезпечувати збереження даних користувача навіть у випадку нештатних ситуацій.

Таким чином, визначені вимоги до технічних та програмних засобів забезпечують стабільну, ефективну та зручну роботу ігрового застосунку. Вони враховують особливості реалізації фізичних процесів, процедурної генерації рівнів та адаптації складності, що є ключовими компонентами даного проєкту.

3.4 Тестування програмного забезпечення

3.4.1 Вибір та обґрунтування методів тестування застосунку

Тестування ПЗ – це процес перевірки програмного продукту з метою виявлення дефектів, помилок та недоліків перед його випуском на ринок або в експлуатацію. Цей процес охоплює запуск програми з різними вхідними даними та умовами, а також аналіз реакції програми на ці дані. Мета тестування – це підтвердження правильності роботи програми відповідно до вимог до неї, а також забезпечення високої якості та надійності програмного продукту [29]. Для ігрових застосунків, що містять фізичні симуляції, процедурну генерацію рівнів та адаптивні алгоритми складності, тестування має особливе значення, оскільки навіть незначні помилки можуть суттєво впливати на ігровий процес.

У межах даного проєкту застосовується комплексний підхід до тестування, який включає кілька основних методів: модульне тестування, інтеграційне тестування та функціональне тестування.

Модульне тестування (Unit testing) – тестування кожної атомарної функціональності додатку окремо, в штучно створеному середовищі [30]. У даному випадку перевіряються модулі LevelGenerator, PhysicsEngine, DifficultyManager та StatTracker. Основна увага приділяється коректності реалізації окремих функцій, таких як генерація рівнів, фізичні розрахунки та збір статистики. Це дозволяє виявляти помилки на ранніх етапах розробки.

Інтеграційне тестування – критично важливий етап розробки ПЗ, коли перевірені окремо модулі збирають та перевіряють як єдине ціле. Воно слідує за модульним тестуванням і передує системному тестуванню. Головне завдання цього етапу – знайти дефекти, які з’являються під час взаємодії частин системи або у зв’язку з зовнішніми компонентами: базами даних, API та іншими сервісами [31]. У межах проєкту перевіряється коректність обміну даними між GameController, системою генерації рівнів, фізичним рушієм та системою

адаптації складності. Зокрема, аналізується передача даних між StatTracker та DifficultyManager, а також взаємодія LevelGenerator і PhysicsEngine.

Функціональне тестування перевіряє програмну систему на відповідність її функціональним вимогам, перевіряючи кожну функцію з визначеними вхідними даними та перевіряючи очікувані результати [32]. У цьому випадку перевіряється повний ігровий цикл: запуск гри, генерація рівня, взаємодія користувача з об’єктами, проходження рівня та збереження результатів. Окрему увагу приділено роботі механізму адаптації складності.

Для тестування також застосовується метод чорної скриньки (black-box testing), який перевіряє функціональність системи без доступу до її внутрішньої структури коду. У цьому підході тестувальник працює як кінцевий користувач, оцінюючи, чи правильно система реагує на вхідні дані. Основна мета – перевірити, чи відповідає ПЗ бізнес-вимогам і очікуванням користувачів [33].

Додатково використовується ручне тестування, при якому тестовий кейс виконується тестувальником вручну без допомоги будь-яких автоматизованих інструментів [34]. Воно дозволяє оцінити поведінку системи в нестандартних сценаріях та загальне відчуття геймплею.

Вибір саме цих методів тестування обумовлений специфікою розробленого застосунку. Оскільки система поєднує процедурну генерацію, фізичне моделювання та адаптивні алгоритми, необхідно перевіряти як окремі компоненти, так і їх взаємодію в комплексі. Комбінація модульного, інтеграційного та функціонального тестування дозволяє забезпечити високу якість програмного продукту та зменшити ймовірність виникнення критичних помилок під час використання.

Таким чином, обрані методи тестування є оптимальними для даного типу програмного забезпечення та забезпечують повну перевірку коректності роботи ігрового застосунку на всіх рівнях його реалізації.

3.4.2 Валідація та верифікація програмного забезпечення

Валідація та верифікація є важливими етапами перевірки програмного забезпечення, що дозволяють оцінити відповідність реалізації вимогам технічного завдання та очікуванням користувача.

Верифікація – це процес оцінки відповідності програмного продукту або системи вимогам та специфікаціям [35]. У межах даного проєкту основна увага приділяється окремим модулям системи: GameController, LevelGenerator, PhysicsEngine, DifficultyManager та StatTracker. Для цього використовується модульне тестування.

Валідація – це оцінка відповідності програмного продукту або системи потребам клієнтів [35]. У цьому випадку аналізується коректність ігрового процесу в цілому, включаючи генерацію рівнів, фізичні взаємодії та адаптацію складності проходження рівня.

У процесі тестування було визначено набір основних тестових випадків, які охоплюють ключові функціональні сценарії системи(Табл. 3.2 – 3.6).

Таблиця 3.2 – Тестовий випадок 1: Генерація рівня

Параметр	Значення
Мета	Перевірка коректності створення рівня
Вхідні дані	Рівень складності = середній
Очікуваний результат	Згенерований прохідний рівень без критичних помилок
Результат	Рівень створено, об’єкти коректно розміщені, рівень в більшості випадків прохідний

Таблиця 3.3 – Тестовий випадок 2: Фізична взаємодія об’єктів

Параметр	Значення
Мета	Перевірка роботи фізичного рушія
Вхідні дані	Зіткнення двох об’єктів
Очікуваний результат	Коректна реакція на зіткнення (відскок, рух)
Результат	Фізика працює стабільно

Таблиця 3.4 – Тестовий випадок 3: Адаптація складності

Параметр	Значення
Мета	Перевірка роботи DifficultyManager
Вхідні дані	Швидке проходження рівня
Очікуваний результат	Підвищення складності наступного рівня
Результат	Складність підвищено та передано генератору рівнів

Таблиця 3.5 – Тестовий випадок 4: Збір статистики

Параметр	Значення
Мета	Перевірка роботи StatTracker
Вхідні дані	Дії гравця під час рівня
Очікуваний результат	Запис усіх подій у систему статистики
Результат	Дані збережені для генерації наступного рівня

Таблиця 3.6 – Тестовий випадок 5: Перехід між станами гри

Параметр	Значення
Мета	Перевірка GameController
Вхідні дані	Завершення рівня
Очікуваний результат	Перехід у стан результатів
Результат	Перехід відбувається після завершення рівня, відображаються результати проходження

Отримані результати свідчать про загальну коректність реалізації основних функціональних модулів системи та їх взаємодії. У більшості перевірених сценаріїв система поводиться відповідно до очікуваної логіки.

3.4.3 Аналіз результатів тестування

Проведене тестування дозволило оцінити роботу основних функціональних модулів розробленої адаптивної фізичної головоломки з процедурною генерацією рівнів. Аналіз базується на тест-кейсах та спостереженні поведінки системи у середовищі Unity.

У ході тестування було перевірено роботу генератора рівнів, фізичного рушія, системи адаптації складності, модуля збору статистики та ігрового

контролера. Окрема увага приділялась взаємодії між модулями та стабільності переходів між станами гри (табл. 3.7).

Таблиця 3.7 – Результати тестування основних модулів

Компонент системи	Тип перевірки	Результат	Висновок
LevelGenerator	Генерація рівня	Рівні створюються з незначною варіативністю структури	Загалом відповідає вимогам
PhysicsEngine	Фізичні взаємодії	Симуляція стабільна, без критичних збоїв	Працює коректно в типових сценаріях
DifficultyManager	Адаптація складності	Параметри змінюються залежно від результатів гри	Поведінка відповідає очікуванням
StatTracker	Збір даних	Основні ігрові події фіксуються	Дані придатні для подальшого аналізу
GameController	Керування станами	Перемикання станів відбувається коректно	Система функціонує стабільно

Результати тестування загалом підтверджують працездатність основних сценаріїв використання системи. Процедурна генерація рівнів функціонує стабільно та забезпечує створення різноманітних ігрових ситуацій без критичних помилок у структурі рівнів. У більшості випадків згенеровані рівні є прохідними, що свідчить про коректність базових алгоритмів генерації.

Фізичний рушій демонструє стабільну поведінку об’єктів у стандартних сценаріях взаємодії. Зіткнення, гравітація та інші фізичні ефекти обробляються коректно без суттєвих відхилень.

Система адаптації складності в цілому реагує на поведінку користувача. У випадках швидкого проходження рівнів спостерігається тенденція до підвищення складності наступних сценаріїв, тоді як при складнішому проходженні параметри можуть знижуватися, що сприяє вирівнюванню ігрового процесу.

Модуль збору статистики фіксує основні події ігрового процесу, включаючи час проходження рівнів, кількість дій гравця та результати взаємодій.

Під час тестування також були виявлені окремі особливості:

– при великій кількості фізичних об’єктів можливе зниження продуктивності;

– деякі згенеровані рівні потребують додаткового балансування складності;

– для точнішої адаптації системі може знадобитися більший обсяг накопичених даних.

