Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Ігровий застосунок жанру аркади на рушії Unity “Move or Die”
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2301166.01.07.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Костянтин КАШЕВАР
Підпис Ім’я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Оксана ОНИШКО
Науковий ступінь, вчене звання Підпис Ім’я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, вчене звання Підпис Ім’я, ПРІЗВИЩЕ

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

2 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

Кашевару Костянтину Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Ігровий застосунок жанру аркади на рушії Unity "Move or Die"

Керівник роботи Онишко Оксана Григорівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Строк подання студентом роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задач, проєктування програмного забезпечення, програмна реалізація та тестування застосунку

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Три креслення:

1. Діаграма варіантів використання
2. Діаграма зв'язків модулів
3. Діаграма класів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., доцент	05.05.26	25.05.26
Антиплагіат	Форкун Ю. В., доцент	05.05.26	25.05.26

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 проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 - 31.12.2025	
2 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3 Проектування програмного забезпечення	21.02 – 20.03.2026	
4 Програмна реалізація з використанням відповідних засобів розробки. 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 – 25.05.2024	
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 _____ Костянтин КАШЕВАР
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ОНИШКО
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Ігровий застосунок жанру аркади на рушії Unity “Move or Die”».

Автор роботи: Кашевар Костянтин Сергійович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 63 с., 19 рис., 4 табл., 4 дод., 31 джерело.

Графічна частина: 3 креслення ф. А3.

ІГРОВИЙ ЗАСТОСУНОК, АРКАДА, C#, UNITY, КОМП’ЮТЕРНА ГРА, ANDROID, ГЕЙМПЛЕЙ, АРХІТЕКТУРА, ПАТЕРН, РУШІЙ.

Мета кваліфікаційної роботи: розробити мобільну аркадну гри «Move or Die» для операційних систем Android та iOS з використанням ігрового рушія Unity

У кваліфікаційній роботі здійснено комплексний аналіз предметної області у сфері мобільного геймінгу, зокрема сегмента казуальних та аркадних ігор. На основі дослідження визначено ключові функціональні та нефункціональні вимоги до майбутнього програмного продукту, спроєктовано загальну архітектуру мобільного застосунку, а також детально розроблено структуру базових ігрових механік і логіку взаємодії користувача з ігровим середовищем.

Для практичної реалізації проєкту було обрано сучасний кросплатформний ігровий рушій Unity та об'єктно-орієнтовану мову програмування C#. Завдяки цим інструментам розроблено повноцінну мобільну гру «Move or Die». Практичне значення отриманих результатів полягає у створенні готового та оптимізованого розважального мобільного застосунку, який забезпечує гравцям збалансований ігровий досвід із прогресуючою складністю. Розроблена гра «Move or Die» повністю відповідає сучасним стандартам мобільного геймінгу і може ефективно задовольняти потреби широкої аудиторії користувачів у проведенні дозвілля.

25.05.26
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2301166.01.07.ПЗ	Пояснювальна записка	63		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.2301166.01.07.Е8	Схема маршрутизації	1		
			аудіосигналів			
5	А3	КвРІПЗ.2301166.01.07.Е8	Діаграма варіантів	1		
			використання для гравця			
6	А3	КвРІПЗ.2301166.01.07.Е8	Блок-схема алгоритму	1		
			генерації цільових			
			об'єктів			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2301166.01.07.ПЗ		
Виконав		Кашевар К.С.		25.05	Ігровий застосунок жанру аркади на рушії Unity "Move or Die" Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Онишко О. Г.		25.05			
Н. контр.		Форкун Ю.В.		25.05		ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л.П.		25.05			

ЗМІСТ

ВСТУП

На сьогоднішній день до підготовки майбутніх фахівців у галузі інженерії програмного забезпечення висуваються серйозні вимоги. Студенти повинні не лише добре засвоїти теоретичну базу, але й вміти застосовувати ці знання для розв'язання реальних виробничих проблем. Невід'ємною складовою професійного становлення є переддипломна практика, яка слугує містком між університетською освітою та реальною роботою в IT-сфері. Вона надає можливість здобути практичний досвід, зрозуміти внутрішні процеси створення програмних продуктів та підготуватися до виконання кваліфікаційної роботи.

Сфера мобільних розваг та відеоігор наразі переживає етап бурхливого розвитку, стаючи однією з глобальних галузей у цифровому світі. Сучасні користувачі смартфонів постійно шукають нові способи проведення дозвілля, причому значний попит мають саме казуальні та аркадні ігри.

У відповідь на цей запит, в рамках індивідуального завдання розробляється мобільна гра під назвою «Move or Die». Суть геймплею полягає в управлінні кубом на ігровій платформі за допомогою віртуального джойстика. Ключова ціль гравця – збір капсул, що генеруються на локації через задані проміжки часу. З кожною підібраною капсулою швидкість появи нових об'єктів зростає, що підвищує динамічність процесу. Накопичивши певну кількість очок, користувач відкриває доступ до наступного рівня.

Головною метою проекту є створення повноцінної аркадної мобільної гри для операційних систем iOS та Android, використовуючи можливості ігрового рушія Unity та мову програмування C#.

Для досягнення поставленої мети під час проходження практики необхідно виконати низку завдань:

– ознайомитися з організаційною структурою підприємства та правилами техніки безпеки;

− дослідити предметну область та проаналізувати наявне програмне забезпечення на ринку;

− сформувати чіткі технічні вимоги до продукту, деталізувати логіку управління, поведінку капсул та систему рівнів;

− створити початковий прототип для перевірки базових механік руху та взаємодії з об'єктами;

− програмно реалізувати всі етапи геймплею, налаштувати таймери та прогресію складності;

− додати візуальне оформлення у 2D-ізометрії та відповідний динамічний звуковий супровід;

− провести комплексне тестування, оптимізувати роботу додатка та підготувати його до фінального релізу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Вивчення предметної області проводиться з метою визначення проблем, актуальних тенденцій та ключових особливостей ринку, для якого розробляється програмний продукт. У контексті даного дипломного проектування предметною областю є індустрія відеоігор, зокрема сегмент мобільних казуальних та аркадних розваг.

Мобільні телефони вже багато років слугують для нас вірними помічниками як вдома, так і на роботі. З часу своєї появи їхні можливості значно розширилися, перетворивши звичайні засоби зв'язку на багатофункціональні пристрої – смартфони. Сучасні мобільні платформи володіють великими обчислювальними потужностями, які ще кілька десятків років тому було важко уявити в такому компактному корпусі. Завдяки тому, що смартфон є легким, компактним і завжди знаходиться під рукою, він став ідеальною платформою не лише для комунікації, але й для споживання мультимедійного контенту та цифрових розваг.

Ігрова індустрія сьогодні є однією з галузей комп'ютерних технологій, що розвиваються найшвидшими темпами, формуючи глобальний сектор розваг [1]. У період стрімкої цифровізації індустрія діджитал-розваг, і зокрема геймінг, демонструє безперервне зростання. Кількість людей, які грають у відеоігри, невпинно збільшується. Дослідження ринку показують, що однією з ключових потреб гравців є бажання розслабитися, згаяти час та відключитися від реальності [2]. Саме ці фактори зумовлюють велику популярність мобільних ігор, адже вони дозволяють задовольнити такі потреби у будь-якому місці та за будь-яких обставин.

Серед усього різноманіття жанрів відеоігор особливе місце займають казуальні та аркадні проекти [3]. Їхня привабливість полягає в інтуїтивно

зрозумілому управлінні, відсутності необхідності довготривалого навчання, можливості проводити короткі ігрові сесії та високій загальній динаміці. Цільовою аудиторією розроблюваного додатка є саме такі користувачі смартфонів, які полюбляють аркадні та казуальні ігри.

Аркада – поширений в індустрії відеоігор термін, що позначає ігри з навмисно примітивним ігровим процесом. Деякі ресурси про відеоігри виділяють їх як окремий жанр і зараховують до них платформери.

Структурні та функціональні особливості мобільних аркад вимагають від розробників створення максимально збалансованого та захоплюючого ігрового процесу (геймплею). Мобільна гра «Move or Die» є яскравим представником цього жанру. З функціональної точки зору, базовий геймплей будується навколо управління головним об'єктом – кубом, який переміщується по спеціальній платформі. Для забезпечення зручної взаємодії з грою на пристроях із сенсорними екранами, управління реалізовано за допомогою віртуального джойстика [4]. Таке рішення є стандартом для мобільного геймінгу, оскільки воно забезпечує точність рухів та швидкий відгук системи.

Основне функціональне завдання користувача в грі полягає у зборі ігрових предметів – капсул, кожна з яких приносить гравцеві визначену кількість очок. Для того, щоб ігровий процес не втрачав своєї гостроти, предметна область вимагає наявності механізмів виклику. У проекті «Move or Die» капсули з'являються на платформі через певний інтервал часу. Структурною особливістю цієї механіки є таймер, який постійно зменшується з кожною успішно зібраною капсулою. Це експоненційно збільшує динаміку гри: гравцеві доводиться приймати рішення та діяти все швидше, покращуючи свою реакцію.

Щоб уникнути монотонності та стимулювати гравця до подальшого проходження, у системі передбачено рівневу структуру. Після досягнення визначеної кількості набраних очок відбувається перехід на наступний рівень. З кожним новим етапом складність гри зростає, зокрема, за рахунок ще більшого зменшення початкового таймеру появи капсул. Окрім самого ігрового поля,

структура додатка включає зручне меню для вибору рівня, старту або виходу, а також розділ налаштувань, де користувач може змінити гучність музики, звуків та чутливість джойстика.

Важливим аспектом предметної області є візуальне та звукове сприйняття продукту. Для мобільних ігор критично важливо мати привабливий дизайн, який оптимізовано під апаратні ресурси пристроїв. Дизайн гри розробляється в ізометричному 2D стилі з використанням яскравих контрастних кольорів. Таке рішення дозволяє створити об'ємне зображення, залишаючись вимогливим лише до базових потужностей смартфона. Аудіовізуальна взаємодія підсилюється динамічними звуковими ефектами, які супроводжують збір капсул та переходи між рівнями, що робить ігровий досвід більш цілісним.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сучасному ринку мобільних додатків існує величезна кількість ігор у жанрах аркади та казуальних розваг. Проте, незважаючи на їхню популярність, багато з них швидко втрачають інтерес користувачів через одноманітність геймплею, перевантаженість інтерфейсу або незручне управління. Для більш глибокого розуміння ринку та виявлення недоліків існуючих рішень було проведено аналіз популярних ігрових додатків, які мають схожі механіки.