У цілому результати тестування дозволяють зробити висновок, що система функціонує відповідно до основних вимог і забезпечує коректну роботу ключових модулів. Виявлені особливості не є критичними та можуть бути враховані у подальшій оптимізації.

3.5 Висновки

У межах третього розділу було здійснено повну програмну реалізацію основних модулів ігрового застосунку та проведено їх комплексне тестування. Розробка базувалася на попередньо визначеній архітектурі та принципах модульності, що дозволило забезпечити логічну структурованість системи та зручність її подальшого супроводу.

Під час реалізації програмного забезпечення було створено ключові компоненти ігрової системи, зокрема модулі керування ігровим процесом, генерації рівнів, фізичного моделювання, адаптації складності та збору статистики. Кожен із цих модулів виконує чітко визначену функцію та взаємодіє з іншими через задані інтерфейси, що відповідає принципам компонентного підходу.

Особливу увагу приділено реалізації процедурної генерації рівнів та системи адаптації складності, оскільки саме ці механізми визначають унікальність ігрового процесу та забезпечують його варіативність. Також реалізовано механізм збору статистики, який дозволяє аналізувати поведінку користувача та використовувати отримані дані для коригування ігрового досвіду.

У процесі тестування було перевірено коректність роботи всіх основних функціональних модулів системи. Результати показали, що програмне забезпечення функціонує стабільно, виконує поставлені завдання та відповідає вимогам технічного завдання. Виявлені незначні особливості не впливають на загальну працездатність системи та можуть бути усунені в процесі подальшої оптимізації.

Таким чином, у результаті виконання третього розділу було отримано повністю функціональний ігровий застосунок, який реалізує основні механіки адаптивної фізичної головоломки з процедурною генерацією рівнів. Отримані результати підтверджують правильність обраних архітектурних рішень та ефективність використаних методів розробки і тестування програмного забезпечення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено комплексне дослідження та розробку програмного забезпечення у вигляді адаптивної фізичної головоломки з процедурною генерацією рівнів. Робота охоплює всі основні етапи створення сучасного ігрового застосунку: від аналізу предметної області та постановки задачі до проєктування архітектури, реалізації програмних модулів і тестування готової системи.

На першому етапі було проведено детальний аналіз предметної області. Розглянуто сучасні тенденції розвитку індустрії відеоігор, зокрема жанру фізичних головоломок, а також роль процедурної генерації контенту та адаптивних систем складності. Встановлено, що сучасні ігрові застосунки все частіше переходять до використання алгоритмічного створення контенту, що дозволяє підвищити реіграбельність та індивідуалізацію ігрового досвіду. Окремо проаналізовано існуючі програмні рішення, які частково реалізують подібні механіки, однак не поєднують одночасно фізичне моделювання, процедурну генерацію та адаптацію складності в єдиній системі.

На основі проведеного аналізу було сформульовано вимоги до програмного продукту. Визначено функціональні вимоги, що включають генерацію рівнів, фізичну взаємодію об’єктів, адаптацію складності та збір статистики користувача, а також нефункціональні вимоги, пов’язані з продуктивністю, стабільністю, масштабованістю та зручністю використання. Це дозволило сформувати чітке технічне завдання для подальшого проєктування системи.

Другий розділ роботи був присвячений проєктуванню програмного забезпечення. Було обґрунтовано вибір компонентної архітектури як основи системи, що забезпечує модульність, гнучкість і можливість подальшого розширення функціоналу. У межах архітектури виділено основні компоненти: GameController, LevelGenerator, PhysicsEngine, DifficultyManager та StatTracker.

Кожен із них виконує окрему функцію та взаємодіє з іншими через чітко визначені інтерфейси.

Також було виконано декомпозицію системи, описано залежності між модулями та визначено структуру взаємодії компонентів. Окрему увагу приділено проєктуванню інтерфейсів, які забезпечують узгоджену взаємодію між підсистемами та зменшують рівень зв’язаності коду. Використання патернів проєктування, таких як Singleton, Observer, Factory та State, дозволило підвищити якість архітектурного рішення та спростити реалізацію складної логіки гри.

У третьому розділі було реалізовано основні програмні модулі системи. Розроблено механізм процедурної генерації рівнів, фізичну модель взаємодії об’єктів, систему адаптації складності та модуль збору статистики. Реалізація виконувалась із використанням середовища Unity та мови програмування C#, що забезпечило інтеграцію з готовим фізичним рушієм і графічною підсистемою.

Особливу увагу було приділено тестуванню програмного забезпечення. Проведено верифікацію та валідацію основних функціональних модулів, розроблено тестові сценарії та проаналізовано результати їх виконання. Тестування показало, що система стабільно виконує основні функції, коректно генерує рівні, забезпечує фізично правдоподібну поведінку об’єктів та адаптує складність відповідно до дій користувача. Виявлені незначні особливості не впливають на загальну працездатність системи та можуть бути усунені в процесі подальшої оптимізації.

Отримані результати підтверджують ефективність обраного підходу до проєктування та реалізації програмного забезпечення. Поєднання процедурної генерації, фізичного моделювання та адаптивної складності дозволило створити гнучку ігрову систему, яка забезпечує унікальний користувацький досвід при кожному проходженні.

Таким чином, мета кваліфікаційної роботи досягнута, а поставлені завдання виконані в повному обсязі. Розроблений програмний продукт може бути використаний як базова платформа для подальшого розвитку ігрових механік, розширення функціоналу та впровадження більш складних алгоритмів адаптації та генерації контенту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ігрові жанри [Електронний ресурс] Режим доступу: https://training.qatestlab.com/blog/technical-articles/games-genres/ (дата звернення 11.02.2026)

2. Ігри-головоломки та їхня користь для розумової діяльності [Електронний ресурс] Режим доступу: https://blog.acer.com/ua/discussion/2545/igri-golovolomki-ta-yihnya-korist-dlya-rozumovoyi-diyalnosti (дата звернення 11.02.2026)

3. Динамічна складність: Огляд адаптивного ШІ у відеоіграх [Електронний ресурс] Режим доступу: https://blog.acer.com/ua/discussion/2123/dinamichna-skladnist-oglyad-adaptivnogo-shi-u-videoigrah (дата звернення 11.02.2026)

4. Ігрові світи: процедурна генерація, сторітелінг у середовищі та наративні ігри [Електронний ресурс] Режим доступу: https://idcgames.com/uk/blog/%D1%96%D0%B3%D1%80%D0%BE%D0%B2%D1%96-%D1%81%D0%B2%D1%96%D1%82%D0%B8-%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%B0-%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8F-%D1%81%D1%82/ (дата звернення 13.02.2026)

5. Unity Documentation. Physics Overview. [Електронний ресурс] Режим доступу:https://docs.unity3d.com/Packages/com.unity.physics@6.5/manual/index.html (дата звернення 13.02.2026)

6. Angry Birds Classic [Електронний ресурс] Режим доступу: https://angry-birds.en.uptodown.com/android (дата звернення 15.02.2026)

7. Human Fall Flat [Електронний ресурс] Режим доступу: https://play.google.com/store/apps/details?id=com.and.games505.humanfallflat&hl=uk (дата звернення 15.02.2026)

8. Функціональні вимоги [Електронний ресурс] Режим доступу: https://www.idealist.in.ua/2026/01/%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8/ (дата звернення 18.02.2026)

9. Нефункціональні вимоги: приклади, типи, підходи [Електронний ресурс] Режим доступу: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/ (дата звернення 18.02.2026)

10. Що таке діаграма варіантів використання UML: символи, шаблони, інструмент і посібник [Електронний ресурс] Режим доступу: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/ (дата звернення 20.02.2026)

11. Діаграма послідовності (Sequence Diagrams) [Електронний ресурс] Режим доступу: https://www.maxzosim.com/sequence-diagrams/ (дата звернення 20.02.2026)

12. Архітектура програмного забезпечення: все що треба знати [Електронний ресурс] Режим доступу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення 18.04.2026)

13. Розуміння компонентної архітектури Unity [Електронний ресурс] Режим доступу: https://uk.sharpcoderblog.com/blog/understanding-unitys-component-based-architecture (дата звернення 18.04.2026)

14. Про патерни проектування [Електронний ресурс] Режим доступу: https://foxminded.ua/paterny-proektuvannia/ (дата звернення 18.04.2026)

15. 10 шаблонів проектування програмного забезпечення, які повинен знати кожен розробник [Електронний ресурс] Режим доступу: https://stfalcon.com/uk/blog/post/10-Software-Design-Patterns-Every-Developer-Must-Know (дата звернення 18.04.2026)

16. Фабричний метод (Factory Method) [Електронний ресурс] Режим доступу: https://abitap.com/2-1-fabrychnyj-metod-factory-method/ (дата звернення 18.04.2026)

17. Чому SOLID — важлива складова мислення програміста. Розбираємося на прикладах з кодом [Електронний ресурс] Режим доступу: https://dou.ua/lenta/articles/solid-principles/ (дата звернення 18.04.2026)

18. Принцип відкритості/закритості [Електронний ресурс] Режим доступу: https://abitap.com/5-2-prynczyp-vidkrytosti-zakrytosti/ (дата звернення 18.04.2026)

19. Декомпозиція в програмуванні: ключ до розуміння складних систем [Електронний ресурс] Режим доступу: https://foxminded.ua/dekompozytsiia-v-prohramuvanni/ (дата звернення 18.04.2026)

20. Rigidbody [Електронний ресурс] Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/class-Rigidbody.html (дата звернення 19.04.2026)

21. Colliders [Електронний ресурс] Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/CollidersOverview.html (дата звернення 19.04.2026)

22. Creation and Use of Scripts [Електронний ресурс] Режим доступу: https://docs.unity3d.com/ru/2019.4/Manual/CreatingAndUsingScripts.html (дата звернення 19.04.2026)

23. Принцип інверсії залежностей (Dependency Inversion Principle) [Електронний ресурс] Режим доступу: https://it-blog.in.ua/pryntsyp-inversiyi-zalezhnostey-dependency-inversion-principle/ (дата звернення 19.04.2026)

24. Dependecy injection та Singletone. Все, що треба знати, і ще трошки [Електронний ресурс] Режим доступу: https://medium.com/@ivanfomenko/dependecy-injection-%D1%82%D0%B0-singletone-%D0%B2%D1%81%D0%B5-%D1%89%D0%BE-%D1%82%D1%80%D0%B5%D0%B1%D0%B0-

%D0%B7%D0%BD%D0%B0%D1%82%D0%B8-%D1%96-%D1%89%D0%B5-%D1%82%D1%80%D0%BE%D1%88%D0%BA%D0%B8-2e738373eb94 (дата звернення 24.04.2026)

25. Що таке бази даних? [Електронний ресурс] Режим доступу: https://university.sigma.software/what-is-database/ (дата звернення 30.05.2026)

26. Types of Databases (Database Examples and Use Cases) [Електронний ресурс] Режим доступу: https://host-world.com.ua/types-of-databases-database-examples-and-use-cases (дата звернення 30.05.2026)

27. Бази даних: типи та особливості [Електронний ресурс] Режим доступу: https://itedu.center/ua/blog/articles/databases-types-and-features/ (дата звернення 30.05.2026)

28. Типи баз даних: особливості, відмінності та приклади [Електронний ресурс] Режим доступу: https://dou.ua/lenta/articles/types-of-databases/ (дата звернення 30.05.2026)

29. Що таке тестування ПЗ, його етапи, види, інструменти [Електронний ресурс] Режим доступу: https://university.sigma.software/what-is-software-testing/ (дата звернення 30.05.2026)

30. Модульне тестування [Електронний ресурс] Режим доступу: https://qalight.ua/baza-znaniy/modulne-testuvannya/ (дата звернення 30.05.2026)

31. Що таке інтеграційне тестування [Електронний ресурс] Режим доступу: https://fnx.com.ua/ua/articles/terms/chto-takoe-integratsionnoe-testirovanie-tsel-vidy-i-primery (дата звернення 30.05.2026)

32. Що таке функціональне тестування? Типи та приклади [Електронний ресурс] Режим доступу: https://www.guru99.com/uk/functional-testing.html (дата звернення 30.05.2026)

33. Що таке тестування за методом “чорної скриньки”? [Електронний ресурс] Режим доступу: https://university.sigma.software/black-box-testing/ (дата звернення 30.05.2026)

34. Ручне тестування – що це таке, типи, процеси, підходи, інструменти та інше! [Електронний ресурс] Режим доступу: https://www.zaptest.com/uk/%D1%80%D1%83%D1%87%D0%BD%D0%B5-%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D1%89%D0%BE-%D1%86%D0%B5-%D1%82%D0%B0%D0%BA%D0%B5-%D1%82%D0%B8%D0%BF%D0%B8-%D0%BF%D1%80%D0%BE%D1%86 (дата звернення 30.05.2026)

35. Що таке валідація та верифікація в розробці ПЗ [Електронний ресурс] Режим доступу: https://www.it-notes.wiki/software-testing/what-is-validation-and-verification/ (дата звернення 30.05.2026)

ДОДАТОК А

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А.1 Код головного контролера гри GameController

```
using UnityEngine;

public class GameController : MonoBehaviour
{
    public static GameController Instance;

    public LevelGenerator levelGenerator;
    public DifficultyManager difficultyManager;
    public StatTracker statTracker;

    private int currentLevel = 0;

    private void Awake()
    {
        if (Instance == null) Instance = this;
        else Destroy(gameObject);
    }

    private void Start()
    {
        StartGame();
    }

    public void StartGame()
    {
        currentLevel = 1;
```

```
        LoadLevel();
    }

    public void LoadLevel()
    {
        levelGenerator.GenerateLevel(currentLevel);
        statTracker.StartLevelTracking(currentLevel);
    }

    public void CompleteLevel(float time)
    {
        statTracker.EndLevelTracking(time);

        int newDifficulty =
difficultyManager.CalculateDifficulty(statTracker.GetLastResult());

        currentLevel++;
        LoadLevel();
    }

    public void GameOver()
    {
        Debug.Log(«Game Over»);
    }
}
```

A.2 Код генератора рівнів LevelGenerator

```
using UnityEngine;

public class LevelGenerator : MonoBehaviour
{
    public GameObject obstaclePrefab;
```

```
    public GameObject goalPrefab;

    public void GenerateLevel(int level)
    {
        ClearLevel();

        int obstacleCount = 3 + level;

        for (int i = 0; i < obstacleCount; i++)
        {
            Vector3 pos = new Vector3(Random.Range(-5, 5), 0,
Random.Range(-5, 5));
            Instantiate(obstaclePrefab, pos, Quaternion.identity);
        }

        Instantiate(goalPrefab, new Vector3(6, 0, 6),
Quaternion.identity);

        Debug.Log(«Level generated: « + level);
    }

    private void ClearLevel()
    {
        foreach (GameObject obj in
GameObject.FindGameObjectsWithTag(«LevelObject»))
        {
            Destroy(obj);
        }
    }
}
```

А.3 Код менеджера складності DifficultyManager

```
using UnityEngine;
```

```
public class DifficultyManager : MonoBehaviour
{
    public int CalculateDifficulty(LevelResult result)
    {
        int difficulty = result.time < 30 ? 2 : 1;

        if (result.failCount > 3)
            difficulty = 1;

        if (result.time < 15 && result.failCount == 0)
            difficulty = 3;

        Debug.Log(«New difficulty: « + difficulty);

        return difficulty;
    }
}

public class LevelResult
{
    public float time;
    public int failCount;
}
```

А.4 Код збирача статистики StatTracker

```
using UnityEngine;

public class StatTracker : MonoBehaviour
{
    private float startTime;
    private int failCount;
```

```
    private LevelResult lastResult;

    public void StartLevelTracking(int level)
    {
        startTime = Time.time;
        failCount = 0;
    }

    public void RegisterFail()
    {
        failCount++;
    }

    public void EndLevelTracking(float manualTime = -1)
    {
        float time = manualTime > 0 ? manualTime : Time.time -
startTime;

        lastResult = new LevelResult
        {
            time = time,
            failCount = failCount
        };
    }

    public LevelResult GetLastResult()
    {
        return lastResult;
    }
}
```

А.5 Код користувацького інтерфейсу SimpleUIController

```
using UnityEngine;
using UnityEngine.UI;

public class SimpleUIController : MonoBehaviour
{
    public Text levelText;

    void Update()
    {
        if (GameController.Instance != null)
        {
            levelText.text = "Level: " + Time.frameCount;
        }
    }
}
```

А.6 Код фізики PhysicsController

```
using UnityEngine;

public class PhysicsController : MonoBehaviour
{
    public float forceMultiplier = 10f;

    // застосування сили до об'єкта
    public void ApplyForce(GameObject obj, Vector3 direction)
    {
        Rigidbody rb = obj.GetComponent<Rigidbody>();

        if (rb != null)
        {
            rb.AddForce(direction * forceMultiplier,
ForceMode.Impulse);
        }
```

```
    }

    // зупинка об’єкта
    public void StopObject(GameObject obj)
    {
        Rigidbody rb = obj.GetComponent<Rigidbody>();

        if (rb != null)
        {
            rb.linearVelocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
        }
    }

    // перевірка чи об’єкт впав / вийшов за межі
    public bool IsOutOfBounds(GameObject obj, float minY = -10f)
    {
        return obj.transform.position.y < minY;
    }
}
```

ДОДАТОК Б

(обов’язковий)

КЕРІВНИЦТВО КОРИСТУВАЧА

Гравець керує фізичним ігровим об’єктом, який взаємодіє з ігровим середовищем за законами фізики Unity. Гра є фізичною головоломкою, у якій необхідно знаходити правильні способи подолання процедурно згенерованих рівнів.

Метою гри є досягнення фінішної зони рівня шляхом вирішення просторових та фізичних задач, що виникають у процесі проходження.