Розглянемо декілька відомих аналогів у цьому жанрі:

Cube Surfer! (від студії Voodoo): це популярна гіперказуальна 3D-гра, де гравець керує кубом, ковзає вперед, і повинен збирати інші куби, щоб будувати вежу та уникати перешкод. Також гравець може збирати фіолетові ромби, які являються валютою та можна витрачати на купівлю нових предметів. Як типовий представник гіперказуального жанру, гра вирізняється максимально спрощеним керуванням та орієнтована на короткі, динамічні ігрові сесії. Геймплей “Cube Surfer!” гра зображена на Рисунку 1.1.

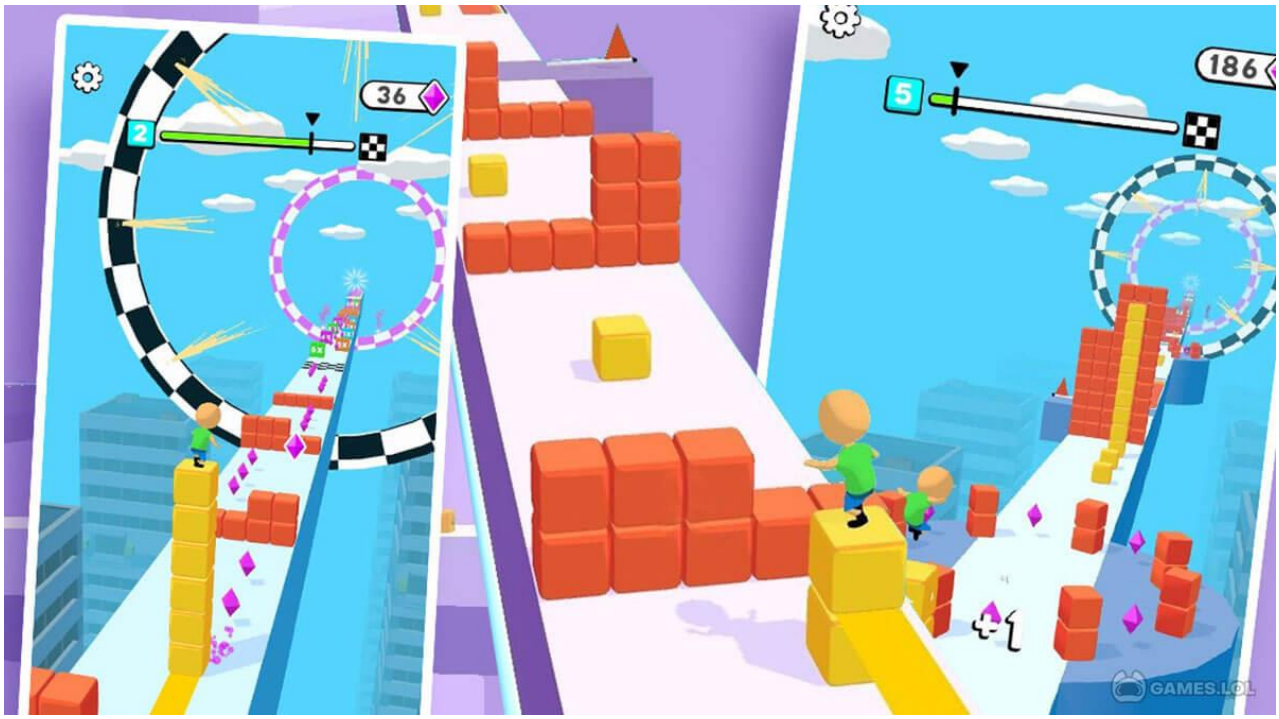


Рисунок 1.1 – Геймплей гри “Cube Surfer!”

Переваги: Максимально просте управління одним пальцем (свайпи), зрозуміла мета, швидкі ігрові сесії, барвиста графіка.

Недоліки: Величезна кількість нав'язливої реклами, яка перериває ігровий процес; монотонність геймплею через відсутність таймерів чи динамічного прискорення; відсутність повноцінного вільного переміщення (рух відбувається лише по заданій прямій траєкторії).

Pac-Man 256 (від Hipster Whale): сучасна ізометрична варіація класичної аркади. Ігровий процес полягає у безперервному пересуванні керованого об'єкта процедурно генерованим лабіринтом, зборі балів та уникненні контакту з комп'ютерно керованими супротивниками, які наділені різними алгоритмами поведінки та переслідування. Ключовою динамічною загрозою виступає візуалізований апаратний збій («глітч»), що поступово руйнує пройдені ділянки рівня знизу екрана, обмежуючи час на прийняття рішень та стимулюючи користувача до беззупинного руху. Для розширення варіативності ігрового процесу в продукт впроваджено систему тимчасових модифікаторів. Гра "Pac-Man 256" зображена на Рисунку 1.2.

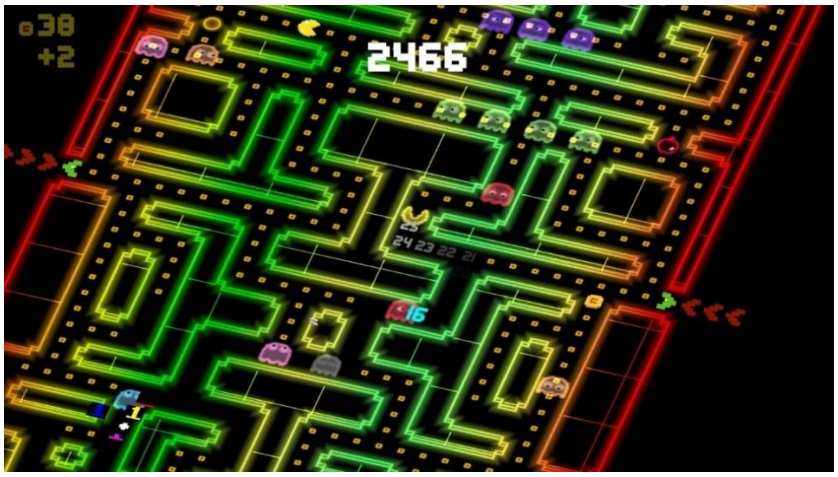


Рисунок 1.2 – Геймплей гри “Pac-Man 256”

Переваги: Якісна ізометрична графіка; наявність механіки вільного переміщення; постійне відчуття динаміки через загрозу, що насувається (аналог таймеру).

Недоліки: Досить високий поріг входження для новачків через велику кількість ворогів зі складною логікою руху; перевантажений користувацький інтерфейс; складна система бонусів, яка відходить від концепції мінімалістичної казуальної гри.

Rolling Sky (від Cheetah Mobile): динамічна аркада, де гравець керує кулею, що котиться, обходячи перешкоди під ритмічну музику. (Рисунок 1.3)

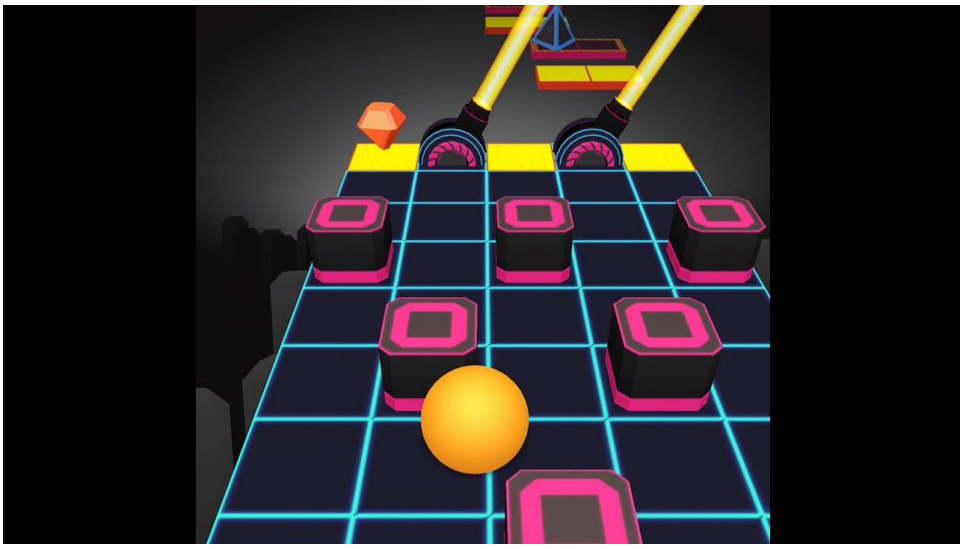


Рисунок 1.3 – Геймплей гри “Rolling Sky”

Переваги: Відмінна синхронізація звукових ефектів та музики з ігровим процесом; яскраве візуальне оформлення; чіткий поділ на рівні.

Недоліки: Гра є надзвичайно складною і вимагає постійного завчання рівнів напам'ять; управління зводиться лише до рухів вліво/вправо; відсутня механіка збору предметів на час.

Дослідження наявних на ринку програмних рішень продемонструвало присутність двох ключових проблем у цьому жанрі. З одного боку, існуючі проєкти часто невиправдано примітивізують систему керування, зводячи переміщення гравця лише до однієї осі, що суттєво обмежує ігровий досвід та знижує динаміку. Враховуючи виявлені недоліки конкурентів, створюваний мобільний застосунок "Move or Die" має нівелювати ці проблеми, знайти оптимальний баланс та забезпечити:

− повний контроль над персонажем (кубом) у просторі завдяки зручному віртуальному джойстику;

− чітку і зрозумілу мету – збір капсул, що дають очки;

− постійне підтримання інтересу гравця за рахунок таймера появи капсул, який динамічно зменшується після кожного збору, змушуючи діяти швидше;

− меню повинно бути динамічним;

− аудіо у грі повинно бути комфортним і загальноприйнятним;

− система рейтингу згідно створених нікнеймів гравцем;

− поступове зростання складності через систему рівнів;

− приємний ізометричний 2D-дизайн [9] із яскравими кольорами, який не перевантажує систему та дозволяє грі плавно працювати на пристроях середнього цінового сегмента (від Snapdragon 660 і вище).

З метою систематизації зібраних даних та глибокого аналізу ринкових аналогів було складено порівняльну таблицю наявного програмного забезпечення (Таблиця 1.1). Це дозволяє не лише візуально зіставити конкурентів за ключовими критеріями – ігровими механіками, графічною стилістикою та паттернами управління, але й виявити актуальні тенденції розробки у

відповідному сегменті, що є необхідним етапом для формування вимог до проектованої системи.

Таблиця 1.1 – Порівняльна характеристика наявного програмного забезпечення (ігор-аналогів)

Назва гри	Тип управління	Графічний стиль	Механіка наростання складності	Основні недоліки
Color Bump 3D	Свайпи вліво/вправо (автоматичний рух вперед)	Мінімалістичний 3D	Статична складність рівнів, відсутність таймерів	Обмежений контроль, монотонність геймплею
PAC-MAN 256	Свайпи у 4-х напрямках	Ізометричний воксельний 3D	Наростання швидкості переслідувачів (ворогів)	Перевантажений інтерфейс, обмежена плавність рухів
Cube Surfer	Свайпи по горизонталі	Яскравий 3D	Збільшення кількості перешкод	Лінійність рівнів, рух лише по прямій
Move or Die	Віртуальний джойстик (повний контроль)	Ізометричний 2D, яскраві контрастні кольори	Динамічне зменшення таймеру появи капсул	Знаходиться на стадії розробки

Таким чином, проведений аналіз показує, що розробка мобільної аркадної гри потребує комплексного підходу. Обрана концепція повністю відповідає сучасним вимогам ринку, а закладені механіки прогресуючої складності гарантують високу зацікавленість цільової аудиторії, забезпечуючи при цьому високий рівень реіграбельності та конкурентоспроможність майбутнього програмного продукту.

1.3 Аналіз вимог до довідково-інформаційної системи

Основні вимоги до розроблюваного ігрового мобільного додатка включають в себе наступні пункти, які, згідно аналізу, являються ключовими для створення якісного продукту:

− надання користувачу можливості зручного керування ігровим об'єктом (кубом) за допомогою віртуального джойстика;

− реалізація механіки появи (спавну) капсул на ігровій платформі через визначений таймер;

− забезпечення алгоритму динамічного зменшення таймеру після кожного успішного збору капсули для підвищення загальної ігрової динаміки;

− нарахування ігрових очок за кожну підібрану гравцем капсулу;

− розробка системи рівневої прогресії, яка дозволяє переходити на наступний етап лише після досягнення необхідної кількості балів;

− забезпечення автоматичного зростання складності на нових рівнях (шляхом прискорення таймерів появи предметів);

− створення повноцінного головного меню з опціями старту гри, вибору розблокованих рівнів та виходу з програми;

− інтеграція панелі налаштувань для регулювання гучності фонової музики, звукових ефектів та налаштування чутливості джойстика;

− реалізація функції призупинення гри через кнопку паузи.

Для архітектурного проектування та наочного представлення функціоналу будується діаграма варіантів використання для кінцевого користувача (гравця), яка описує його можливі дії в межах ігрової системи. Також формується діаграма послідовності для ключового варіанту використання, наприклад, «Збір капсули та нарахування очок». Окремо розробляється діаграма, яка відображає процес переходу між рівнями на етапах взаємодії гравця, користувацького інтерфейсу та алгоритмів ігрового рушія Unity [10].

Окрім базових функціональних та системних характеристик, глибокий аналіз вимог вимагає врахування психологічних аспектів взаємодії користувача з мобільним застосунком. Оскільки цільовою аудиторією є гравці казуального та мідкорного сегментів, які зазвичай приділяють грі від кількох хвилин до півгодини на день, архітектура вимог повинна базуватися на концепції миттєвого залучення. Це означає, що час від запуску застосунку до початку активного ігрового процесу має бути зведений до абсолютного мінімуму. Вимога щодо забезпечення так званого «холодного старту» (Cold Start) менш ніж за три секунди зумовлює необхідність відмови від важких графічних завантажувальних екранів на користь асинхронного завантаження ресурсів в оперативну пам'ять пристрою.

Не менш важливою вимогою є забезпечення високого рівня ергономіки користувацького інтерфейсу (UX/UI). Мобільні пристрої мають відносно невелику площу екрана, значну частину якої гравець перекриває власними руками під час керування. Відповідно, висувається вимога до проєктування віртуального джойстика: він повинен мати динамічну або плаваючу природу, активуючись у зоні дотику великого пальця, щоб не обмежувати огляд ігрової сцени. Розташування ключових елементів HUD (Head-Up Display), таких як лічильник очок або шкала прогресу, має відповідати патернам периферійного зору, розміщуючись у верхній частині екрана. Візуальний стиль, побудований на контрастних кольорах та ізометричній проєкції, не лише відповідає сучасним трендам, але й виконує пряму вимогу щодо візуального розділення інтерактивних об'єктів (гравця, бонусів) та фонового середовища, знижуючи когнітивне навантаження на користувача під час тривалих сесій.

Функціональна вимога щодо динамічного масштабування складності потребує окремого математичного обґрунтування. В основі ігрового циклу лежить концепція стану «потоку» (Flow), яка передбачає постійний баланс між рівнем навичок гравця та складністю викликів, які генерує система [5]. Якщо швидкість появи капсул залишатиметься статичною, гра швидко перетвориться

на рутину. З іншого боку, занадто різке прискорення таймерів призведе до неможливості проходження рівня та фрустрації. Отже, система вимагає впровадження гнучких інтерполяційних математичних функцій (наприклад, кривих Безьє або логарифмічних залежностей) для розрахунку таймерів спавну. Це гарантуватиме плавне, але невідворотне зростання напруги, стимулюючи гравця до максимальної концентрації та швидкого прийняття рішень.

З точки зору аудіального супроводу, вимоги виходять за межі простої наявності фонової музики. Звук у динамічних аркадах відіграє роль критично важливого каналу передачі зворотного зв'язку [6]. Додаток повинен мати чітку систему аудіальних маркерів: підбір звичайної капсули, активація унікального бонусу, перехід на новий рівень або зіткнення з перешкодою повинні супроводжуватися унікальними, легко впізнаваними звуковими ефектами. Це дозволяє гравцеві орієнтуватися в ігрових подіях не лише візуально, але й на слух, що значно покращує загальний рівень занурення. Відповідно, програмна архітектура повинна включати оптимізований аудіомікшер, здатний відтворювати кілька звукових доріжок одночасно без спотворення сигналу та затримок [7].

Враховуючи високу фрагментацію екосистеми Android, окремим блоком стоять вимоги до апаратної сумісності та обробки системних переривань [8]. Сучасні смартфони мають різні форм-фактори екранів, включаючи наявність вирізів під камери та закруглені кути. Застосунок вимагає використання механізмів безпечних зон при верстці інтерфейсу, щоб жоден функціональний елемент або текст не був обрізаний фізичними межами дисплея. Крім того, операційна система мобільного пристрою є багатозадачним середовищем. Вимоги до надійності зобов'язують програму коректно обробляти життєвий цикл застосунку: у разі надходження вхідного дзвінка, спрацьовування будильника або простого згортання гри (втрати фокусу додатком), ігрова сесія повинна миттєво ставитися на паузу з кешуванням поточних координат усіх об'єктів та стану

таймерів. Ігнорування цієї вимоги призвело б до втрати ігрового прогресу та різкого зниження лояльності користувачів.

Проєктування програмної архітектури також має відповідати вимозі розширюваності та підтримуваності коду. Незважаючи на те, що «Move or Die» створюється як цілісний самодостатній продукт, закладені алгоритми повинні дозволяти легке інтегрування нових ігрових механік у майбутньому без необхідності глибокого рефакторингу базового ядра. Це вимагає строгого дотримання принципів об'єктно-орієнтованого проєктування, зокрема принципу відкритості/закритості [9]. Наприклад, модуль процедурної генерації повинен бути спроєктований так, щоб додавання нового типу бонусів вимагало лише створення нового скрипту з наслідуванням базового інтерфейсу, а не переписування логіки самого генератора [10]. Така інкапсуляція даних значно зменшує кількість перехресних залежностей і спрощує тестування окремих компонентів.

Впровадження всіх перелічених вимог до мобільного додатка дозволить створити якісний розважальний продукт, що задовольнить потреби цільової аудиторії. Підвищення якості ігрового досвіду досягається за рахунок того, що гравець отримує інтуїтивно зрозуміле управління та плавно збалансовану криву складності. Завдяки чітко спроектованій архітектурі та технічній оптимізації, зменшиться кількість потенційних помилок у логіці гри, що гарантуватиме її безперебійне та плавне функціонування на смартфонах середнього рівня (від Snapdragon 660+). Крім того, такий ґрунтовний підхід до розробки забезпечить необхідну гнучкість програмної системи для її подальшого масштабування. Це дозволить у майбутньому легко інтегрувати новий ігровий контент, додаткові рівні або механіки без необхідності суттєвої переробки базового коду. У кінцевому підсумку, суворе дотримання сформованих вимог стане надійним фундаментом для успішного релізу застосунку.

Підсумувавши все проаналізоване вище, маємо Рисунок 1.4, на якому зображена діаграма варіантів використання для гравця «Move or Die».

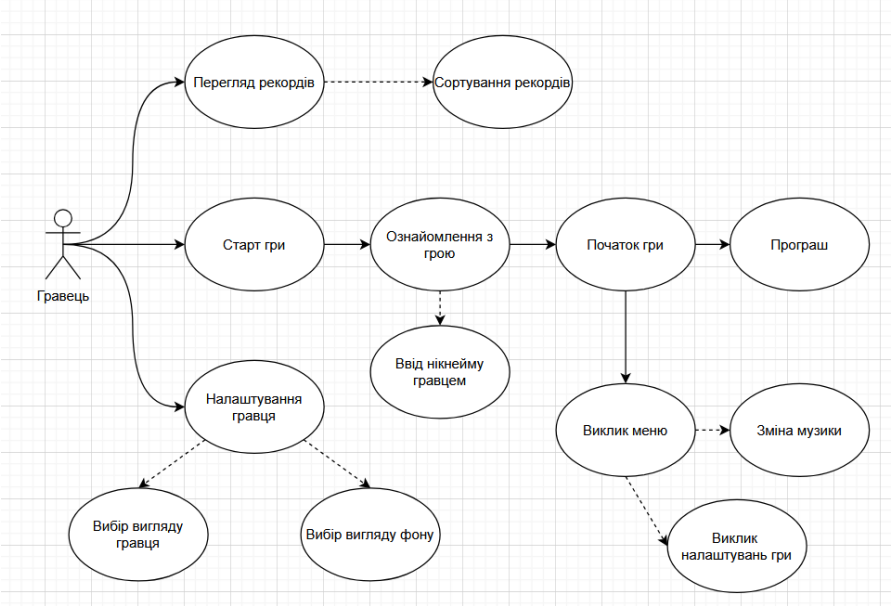


Рисунок 1.4 – Діаграма варіантів використання для гравця «Move or Die»

1.4 Висновки до першого розділу. Постановка задачі

У цьому розділі було досліджено предметну область розробки мобільних ігрових застосунків, зокрема в сегменті казуальних та аркадних ігор. Проаналізовано ключові етапи створення мобільної гри, їхнє призначення та основну мету розробки проєкту «Move or Die». Для чіткого визначення вимог до застосунку було розглянуто жанрові особливості аркад, специфіку цільової аудиторії та її очікування від ігрового процесу (зокрема, потребу в динаміці та зручності проведення коротких ігрових сесій).