Кожен рівень складається з платформ, механізмів, перешкод та порожнин. Їх розташування формується автоматично за допомогою процедурної генерації перед початком гри. Гравець повинен використовувати інерцію, гравітацію, зіткнення та взаємодію з об’єктами для пошуку правильного маршруту проходження рівня. Помилки у виборі траєкторії руху призводять до падіння або неможливості продовження проходження, після чого рівень перезапускається.

Після завершення рівня система фіксує такі результати: час проходження рівня, кількість спроб та кількість помилок. На основі цих даних система визначає складність наступних рівнів шляхом зміни структури рівня та розташування об’єктів на ньому.

Призначення клавіш:

W, A, S, D – керування рухом фізичного ігрового об’єкта

Space – стрибок або імпульс руху

Mouse Move – керування камерою огляду рівня

ЛКМ – взаємодія з елементами рівня

E – активація механізмів та об’єктів головоломки

R – перезапуск рівня

Esc – пауза та відкриття меню

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

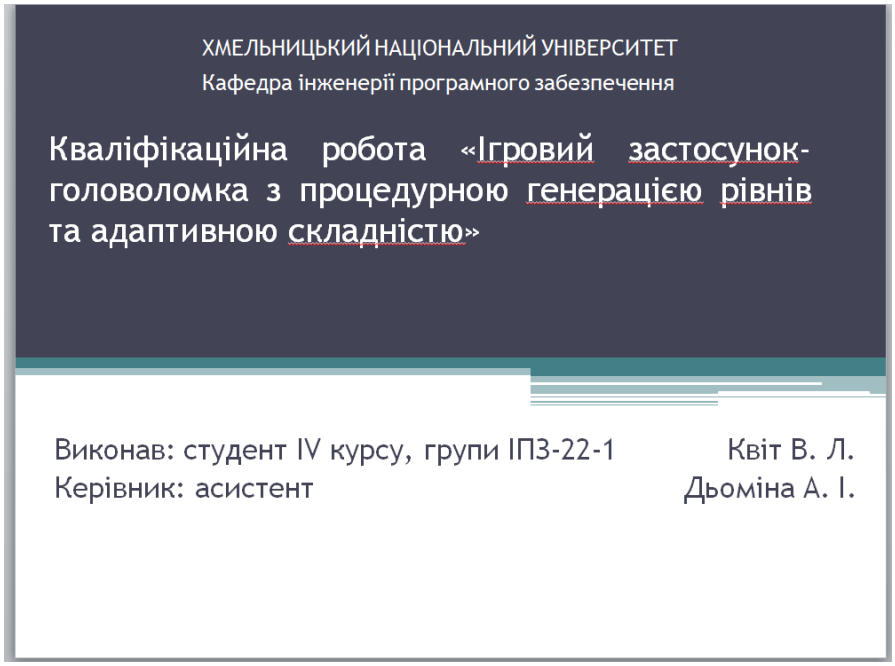


Рисунок В.1 – Слайд 1

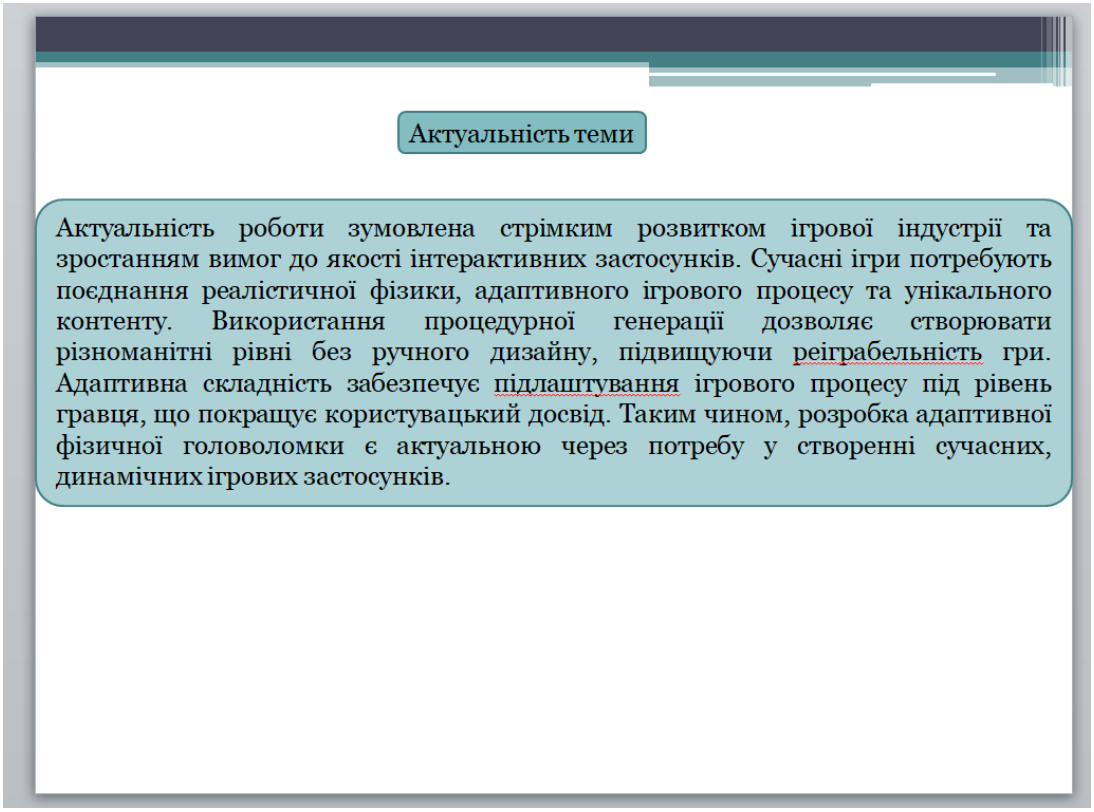


Рисунок В.2 – Слайд 2

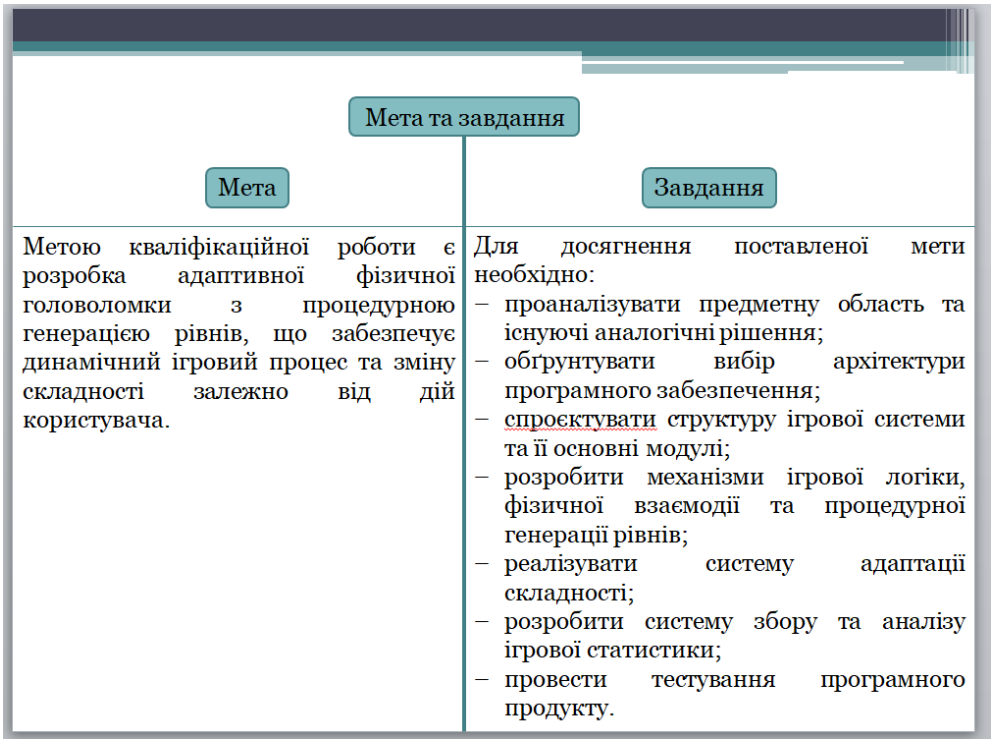


Рисунок В.3 – Слайд 3

Дослідження предметної області та постановка задачі
Змістовний аналіз предметної області її структурних та функціональних особливостей

Сучасна індустрія ігор є однією з найдинамічніших галузей ІТ, що поєднує програмну інженерію, графіку, фізичне моделювання та елементи штучного інтелекту. Відеоігри сьогодні виконують не лише розважальну, а й навчальну та розвивальну функцію.
Особливе місце займають головоломки, зокрема фізичні, де ігровий процес базується на взаємодії об'єктів за законами механіки. Такі ігри підвищують логічне мислення та аналітичні навички.
Важливими сучасними технологіями є:
– фізичні рушії, що забезпечують реалістичну взаємодію об'єктів;
– адаптивний ШІ, який підлаштовує складність під гравця;
– процедурна генерація, що створює унікальні рівні та підвищує реіграбельність.
Поєднання цих підходів дозволяє створювати динамічні ігрові системи з індивідуальним досвідом для кожного користувача, що і визначає актуальність розробки адаптивної фізичної головоломки.