З метою виокремлення найбільш вдалих та захоплюючих ігрових механік було проведено огляд популярних аналогів на ринку мобільних додатків (для платформ Android та iOS). Повний аналіз предметної області дав змогу сформувати чіткі функціональні та нефункціональні вимоги до майбутньої гри, включаючи критерії до оптимізації графіки, управління та балансу складності. Ці дані підсумовуються у вигляді технічного завдання, яке остаточно окреслює межі проєкту та значно структурує подальші етапи розробки.

Отримавши вичерпну функціональну і технічну специфікацію під час роботи над розділом, можна поставити конкретні задачі, які мають бути реалізовані в подальшому під час виконання кваліфікаційної роботи.

Перелік поставлених задач:

− провести аналіз доступних архітектурних рішень та патернів проєктування, обравши найбільш оптимальні для реалізації логіки аркадної гри;

− описати структурну декомпозицію проєкту, взаємодію між основними класами та модулями;

− розробити макет користувацького інтерфейсу (головне меню, екран ігрового процесу, вікно паузи);

− обґрунтувати вибір технологічного стеку (ігровий рушій Unity та мова C#) для виконання поставлених вимог;

− запрограмувати ключові ігрові механіки: рух об'єкта за допомогою віртуального джойстика, систему генерації капсул та алгоритми динамічного таймера;

− інтегрувати базові візуальні ресурси у стилі 2D-ізометрії та налаштувати необхідні графічні ефекти;

− підібрати та імплементувати відповідний звуковий супровід, синхронізувавши його з ігровими подіями;

− спроєктувати та запрограмувати інтерактивний інтерфейс користувача;

− визначити стратегію тестування, провести перевірку продуктивності та коректності роботи на мобільних пристроях;

− проаналізувати отримані результати тестування, усунути виявлені дефекти та сформувати загальні висновки щодо виконаної роботи.

2 ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

2.1 Вибір типу архітектури та шаблонів проєктування

Проєктування ігрового застосунку «Move or Die» розпочинається з комплексного аналізу архітектурних підходів та вибору інструментальних засобів, оскільки ці рішення визначають структуру системи, ефективність розробки та стабільність роботи продукту на мобільних пристроях. Правильний вибір технологічного стеку дозволяє реалізувати закладені функціональні вимоги з максимальною продуктивністю та мінімальними витратами ресурсів.

Для розроблення ядра ігрового застосунку було розглянуто декілька фундаментальних типів архітектур. Монолітна архітектура, де всі компоненти системи інтегровані в єдиний тісно пов'язаний код, була відхилена через низьку масштабованість та складність підтримки при розширенні проєкту. Клієнт-серверний підхід визначено як надмірний, оскільки застосунок не передбачає мережевої взаємодії. Архітектура ECS (Entity Component System), попри високу ефективність у симуляціях з великою кількістю однотипних об'єктів, також визнана надлишковою для аркадної гри з помірною кількістю сутностей на сцені.

На основі аналізу специфіки проєкту «Move or Die» за базову обрано компонентно-орієнтовану архітектуру, яка є нативною для обраного ігрового рушія Unity [11]. Вона дозволяє розділити логіку на незалежні компоненти (наприклад, Movement, Collision, Spawner), що прикріплюються до ігрових об'єктів. Для забезпечення низької зв’язності модулів цей підхід доповнено подійно-орієнтованою парадигмою. Така змішана архітектура дозволяє реалізувати взаємодію об'єктів через систему подій, де, наприклад, підбір капсули активує незалежні реакції в системах підрахунку очок, звуку та візуальних ефектів без прямого звернення до їхніх методів.

Реалізація обраної архітектури здійснюється за допомогою наступного технологічного стеку, який було відібрано для зручності розробки мобільного застосунку:

− ігровий рушій Unity: обраний через потужну підтримку мобільних платформ Android та iOS, наявність вбудованої фізичної системи PhysX та повну відповідність компонентно-орієнтованій моделі розробки;

− мова програмування C#: забезпечує строгу типізацію, зручну роботу з подіями та делегатами, а також ефективне управління пам'яттю, що є критичним для стабільної частоти кадрів (FPS) на мобільних пристроях [12];

− середовище розробки JetBrains Rider: визначено як основний інструмент через глибоку інтеграцію з Unity API, що серйозно пришвидшує розробку та потужні засоби статичного аналізу коду, що пришвидшує відлагодження алгоритмів;

− система контролю версій Git: застосовується для управління змінами вихідного коду та забезпечення цілісності проєкту під час ітеративної розробки.

На Рисунку 2.1 зображена загальна архітектура системи ігрового застосунку.

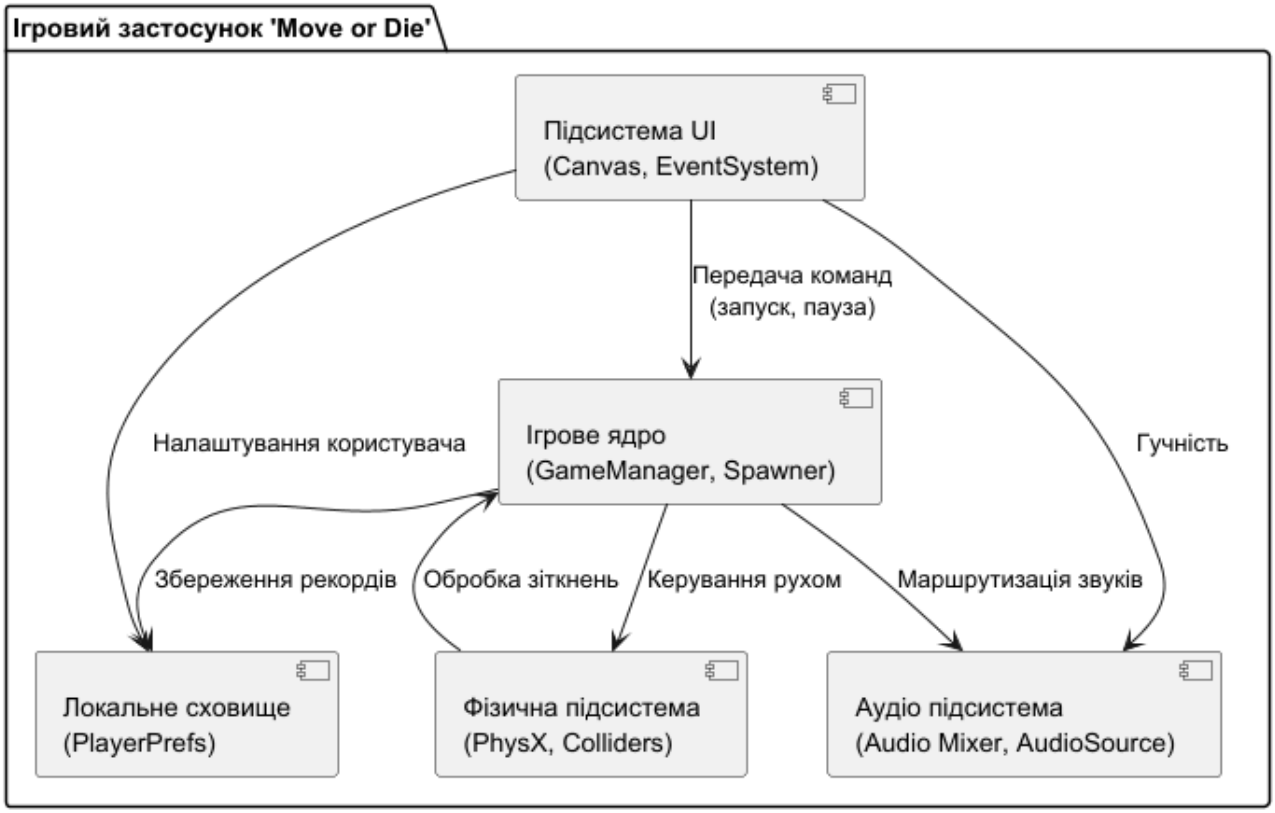


Рисунок 2.1 – Загальна архітектура системи

Для оптимізації роботи застосунку в межах обраної архітектури визначено застосування спеціалізованих шаблонів проєктування. Патерн «Одинак» (Singleton) використовується для централізованого управління станами гри через GameManager. Для реалізації динамічної генерації об'єктів без перевантаження процесора операціями виділення пам'яті обрано метод «Об'єктний пул» (Object Pool), який дозволяє циклічно використовувати деактивовані капсули замість їхнього постійного створення та знищення [13]. Взаємодія між ігровою логікою та інтерфейсом базується на шаблоні «Спостерігач» (Observer).

Таким чином, поєднання компонентно- та подійно-орієнтованої архітектури з використанням рушія Unity, мови C# та оптимізаційних патернів проєктування дозволяє створити гнучку, масштабовану та високопродуктивну систему, що повністю відповідає вимогам до сучасних мобільних аркадних ігор.

2.2 Опис декомпозиції

2.2.1 Загальна декомпозиція

Після обґрунтування компонентно-орієнтованого підходу наступним логічним етапом проєктування є декомпозиція архітектури ігрового застосунку. У контексті програмної інженерії декомпозиція – це процес поділу складної монолітної системи на менші, простіші та структурно незалежні елементи (підсистеми), які взаємодіють між собою за чітко визначеними правилами.

Для мобільних ігрових застосунків, які функціонують в умовах обмежених апаратних ресурсів (лімітований обсяг оперативної пам'яті, теплові обмеження процесора та енергоспоживання), правильна декомпозиція є критично важливою. Вона не лише забезпечує гнучкість розробки та полегшує процес ізольованого тестування (Unit Testing), але й дозволяє оптимізувати навантаження на систему, оскільки незалежні модулі можуть завантажуватися та вивантажуватися з пам'яті за потреби, не блокуючи основний ігровий потік.

В основу процесу декомпозиції застосунку «Move or Die» покладено метод проєктування «зверху-вниз» (Top-Down Design) та фундаментальні принципи SOLID, зокрема принцип єдиної відповідальності (Single Responsibility Principle) та принцип розділення інтересів (Separation of Concerns) [14]. Це означає, що кожна виділена підсистема повинна вирішувати виключно одну концептуальну задачу і мати мінімальну залежність від внутрішньої реалізації інших підсистем (Low Coupling), зберігаючи при цьому високу внутрішню цілісність (High Cohesion).

На основі проведеного аналізу предметної області та сформованих функціональних вимог, загальну архітектуру застосунку було декомпозиовано на шість ключових високорівневих підсистем:

− Система глобального управління (Core Management System): виступає центральним ядром архітектури та виконує роль головного контролера (Game Flow Controller). Її головне завдання – ініціалізація ігрової сесії, керування життєвим циклом застосунку та управління глобальними станами (завантаження середовища, перехід у головне меню, активація ігрового процесу, виклик паузи, обробка завершення рівня). Ця підсистема діє як фасад: вона координує запуск інших підсистем, але свідомо не містить у собі жодної ігрової логіки чи фізичних розрахунків. Крім того, вона інкапсулює роботу з платформозалежними API (наприклад, звернення до системного реєстру пристрою для збереження даних), ізолюючи ці процеси від ігрового ядра.

− Система гравця та фізичної взаємодії (Player & Physics System): підсистема повністю інкапсулює логіку, пов'язану з ігровим аватаром (персонажем). Оскільки фізичні розрахунки в рушії Unity виконуються в окремому циклі (FixedUpdate), відокремленому від циклу рендерингу графіки (Update), виділення цієї логіки в окрему підсистему забезпечує незалежність руху гравця від частоти кадрів (Frame Rate Independence). До відповідальності підсистеми входить:

1) Зчитування користувацького введення (векторів від сенсорного віртуального джойстика) та трансляція їх у фізичні імпульси.

2) Відстеження кінематичного стану об'єкта.

3) Матриця виявлення зіткнень (Collision Detection), яка ідентифікує тип об'єкта, з яким відбувся контакт (стіна платформи, цільова капсула, перешкода чи унікальний бонус), і генерує відповідні системні переривання.

− Система генерації та просторового оточення (Environment & Spawning System): відповідає за динамічне та процедурне наповнення ігрової сцени об'єктами. Відокремлення цієї системи продиктовано необхідністю жорсткого контролю за виділенням оперативної пам'яті (Memory Allocation). Центральним елементом є алгоритм процедурної генерації (Spawner), який керує появою цільових об'єктів з урахуванням просторової обізнаності (унеможливлення появи об'єктів у координатах, де вже знаходиться гравець або інший об'єкт). Підсистема керує пулом об'єктів (Object Pool), розділяючи правила появи об'єктів (де і коли вони з'являються) та життєвий цикл самих об'єктів (їхня активація та деактивація без руйнування).

− Система унікальних бонусів та модифікаторів (Bonus & Modifier System): розроблена з урахуванням принципу відкритості/закритості (Open/Closed Principle). Базова логіка гри не повинна змінюватися при додаванні нових типів бонусів. Ця підсистема ізолює сукупність модулів, що тимчасово змінюють фундаментальні правила гри (магнетизм, уповільнення часу, множники очок, зміна розмірів). Кожен бонус функціонує як незалежний модифікатор зі своїм внутрішнім тайм-менеджментом. Підсистема відповідає за активацію ефекту через втручання у параметри інших систем та гарантоване безпечне скасування ефекту з поверненням системи до її нормального та стабільного стану.

− Аудіовізуальна підсистема (Audio-Visual Presentation System): ця підсистема реалізує принцип відокремлення логіки від представлення (Logic/View Separation). Візуальні ефекти (шейдери, зміна матеріалів, системи

частинок) та аудіо супровід (фонова музика, звукові ефекти колізій) розглядаються як "реакції" на зміну ігрового стану. Виділення їх в окрему сутність гарантує, що у разі відключення графіки чи звуку, математична та фізична моделі гри продовжать функціонувати без помилок. Підсистема також керує аудіомікшером для згладжування звукових переходів та рандомізації треків, щоб уникнути монотонності ігрового процесу.

− Система інтерфейсу користувача та прогресу (UI & Progression System) Спроєктована на базі концепції MVC (Model-View-Controller). Підсистема прогресу (Model) накопичує зібрані бали, розраховує пороги для переходу на нові рівні складності та веде статистику. Підсистема інтерфейсу (View) відповідає за адаптивне відображення цієї інформації (HUD, шкали прогресу, спливаючі повідомлення, навігаційні меню) на екранах із різним співвідношенням сторін. Відокремлення розрахунків прогресу від їхнього малювання на екрані дозволяє оптимізувати процес рендерингу (UI Batching) і запобігає зниженню частоти кадрів під час оновлення текстової інформації.

Для забезпечення цілісності роботи декомпозиованого застосунку використовується архітектура обміну повідомленнями (Event-Driven Architecture). Замість того, щоб системи зверталися одна до одної напряму (що створює жорстку залежність), вони використовують подійну модель (Observer/Event Bus) [15]. Наприклад, коли Система фізичної взаємодії фіксує підбір капсули, вона не викликає метод нарахування очок або програвання звуку безпосередньо. Вона лише генерує глобальну подію OnCapsuleCollected. Система прогресу перехоплює цю подію і збільшує лічильник балів, а Аудіовізуальна підсистема перехоплює її ж для відтворення звукового ефекту. Такий підхід забезпечує максимальну гнучкість, надійність та розширюваність ігрової архітектури проєкту. Завдяки цьому інтеграція нових модулів, наприклад, системи досягнень чи збору ігрової аналітики, не вимагатиме внесення змін до вже наявного та відлагодженого коду.

2.2.2 Модульна декомпозиція

На основі виділених високорівневих системних сутностей здійснюється детальна модульна декомпозиція. Цей етап проєктування відображає фізичний поділ архітектури проєкту на окремі програмні компоненти – функціональні скрипти мовою C# [16]. Головною метою цього поділу є забезпечення чистоти програмного коду та суворе дотримання принципу єдиної відповідальності (Single Responsibility Principle). Кожен розроблений модуль інкапсулює виключно один конкретний алгоритм або вузький набір пов'язаних функцій, що мінімізує міжмодульну зв'язність (Low Coupling). Такий підхід значно спрощує процеси подальшого налагодження та незалежного тестування кожної окремої підсистеми застосунку. У результаті загальна архітектура стає стійкою до каскадних помилок, дозволяючи безпечно модифікувати та розширювати код без ризику порушити стабільність вже відлагоджених механік.

У контексті ігрового рушія Unity модульна декомпозиція також тісно пов'язана з концепцією префабів (Prefabs). Поділ логіки на дрібні скрипти дозволяє гнучко комбінувати їх на різних ігрових об'єктах як компоненти, створюючи складну поведінку без дублювання коду (принцип DRY – Don't Repeat Yourself). Крім того, такий гранулярний підхід мінімізує ризик виникнення конфліктів при злитті гілок (merge conflicts) у системі контролю версій Git, оскільки розробка різних механік ведеться в ізольованих файлах. Це також створює надійний фундамент для подальшого масштабування проєкту та зручного впровадження нових ігрових механік. Зокрема, додавання нових типів перешкод або бонусів вимагатиме лише розробки окремих скриптів-компонентів, які легко інтегруються в існуючу систему без втручання в базову логіку гри. Структуру програмних модулів ігрового застосунку «Move or Die» класифіковано та розподілено на наступні логічні групи:

− керуючі модулі (Control Modules): ці компоненти утворюють ядро життєвого циклу гри, відповідаючи за координацію глобальних процесів, обробку станів та маршрутизацію даних між іншими підсистемами;

1) GameManager – центральний керуючий скрипт, що ініціалізує ігрову сесію, відстежує стан загального прогресу та виступає головним диспетчером ігрових подій. Реалізований за допомогою архітектурного шаблону «Одинак» (Singleton) з використанням методу DontDestroyOnLoad, що забезпечує безпечний глобальний доступ до параметрів сесії та збереження стану менеджера при перезавантаженні сцен;

2) GamePausing та GameRestart – спеціалізовані модулі для маніпуляції фізичним часом ігрового рушія (властивість Time.timeScale) та ініціалізації коректного перезавантаження поточної сцени з попереднім очищенням оперативної пам'яті і скиданням локальних таймерів;

3) GameSettings – компонент серіалізації даних. Забезпечує інтерфейс між налаштуваннями користувача та локальним сховищем даних пристрою (PlayerPrefs). Відповідає за зчитування, кешування в оперативній пам'яті та застосування конфігурацій (наприклад, рівнів гучності) під час холодного старту застосунку, мінімізуючи кількість звернень до постійної пам'яті;

4) StartAfterGuide – контролер логічної послідовності, який інтегрується в етап навчання. Забезпечує блокування ігрових механік та алгоритмів генерації до моменту успішного завершення гравцем обов'язкового туторіалу, виступаючи своєрідним "запобіжником" раннього старту;

5) GameLeaveConfirmation – модуль обробки апаратних переривань ОС Android. Перехоплює системну подію натискання кнопки «Назад» (Escape) та ініціює виклик діалогового вікна підтвердження виходу, призупиняючи при цьому ігрові процеси.

− модулі ігрової механіки та гравця (Gameplay Modules): група скриптів, що обробляє кінематику та фізичну поведінку об'єктів у тривимірному просторі

ігрової сцени. Усі розрахунки в цих модулях оптимізовані для виконання у фіксованому циклі FixedUpdate:

1) AddMovement – абстракція підсистеми введення. Транслює вектори від віртуального сенсорного джойстика у фізичні імпульси, які прикладаються до компонента Rigidbody ігрового персонажа. Параметри швидкості та прискорення винесені у публічні змінні для можливості їхньої динамічної модифікації зовнішніми ефектами;

2) PlayerCollision – інтелектуальний обробник фізичних контактів. Використовує побітові маски шарів (LayerMask) та систему ігрових тегів (Tags) для швидкої просторової фільтрації зіткнень. Диференціює контакти з цільовими предметами, унікальними бонусами чи смертельними межами платформи, генеруючи відповідні переривання;

3) GetCharacterPosition – інкапсульований провайдер просторових даних. Знижує навантаження на процесор, кешуючи посилання на компонент Transform гравця та передаючи його актуальні світові координати іншим підсистемам без необхідності постійного пошуку об'єкта на сцені;

4) CapsuleSpawner – модуль процедурної генерації. Містить математичну логіку векторного розрахунку випадкових точок появи об'єктів на площині. Для запобігання надмірному виділенню пам'яті (Memory Allocation) та активації збирача сміття (Garbage Collector), модуль керує пулом об'єктів (Object Pool), активуючи та деактивуючи префаби замість їхнього створення і знищення;

5) PlatformManipulation – скрипт керування параметрами ігрового середовища, який відповідає за динамічну модифікацію ігрової платформи (наприклад, зміну її фізичних властивостей, розміру або кута нахилу) у відповідь на ігрові події.