Рисунок В.4 – Слайд 4

Дослідження предметної області та постановка задачі
Аналіз наявного програмного забезпечення

Angry Birds

Human: Fall Flat

Гра	Фізика	Процедурна генерація	Адаптивність	Логічна головоломка
Angry Birds	Так	Ні	Ні	Так
Human: Fall Flat	Так	Ні	Ні	частково

Як бачимо жодна з цих двох ігор не поєднує одночасно фізичне моделювання, процедурну генерацію та адаптивну складність. Це створює потребу у новому рішенні — адаптивній фізичній головоломці з процедурною генерацією рівнів, що забезпечує унікальний ігровий досвід для кожного користувача.

Рисунок В.5 – Слайд 5

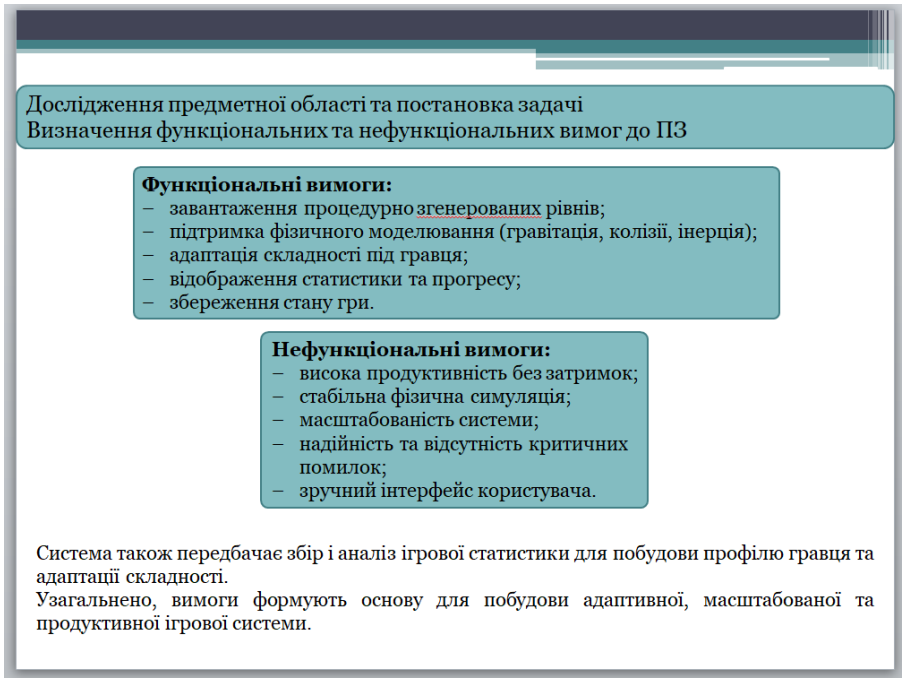


Рисунок В.6 – Слайд 6

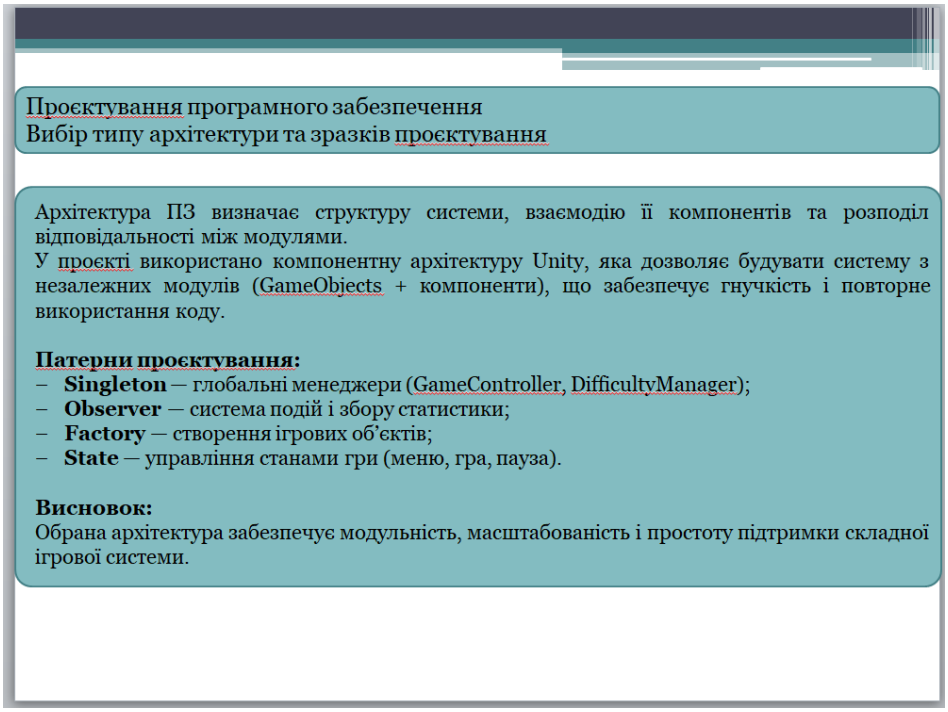


Рисунок В.7 – Слайд 7

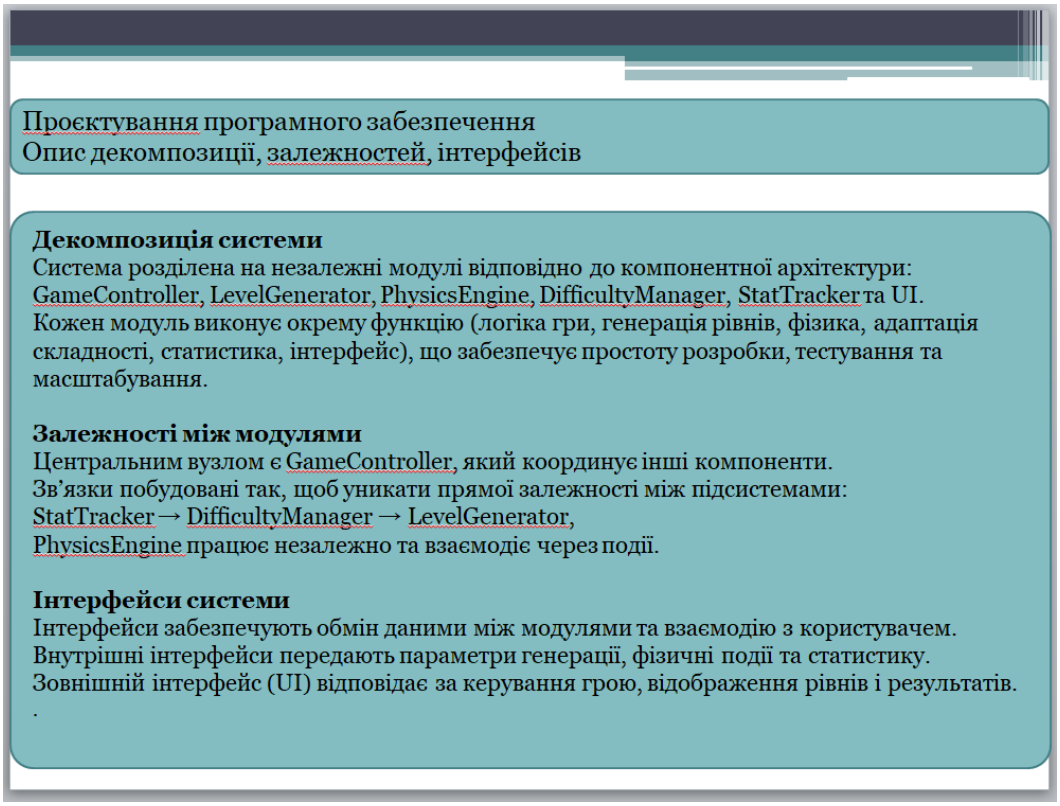


Рисунок В.8 – Слайд 8