− модулі унікальних бонусів (Unique Capsule Bonuses): з метою дотримання принципу відкритості/закритості (Open/Closed Principle) та використання переваг поліморфізму, кожна бонусна механіка винесена в окремий

модуль [17]. Це дозволяє взаємодіяти з ними через загальний інтерфейс виклику, не перевантажуючи обробник колізій перевірками конкретних типів:

1) UniqueCapsuleBonusBiggersize – модуль модифікації масштабів (змінює параметри Transform.localScale об'єктів з використанням плавної інтерполяції);

2) UniqueCapsuleBonusMagnetism – реалізація алгоритму векторного притягування (Vector3.MoveTowards). Змінює траєкторію руху всіх активних цільових об'єктів до поточних координат гравця, ігноруючи їхню стандартну кінематику;

3) UniqueCapsuleBonusSlowedtime – маніпулятор часовим континуумом. Штучно зменшує глобальний показник Time.timeScale для уповільнення руху об'єктів, водночас компенсуючи фізичні сили гравця для збереження його початкової швидкості;

4) UniqueCapsuleBonusMultiplier – логіка застосування тимчасового математичного множника до алгоритмів нарахування балів у GameManager;

5) UniqueCapsuleBonusInstalevelup та UniqueCapsuleBonusInstapoints – модулі миттєвого втручання в систему прогресу для обходу стандартних алгоритмів накопичення.

− модулі інтерфейсу та прогресу (UI Modules): Утворюють рівень представлення (Presentation Layer). Для оптимізації рендерингу на мобільних GPU, елементи розділені на динамічні та статичні полотна (Canvas), що запобігає перемальовуванню всього екрана при зміні одного числа [18]:

1) StartMenu – контролер навігації, що керує переходами між панелями (Panels) за допомогою компонента CanvasGroup (маніпуляція прозорістю alpha та блокуванням interactable);

2) ProgressBarController – модуль візуалізації шкали досвіду, що використовує математичну функцію Mathf.Lerp для плавного анімованого заповнення індикатора (компонент Image у режимі Fill);

3) LeaderboardTable – модуль формування рейтингу, який десеріалізує масиви збережених рекордів та динамічно інстанціює рядки таблиці за готовим префабом;

4) ToastManager – оптимізована система короткочасних сповіщень. Функціонує за принципом черги (Queue FIFO) з використанням власного пулу текстових об'єктів, що гарантує відсутність пропусків повідомлень при одночасній активації кількох подій;

− аудіовізуальні модулі (Audio & Visuals): відповідають за обробку, синхронізацію та відтворення всього графічного і звукового контенту в додатку, їх задача полягає в забезпеченні безперебійного рендерингу та оптимізованого використання ресурсів. Зокрема, вони керують системами частинок (Particle Systems), анімаціями користувацького інтерфейсу та динамічним мікшуванням звукових ефектів під час взаємодії персонажа з ігровим світом. Завдяки незалежності цих модулів від основної математичної логіки гри гарантується миттєвий аудіовізуальний зворотний зв'язок без негативного впливу на стабільність частоти кадрів (FPS):

1) MusicRandomizer – алгоритм псевдовипадкового вибору композицій із завантаженого масиву AudioClip. Гарантує уникнення послідовного відтворення одного й того ж треку;

2) SliderMusicChange – інтерфейсний адаптер. Конвертує лінійні значення UI-повзунків (від 0.0001 до 1) у логарифмічну шкалу децибел (від -80dB до 0dB) для коректної математичної передачі параметрів до груп Audio Mixer;

3) ChangeMaterialWhenLevelUp – скрипт візуальної сигналізації. Здійснює ефективну гарячу заміну матеріалів (Materials) у компонентах MeshRenderer без необхідності перерахунку освітлення сцени;

4) RainbowShaderForCapsule – програмований контролер шейдера. Використовує математичні функції синуса для динамічної зміни параметрів кольору (RGB-каналів) матеріалу в часі, забезпечуючи ефект світіння (Emission) без значного навантаження на процесор.

2.2.3 Декомпозиція моделі переходів станів

Декомпозиція моделі переходів станів використовується для чіткого визначення життєвого циклу окремих об'єктів та застосунку в цілому [19]. Оскільки ігровий процес розбито на велику кількість спеціалізованих скриптів, контроль їхніх станів є критично важливим.

Головний керівний об'єкт (GameManager) має наступні чітко визначені стани:

- ініціалізація (Initialization): завантаження збережених параметрів через GameSettings та підготовка сцени.
- навчання (Guide): активація туторіалу; генерація об'єктів призупинена, працює модуль StartAfterGuide.
- активна гра (Playing): повна взаємодія з модулями AddMovement та CapsuleSpawner.
- пауза (Paused): спрацьовує переривання від GamePausing; ігровий час зупиняється (TimeScale = 0).
- завершення/Рестарт (GameOver/Restarting): активація модулів LeaderboardTable для запису результату та GameRestart.

Об'єкт генератора капсул (CapsuleSpawner) також функціонує на основі станів:

- очікування (Idle): Генерація не відбувається (наприклад, під час паузи).
- стандартна генерація (Normal Spawning): Створення звичайних об'єктів за таймером.
- бонусна генерація (Bonus Spawning): Вкидання у пул унікальних капсул із папки UniqueCapsuleBonuses з певною ймовірністю.

Окрему модель станів мають самі унікальні бонуси (наприклад, UniqueCapsuleBonusMagnetism чи UniqueCapsuleBonusSlowedtime):

- доступний (Available): Бонус розміщено на сцені, застосовано ефект RainbowShaderForCapsule.

− активований (Active): Відбулося зіткнення (PlayerCollision), таймер дії бонусу розпочав відлік.

− завершений (Depleted): Ефект вимикається, застосовуються стандартні правила сцени.

Структурування проєкту через подібну детальну модульну декомпозицію та ідентифікацію станів дозволяє створити оптимізовану архітектуру, де кожний скрипт виконує єдину відповідальність (Single Responsibility Principle), що є стандартом при роботі з сучасними ігровими рушіями.

2.3 Опис залежностей

Опис залежностей є важливим етапом при проєктуванні системи, оскільки він допомагає визначити взаємодію між компонентами (модулями) та напрямки потоків даних, що забезпечує керованість та розширюваність архітектури. Для ігрового застосунку «Move or Die» міжмодульні залежності формуються на основі ієрархії керування об'єктами сцени та обробки ігрових подій.

На основі проведеної модульної декомпозиції визначено наступні зв'язки між програмними компонентами:

− модуль GameManager виступає центральним ядром і має залежності від модулів GameSettings (для отримання конфігурацій до початку сесії), GamePausing та GameRestart (для управління станом ігрового часу та перезапуску). Також він безпосередньо звертається до LeaderboardTable для запису фінальних результатів та до StartAfterGuide для ініціалізації основного геймплею.

− модуль CapsuleSpawner залежить від GetCharacterPosition. Ця залежність є критичною, оскільки алгоритм генерації повинен розраховувати координати появи нових капсул відносно поточного розташування гравця, щоб уникати генерації об'єктів безпосередньо в його координатах. Також генератор

має структурну залежність від усіх скриптів з папки UniqueCapsuleBonuses (наприклад, UniqueCapsuleBonusMagnetism, UniqueCapsuleBonusMultiplier), оскільки керує їхньою появою на ігровій платформі.

− модуль PlayerCollision має пряму залежність від GameManager (передає сигнал про збір звичайної капсули або зіткнення з перешкодою, що призводить до програшу). Крім того, він взаємодіє з модулями унікальних бонусів, викликаючи їхню активацію при торканні.

− модуль AddMovement працює відносно автономно для обробки фізики, проте його параметри (наприклад, швидкість) можуть тимчасово модифікуватися зовнішніми модулями, зокрема скриптом UniqueCapsuleBonusSlowedtime.

− модулі візуалізації ProgressBarController та ToastManager залежать від GameManager та модулів бонусів. Вони виконують роль слухачів (спостерігачів), отримуючи дані про зміну рахунку або активацію ефектів для їхнього відображення на екрані.

− модуль SliderMusicChange взаємодіє з MusicRandomizer для коректного перемикання аудіотреків через інтерфейс налаштувань.

− модуль ChangeMaterialWhenLevelUp залежить від GameManager (чекає на сигнал про підвищення рівня) та взаємодіє з компонентами рендерингу платформи (PlatformManipulation).

Опис цих зв'язків дозволяє уникнути циклічних залежностей (коли модулі безкінечно посилаються один на одного) та оптимізувати процес подальшого детального проєктування класів. Більше того, чітке розмежування зон відповідальності сприяє дотриманню принципу слабкої зв'язності (low coupling) та високого зчеплення (high cohesion). Завдяки цьому значно спрощується процес модульного тестування, оскільки незалежні компоненти легше ізолювати та перевіряти на наявність помилок, що в кінцевому підсумку підвищує загальну стабільність програмного продукту. Зрештою, такий підхід закладає міцний фундамент для системи, дозволяючи безперешкодно впроваджувати функції або модифікувати наявні без ризику порушити працездатність суміжних модулів.

2.4 Опис інтерфейсу модулів

Для розроблених модулів гри «Move or Die» визначено набір публічних методів (інтерфейсів), які дозволяють компонентам взаємодіяти між собою. Загальне визначення ключових модулів та їхніх інтерфейсів наведено у таблиці (Таблиця 2.1).

Таблиця 2.1 – Основні модулі застосунку та їхні інтерфейси

Модуль	Інтерфейс
GameManager	StartGameSession(), EndGame(), AddScore(int amount), TriggerLevelUp(), GetCurrentScore()
GamePausing	PauseGame(), ResumeGame(), IsGamePaused()
AddMovement	SetMovementSpeed(float newSpeed), GetMovementSpeed(), ApplyImpulse(Vector3 direction)
PlayerCollision	OnCollisionEnter(Collision collision), OnTriggerEnter(Collider other)
CapsuleSpawner	StartSpawning(), StopSpawning(), SpawnStandardCapsule(), SpawnBonusCapsule(BonusType type)
ProgressBarController	UpdateProgress(float currentPoints, float requiredPoints)
LeaderboardTable	SaveResult(string playerName, int score), DisplayLeaderboard()
MusicRandomizer	PlayRandomTrack(), StopMusic(), SetVolume(float volumeLevel)

Наявність чітко визначених публічних методів дозволяє, наприклад, модулю PlayerCollision при зіткненні з капсулою просто викликати метод AddScore(1) у GameManager, не втручаючись у внутрішні алгоритми підрахунку балів чи перевірки умов підвищення рівня. Це забезпечує високу стабільність та надійність архітектури ігрового застосунку.