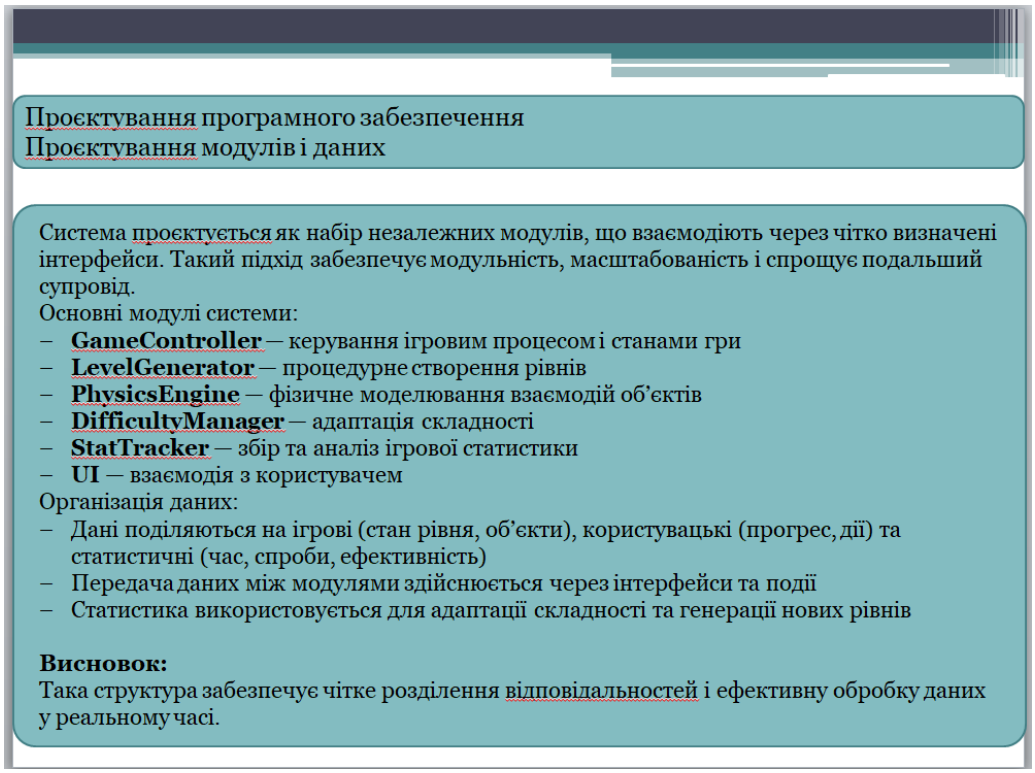


Рисунок В.9 – Слайд 9

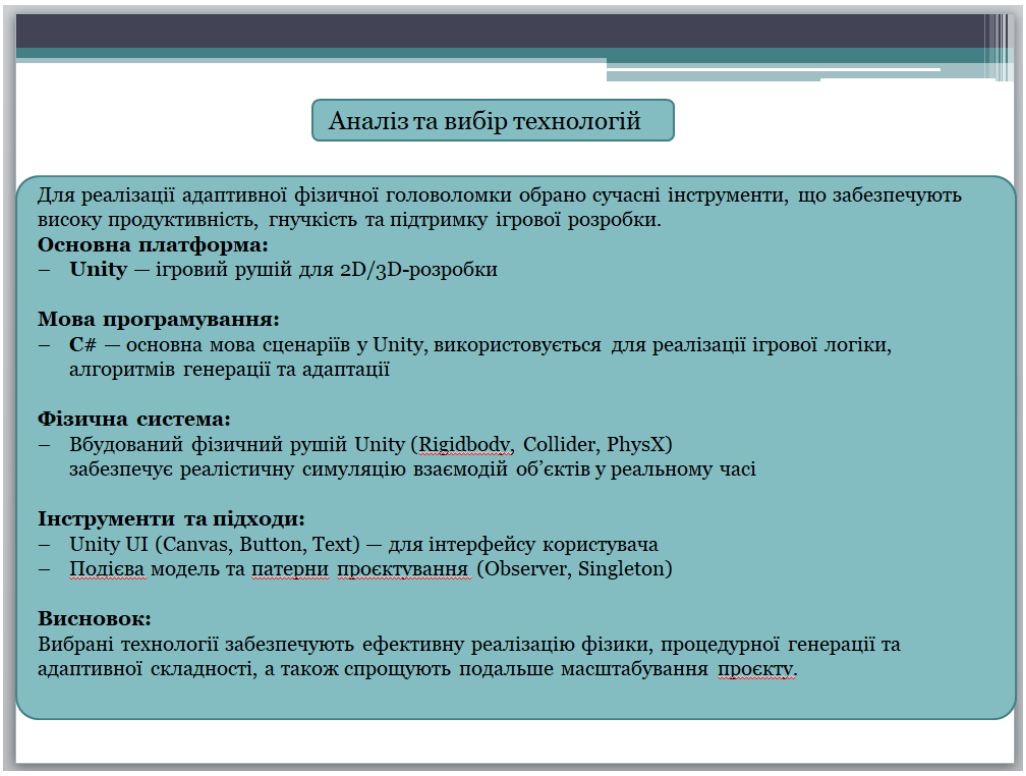


Рисунок В.10 – Слайд 10

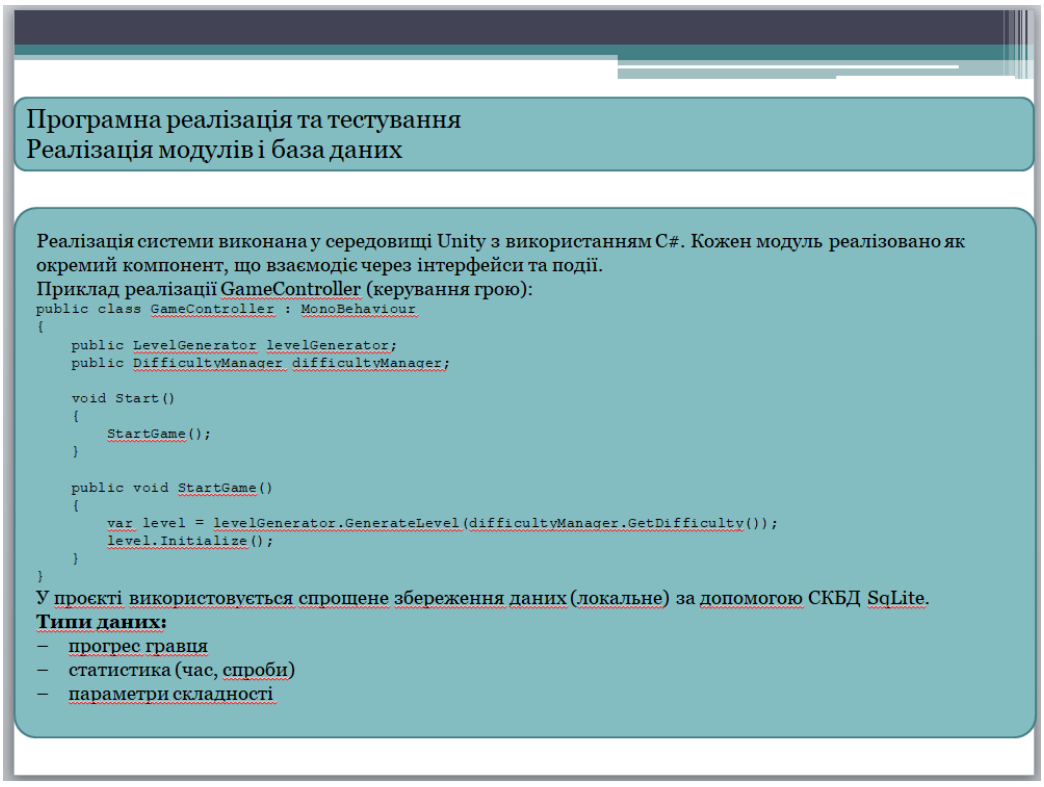


Рисунок В.11 – Слайд 11

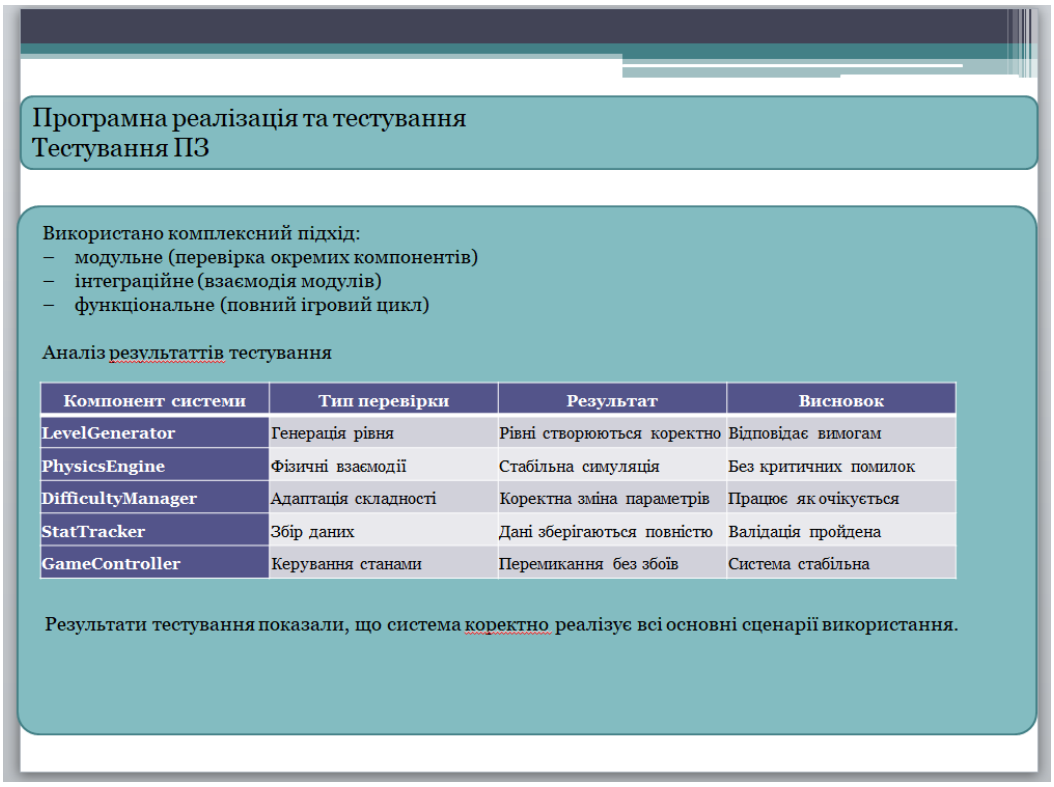


Рисунок В.12 – Слайд 12

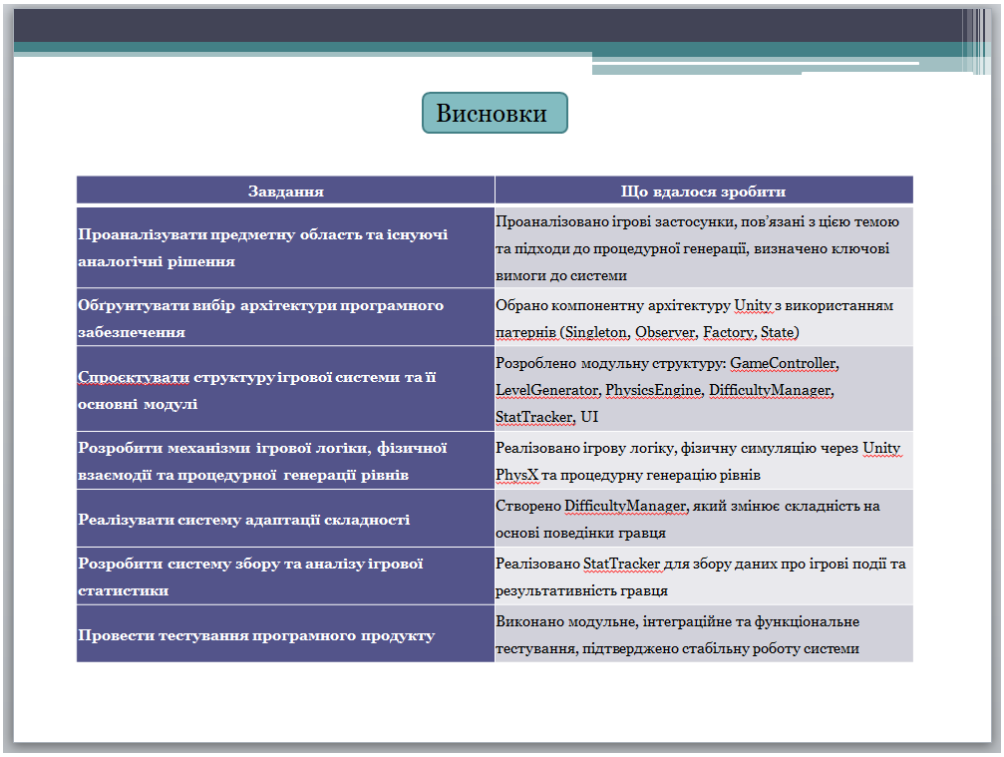
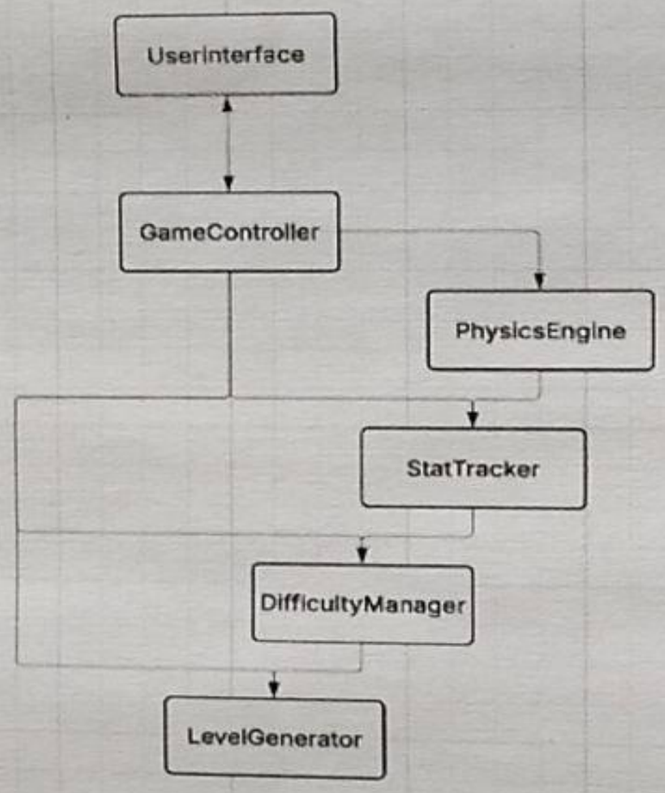
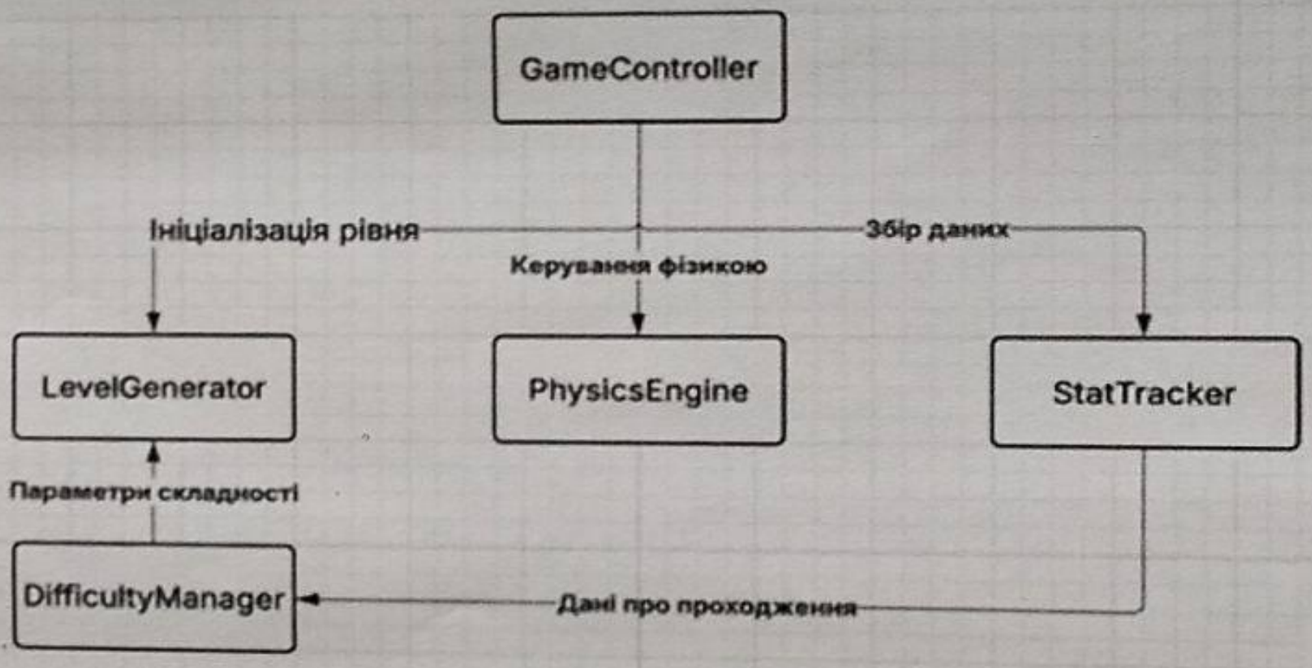


Рисунок В.13 – Слайд 13

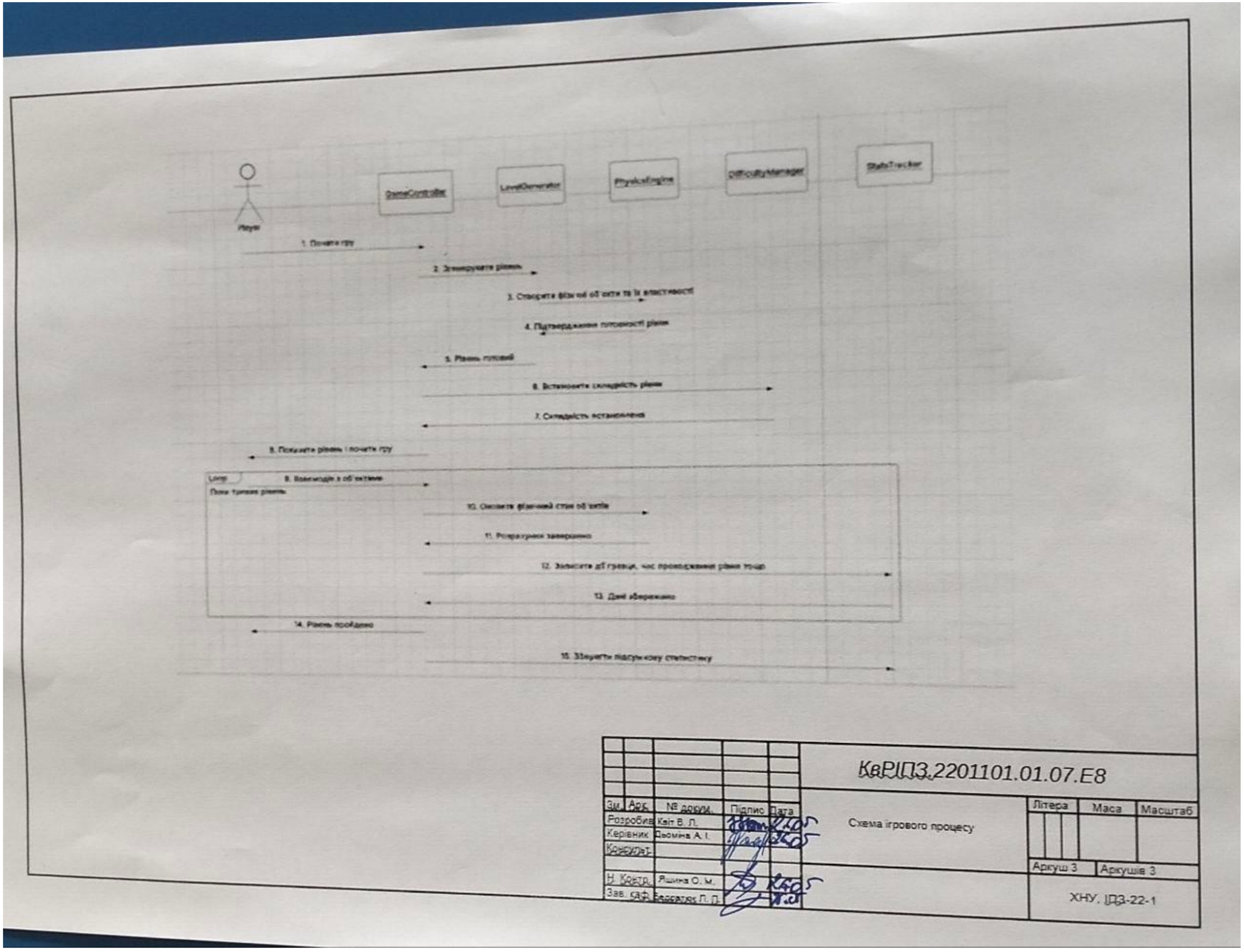
ГРАФІЧНІ МАТЕРІАЛИ



					КвРІПЗ.2201101.01.07.Е8			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Кеіт В. Л.			Структурна схема архітектури системи	Літера	Маса	Масштаб
Керівник		Дюміна А. І.						
Консульт.								
Н. Контр.		Яшина О. М.				Аркуш 1	Аркушів 3	
Зав. каф.		Бєлевцов Л. П.				ХНУ, ІПЗ-22-1		



					КвРПЗ.2201101.01.07.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Квіт В. Л.			Схема залежностей між модулями	Літера / Маса / Масштаб
Керівник		Дьоміна А. І.				
Консульт.						Аркуш 2 / Аркушів 3
Н. контр.		Яшина О. М.				
Зав. каф.		Бобрівська Л. П.				ХНУ, ІДЗ-22-1



					КвРІПЗ.2201101.01.07.Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Схема ігрового процесу	Літера	Маса	Масштаб
Розробив		Квіт В. Л.						
Керівник		Дьоміна А. І.				Аркуш 3	Аркушів 3	
Консульт.								
Н. контр.		Яшина О. М.				ХНУ, ІДЗ-22-1		
Зав. каф.		Федорець Л. Л.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Квіта Владислава Леонідовича
факультет ІТ, IVкурс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.04.26
дата

підпис

Tue Jun 09 10:39:03 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 18.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 10%**

ID: 274278 Назва: БКР_Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю Додано в БД: 2026-06-09 Автора: Владислав КВІТ Керівники: Анастасія ДЬОМІНА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	100356	911	22317 (22%)	206 (23%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269633	Назва: Переддипломна практика Додано в БД: 2026-03-02 Автора: Владислав КВІТ Керівники: ДЬОМІНА А. І., асистент Консультанти: Опоненти:	17791 (18.0%)	151 (17.0%)

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав КВІТ

Співавтор:

Назва: Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю

Науковий керівник: Анастасія ДЬОМІНА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:14.49%

Коефіцієнт подібності 2:7.11%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-06-10 02:16:08.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 10.06.26 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Квіт Владислав Леонідович

Тема Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 75

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено та проаналізовано предметну область адаптивних ігрових застосунків з процедурною генерацією рівнів. Визначено основні функціональні та нефункціональні вимоги до програмного забезпечення. Проведено аналіз існуючих аналогів, розглянуто їх переваги та недоліки, а також обгрунтовано актуальність розробки власного програмного продукту. Виконано проєктування архітектури системи, описано її основні компоненти, інтерфейси та взаємозв'язки між модулями. На основі обраних інструментів і технологій реалізовано програмний застосунок, що включає механізми процедурної генерації рівнів, фізичної взаємодії об'єктів та адаптації складності. Також проведено тестування розробленого програмного забезпечення, результати якого свідчать про працездатність основних функцій та можливість подальшого розвитку і вдосконалення системи.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота загалом виконана відповідно до поставленого завдання. Реалізовані функціональні можливості відповідають основним вимогам, а отримані результати підтверджують досягнення поставленої мети роботи.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання кваліфікаційної роботи. У першому розділі проведено аналіз предметної області, розглянуто існуючі аналоги та визначено функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі виконано проєктування системи, розглянуто архітектуру програмного забезпечення та обрано компонентну, обрано шаблони проектування з описом їх ролі у застосунку та описано основні компоненти, їх взаємодію, залежності та інтерфейси. У третьому розділі виконано реалізацію основних модулів застосунку, включаючи процедурну генерацію рівнів, фізичну взаємодію об'єктів, збір статистики та адаптацію складності та реалізація бази даних. Також у цьому розділі проведено тестування системи та аналіз отриманих результатів, що дозволило підтвердити працездатність реалізованих функцій у межах перевірених сценаріїв використання.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки розробка ігрових застосунків з елементами процедурної генерації та адаптивної складності є затребуваною у сфері сучасної індустрії розваг та навчальних програмних продуктів. У роботі використано сучасні підходи до проєктування програмного забезпечення, зокрема компонентну архітектуру, принципи слабкої

зв'язаності компонентів та інтерфейсну взаємодію між модулями. Позитивною стороною є також застосування сучасних технологій розробки ігрових застосунків(сучасна версія рушія Unity), що дозволило реалізувати основні функціональні можливості системи. Окремо варто відзначити використання механізмів фізичного моделювання та адаптивної зміни складності, що підвищує інтерактивність ігрового процесу та потенціал подальшого розвитку програмного продукту.

5. Негативні сторони роботи У роботі реалізовано базовий функціонал ігрового застосунку, однак деякі його частини потребують подальшого вдосконалення. Зокрема, процедурна генерація рівнів іноді створює конфігурації, які потребують додаткового балансування складності, що може впливати на стабільність ігрового досвіду. Також система адаптації складності ще недостатньо точно підлаштовується під різні стилі гри користувача, оскільки для аналізу використовується обмежений обсяг статистичних даних. Це може призводити до не завжди оптимального рівня складності. Окремі елементи ігрового інтерфейсу та взаємодії з користувачем можуть бути розширені для підвищення зручності та більш різноманітного ігрового процесу.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота в цілому виконана на достатньому рівні. Матеріал пояснювальної записки викладено послідовно, структуровано та зрозуміло, що дозволяє орієнтуватися в представленому матеріалі та логіці розробки програмного забезпечення. Графічні матеріали доповнюють текстову частину та наочно відображають основні етапи проєктування системи та її архітектуру.

8. Інші зауваження Загалом суттєвих зауважень до виконаної кваліфікаційної роботи немає. Окремі формулювання в пояснювальній записці можуть бути уточнені для підвищення точності викладу, однак це не впливає на загальну якість та цілісність роботи.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана в повному обсязі, відповідає поставленому завданню та загалом демонструє достатній рівень опрацювання теми. Розроблене програмне забезпечення є працездатним і виконує основні функціональні вимоги. Робота заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) к.і.н., доцент, доцент кафедри КІС Мічкорук Л.О

"26" 05 2026 р. (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Ігровий застосунок-головоломка з процедурною генерацією рівнів та адаптивною складністю»
Автор: Квіт Владислав Леонідович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Дьоміна Анастасія Іванівна, асистент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення взяті із звіту з переддипломної практики;

2) система фіксувала технічні особливості (наприклад, поєднання латиниці й українських індексів), а не модифікацію тексту.

Сумарний обсяг запозичень — 18%, що відповідає стандартам і не впливає на якість кваліфікаційної роботи.

Дата 10.06.2026

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Анастасія ДЬОМІНА