2.5 Проєктування інтерфейсу користувача

На етапі проєктування інтерфейсу користувача (UI) визначаються функції, які має підтримувати графічна оболонка застосунку, розробляється структура екранів та створюються відповідні макети (wireframes) для візуалізації взаємодії. Оскільки гра «Move or Die» орієнтована на мобільні пристрої, ключовим завданням є забезпечення ергономіки сенсорного управління (Touch UI) та оптимального розміщення елементів, що не перекриватимуть огляд ігрової платформи [20].

Аналіз функціональних вимог до застосунку дозволяє виділити дві основні групи графічних екранів:

− навігаційні меню: екрани, що забезпечують доступ до загального функціоналу поза ігровим процесом (запуск, налаштування, статистика);

− HUD (Head-Up Display): елементи, що накладаються поверх ігрової сцени під час проходження рівня та надають оперативну інформацію або інструменти керування;

Навігаційна система меню складатиметься з наступних екранів та вікон:

− головне меню (модуль StartMenu): містить кнопки для запуску ігрового процесу, переходу до налаштувань, перегляду рейтингу гравців та виходу з гри. Вихід супроводжується вікном підтвердження (GameLeaveConfirmation) для уникнення закриття застосунку;

− меню налаштувань (GameSettings): вікно з повзунками для регулювання гучності музики та звукових ефектів (з використанням логіки SliderMusicChange);

− таблиця лідерів (LeaderboardTable): екран, що відображає найкращі результати (набрані очки) поточного користувача або глобального рейтингу.

Ігровий інтерфейс (HUD) включатиме такі динамічні елементи:

− шкала прогресу (ProgressBarController): візуальний індикатор, розміщений на екрані, який відображає поточну кількість набраних очок та прогрес до досягнення наступного рівня складності;

− спливаючі сповіщення (ToastManager): система короткочасних текстових або графічних повідомлень. Вони з'являтимуться на екрані при підборі унікальних бонусів (наприклад, повідомлення «Магнетизм активовано» або «Уповільнення часу»), інформуючи гравця про зміну правил без зупинки ігрового процесу;

− сенсорний джойстик (Virtual Joystick): невидима або напівпрозора область у нижній частині екрана для керування рухом куба;

− меню паузи (GamePausing): напівпрозоре вікно, що викликається відповідною кнопкою в куті екрана. Містить опції продовження гри, швидкого перезапуску рівня (GameRestart) та виходу до головного меню.

Для організації цих елементів розробляються схематичні макети. У мобільному геймінгу елементи активного керування (наприклад, зони джойстика) традиційно розміщуються у нижніх кутах екрана, в зоні досяжності великих пальців гравця для комфортної взаємодії.

Інформаційні панелі (панель прогресу, лічильник рівня) розташовуються у верхній частині екрана. Сповіщення (ToastManager) доцільно виводити по центру у верхній третині екрана, щоб вони були помітними, але не блокували видимість цільових капсул на ігровій сцені.

Створення прототипів та макетів інтерфейсу дозволяє наочно проаналізувати взаємне розташування компонентів, провести дослідження

(перевірку зручності) та гарантувати, що обрані дизайн-рішення є ергономічними і дозволяють гравцям ефективно орієнтуватися у грі.

2.6 Детальне проєктування модулів

Визначення функціонального призначення основних модулів та опис їхніх інтерфейсів було проведено на попередніх етапах. Наступним кроком є детальне проєктування, яке передбачає уточнення модулів як об'єктів об'єктно-орієнтованого програмування та повний опис їхніх внутрішніх зв'язків. Це дає можливість розкрити архітектуру проєкту для подальшої програмної реалізації.

Найвищим у логічній ієрархії гри «Move or Die» виступає об'єкт GameManager. Цей керуючий клас асоціюється з підсистемами інтерфейсу (StartMenu, ProgressBarController, ToastManager) шляхом агрегації, оскільки керує їхнім відображенням, але вони можуть існувати як незалежні графічні об'єкти на сцені. Об'єкт ігрового персонажа (куб) композиційно містить у собі модулі AddMovement та PlayerCollision. Модуль CapsuleSpawner пов'язаний асоціативним зв'язком «один до багатьох» із пулом базових капсул та модулями унікальних бонусів (наприклад, UniqueCapsuleBonusBiggersize, UniqueCapsuleBonusSlowedtime). На основі цієї деталізації для наочності розробляється діаграма зв'язків об'єктів, яка допомагає уникнути жорсткої залежності (tight coupling) між скриптами.

Для опису динамічної взаємодії об'єктів у часі найефективніше використовувати UML-діаграми послідовності (Sequence Diagrams). Вони детально відображають, якими повідомленнями обмінюються класи під час виконання конкретного ігрового сценарію та яким чином відбувається передача інформації від одного класу до іншого.

Першим ключовим сценарієм є процес ініціалізації ігрового рівня. Під час цього сценарію GameManager звертається до GameSettings для завантаження

параметрів, після чого передає управління модулю StartAfterGuide. Коли гравець завершує етап навчання, StartAfterGuide відправляє подію до GameManager, який, у свою чергу, активує компонент CapsuleSpawner та дозволяє модулю AddMovement зчитувати введення користувача.

Другим критично важливим сценарієм є підбір унікального бонусу (наприклад, уповільнення часу). Процес відбувається наступним чином:

− модуль PlayerCollision фіксує входження у тригерну зону об'єкта UniqueCapsuleBonusSlowedtime;

− здійснюється виклик методу активації бонусу, який звертається до AddMovement з вимогою тимчасово зменшити швидкість платформи або перешкод;

− одночасно бонус надсилає сигнал до ToastManager для відображення текстового повідомлення на екрані;

− у модулі бонусу запускається внутрішній таймер;

− після завершення відліку таймера викликається метод деактивації, який повертає параметри руху до початкових значень, а сам об'єкт бонусу повертається до об'єктного пулу.

Для чіткого розуміння внутрішньої логіки класів розробляються діаграми станів (State Machine Diagrams). Центральним об'єктом для розгляду моделі станів є GameManager, оскільки він керує загальним потоком гри. Цей модуль може перебувати у таких станах:

− ініціалізація (очікування завантаження сцени);

− навчання (гравець ознайомлюється з керуванням, таймери генерації зупинено);

− активна гра (основний цикл опрацювання фізики та генерації);

− пауза (викликається через модуль GamePausing, зупиняє ігровий час та блокує введення);

− завершення гри (виклик модуля GameRestart або перехід до LeaderboardTable).

Ще одним об'єктом, що змінює стани, є генератор CapsuleSpawner. Він перебуває у стані очікування до початку гри, після чого переходить у стан активної генерації, де циклічно змінює інтервали між появою нових капсул залежно від прогресу (рівня) гравця. Такий підхід гарантує, що генерація чітко синхронізована із загальним життєвим циклом ігрової сесії, запобігаючи появі предметів на сцені у невідповідні моменти. Процес зображено на Рисунку 2.2.

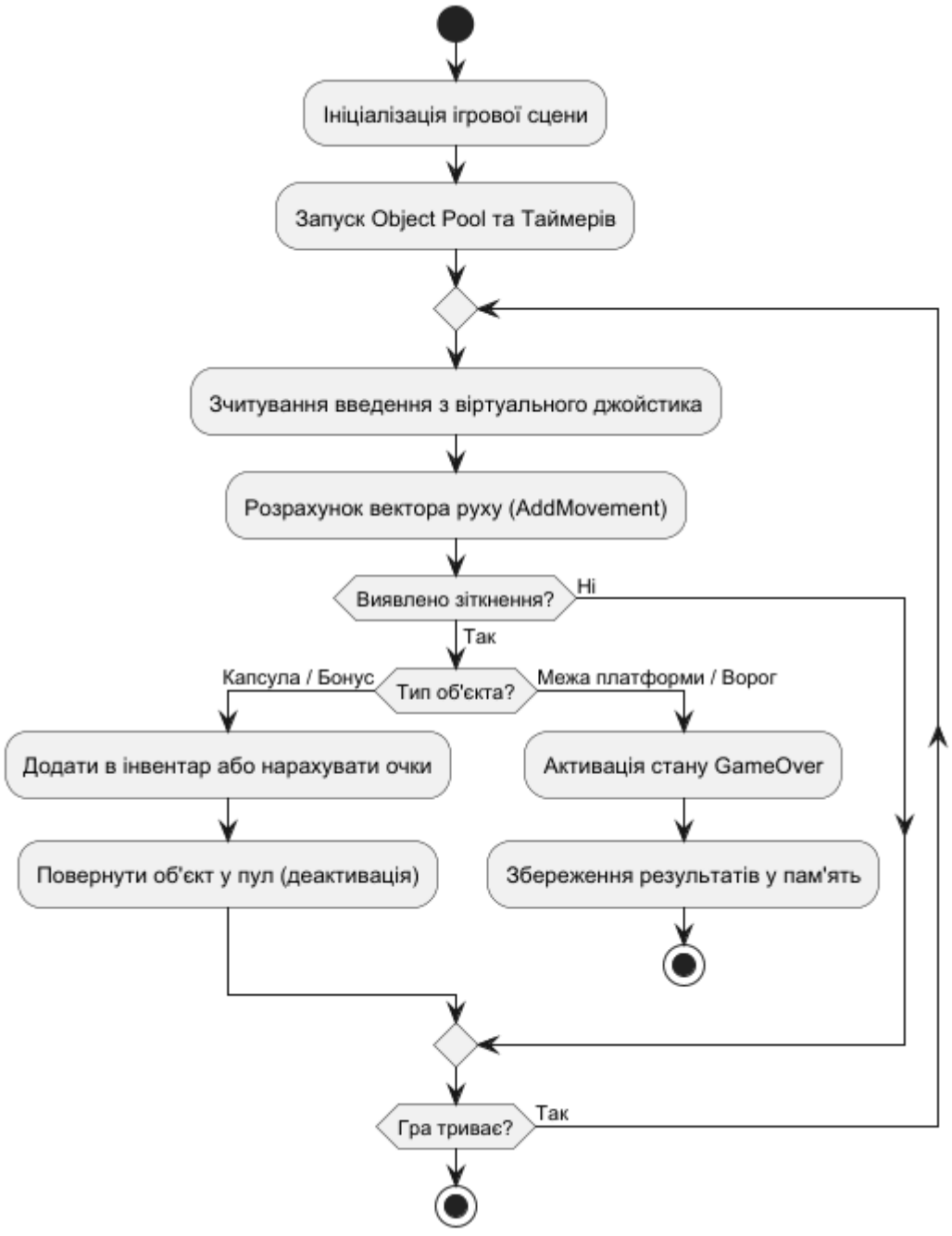


Рисунок 2.2 – Блок-схема алгоритму генерації цільових об'єктів

Завершальним етапом опису поведінки є створення блок-схем для ключових алгоритмів, логіка яких вимагає специфічних розрахунків. Для гри «Move or Die» таким є алгоритм динамічної генерації цільових капсул. Його блок-схема включає наступні кроки:

– запуск відліку таймера спавну;

– перевірка поточної кількості активних капсул на сцені (чи не перевищено ліміт);

– якщо ліміт не перевищено, виконується виклик модуля GetCharacterPosition для отримання поточних координат гравця;

– розрахунок випадкової точки появи капсули так, щоб вона знаходилася на безпечній відстані від визначених координат гравця (уникнення миттєвого підбору);

– визначення типу капсули (звичайна або унікальний бонус) за допомогою генератора псевдовипадкових чисел;

– спавн об'єкта та динамічне зменшення часу таймера на наступний цикл для поступового підвищення складності.

У підсумку, на основі проведеної модульної декомпозиції, визначення інтерфейсів та опису поведінкових алгоритмів формується загальна UML-діаграма класів (Class Diagram), яка зображена на Рисунку 2.3.

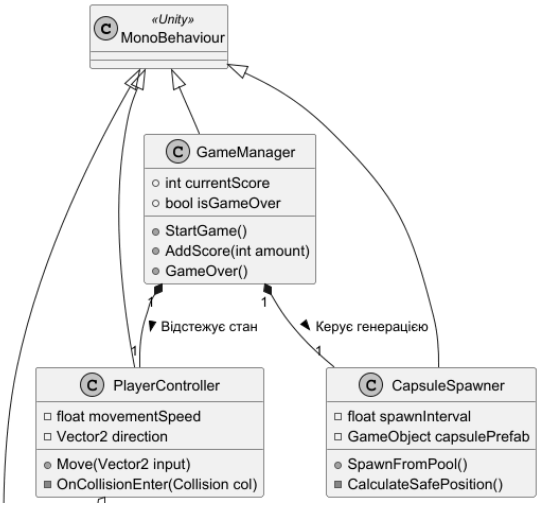


Рисунок 2.3 – Загальна UML-діаграма класів застосунку

2.7 Проєктування структури та механізмів збереження даних

Забезпечення тривалого зберігання ігрових даних є критично важливою вимогою для сучасних мобільних застосунків, оскільки це дозволяє підтримувати безперервність прогресу користувача та зберігати індивідуальні налаштування інтерфейсу між різними сесіями роботи програми. Для проєкту «Move or Die» було визначено необхідність збереження двох основних категорій інформації: параметрів конфігурації аудіосистеми та рекордних результатів ігрового процесу.

У процесі проєктування було проведено порівняльний аналіз методів серіалізації даних у середовищі Unity. З огляду на казуальний характер аркади та відносно малий обсяг інформації, використання повноцінних реляційних баз даних було визнано недоцільним через високі накладні витрати ресурсів. Оптимальним рішенням визначено застосування вбудованої підсистеми PlayerPrefs, яка забезпечує швидкий доступ до невеликих обсягів даних за принципом «ключ-значення» та підтримується на всіх цільових мобільних платформах.

Логічна структура збережених даних представлена сукупністю параметрів, доступ до яких здійснюється за допомогою зарезервованих ідентифікаторів:

− BestScore (тип даних Integer) – відображає найвищий результат, зафіксований у модулі LeaderboardTable;

− MusicVol (тип даних Float) – числовий коефіцієнт гучності, що зчитується модулем SliderMusicChange;

− CurrentLevel (тип даних Integer) – індикатор останнього відкритого рівня складності для візуалізації у ProgressBarController.

Архітектура взаємодії зі сховищем даних побудована на принципі мінімізації операцій введення-виведення. Завантаження параметрів ініціюється модулем GameSettings лише під час запуску застосунку. Оновлення даних у пам'яті мобільного пристрою відбувається подійно: у момент зміщення повзунка

гучності в інтерфейсі або при виклику методу завершення ігрової сесії у випадку досягнення гравцем нового рекорду. Таке проєктування механізму збереження гарантує високу швидкість роботи застосунку та надійне збереження прогресу гравця без створення додаткового навантаження на апаратну частину мобільного пристрою. Дана взаємодія зображена на Рисунку 2.4.

Рисунок 2.4 – Структура механізму збереження локальних даних

2.8 Висновки

У другому розділі проведено комплексне проєктування архітектури та функціональних модулів мобільної аркадної гри «Move or Die». На основі аналізу специфіки проєкту та вимог до мобільних ігрових систем обґрунтовано вибір змішаної архітектури, що поєднує компонентно-орієнтований підхід ігрового рушія Unity із подійно-орієнтованою парадигмою взаємодії. Таке рішення дозволило забезпечити низьку зв’язність програмних компонентів, що є необхідною умовою для гнучкої розробки та подальшого масштабування ігрового контенту.

Здійснено детальну декомпозицію застосунку на автономні модулі, що дозволило чітко розмежувати відповідальність між системами керування

гравцем, генерації об'єктів, обробки бонусів та візуалізації інтерфейсу. Описано інтерфейси взаємодії модулів та їхні залежності, що виключило можливість виникнення критичних помилок при одночасній роботі кількох систем. Особливу увагу приділено детальному проєктуванню алгоритмів переміщення та динамічного спавну цільових капсул, логіка яких базується на використанні патернів проєктування «Одинак», «Об’єктний пул» та «Спостерігач» для забезпечення високої продуктивності застосунку.

Проєктування інтерфейсу користувача було виконано з урахуванням ергономіки мобільних пристроїв, що забезпечило зручний доступ до елементів керування та індикації прогресу. Визначено структуру та механізми збереження ігрових даних та налаштувань, що гарантує надійне утримання прогресу користувача між сесіями. Сформований набір архітектурних рішень, UML-моделей та алгоритмічних схем є цілісним технічним підґрунтям для переходу до етапу безпосередньої програмної реалізації проєкту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1 Програмна реалізація основних модулів ігрової логіки

Програмна реалізація ігрового застосунку «Move or Die» базується на спроєктованих раніше архітектурних моделях та діаграмах. Основу програмного коду складають класи об'єктно-орієнтованої мови C#, які успадковують базовий клас MonoBehaviour. Це забезпечує їхню тісну інтеграцію у життєвий цикл ігрового рушія Unity та дозволяє прикріплювати ці скрипти безпосередньо до ігрових об'єктів (GameObject) на сцені для надання їм відповідної поведінки [21]. Такий компонентно-орієнтований підхід дозволяє гнучко налаштовувати параметри кожного об'єкта безпосередньо через візуальний редактор Unity Inspector, мінімізуючи необхідність жорсткого хардкодування (hardcoding) значень у скриптах.

Головним керуючим компонентом системи виступає клас GameManager. Для забезпечення єдиної точки доступу до глобальних станів застосунку та уникнення дублювання керівних об'єктів його реалізовано з використанням шаблону проєктування «Одинак» (Singleton) [22]. Програмно це вирішено шляхом створення статичної змінної екземпляра та використання системного методу DontDestroyOnLoad(), який запобігає знищенню об'єкта під час перезавантаження ігрових сцен. Даний клас повністю інкапсулює логіку ініціалізації ігрової сесії, централізованого підрахунку набраних балів, а також контролю переходу між етапами гри та виклику екранів завершення рівня. Крім того, на рівні цього менеджера реалізовано обробку системних переривань ОС Android: метод OnApplicationPause() відстежує згортання застосунку або вхідні дзвінки, автоматично зупиняючи ігровий час (встановлюючи Time.timeScale = 0f), що гарантує збереження прогресу гравця у непередбачуваних ситуаціях [23].

Фізична взаємодія та механіка переміщення головного ігрового персонажа (куба) реалізовані у класі AddMovement. Зважаючи на архітектурні особливості рушія Unity, розрахунки фізики винесені у спеціалізований метод FixedUpdate(),

який, на відміну від стандартного Update(), викликається із фіксованим часовим кроком. Для досягнення плавного переміщення в ізометричній перспективі та забезпечення незалежності швидкості гравця від частоти кадрів (Frame Rate Independence), застосовано алгоритм передачі фізичних імпульсів до компонента твердого тіла (Rigidbody) за допомогою методу AddForce(), замість прямої маніпуляції вектором координат transform.position. Такий підхід гарантує, що підсистема PhysX коректно та реалістично опрацьовуватиме масу об'єкта, сили тертя та зіткнення з межами платформи або іншими перешкодами [24].

Зчитування команд користувача винесено в окремий клас PlayerController, який функціонує як абстракція над системою введення. Він безперервно відстежує координати дотику до віртуального джойстика на екрані мобільного пристрою. Ключовою математичною операцією у цьому модулі є нормалізація отриманого вектора напрямку (Vector2.normalized). Це необхідно для того, щоб діагональний рух джойстика не призводив до прискорення персонажа (оскільки довжина діагоналі квадрата більша за його сторону). Також у коді реалізовано механізм «мертвої зони» (Deadzone), який ігнорує мікродотики та тремтіння пальців, передаючи до модуля AddMovement лише чітко верифіковані вектори напрямку для застосування відповідної сили з урахуванням константи швидкості.

Логіку динамічної генерації цільових капсул запрограмовано у спеціалізованому модулі CapsuleSpawner. Найбільшим технічним викликом під час розробки алгоритмів генерації для мобільних пристроїв є оптимізація оперативної пам'яті. Використання стандартних функцій Instantiate() та Destroy() для кожної капсули призводило б до сильної фрагментації пам'яті та постійного виклику збирача сміття (Garbage Collector), що провокує мікрозависання гри (GC Spikes). Для вирішення цієї проблеми програмно реалізовано патерн «Пул об'єктів» (Object Pool) на базі структури даних Queue<GameObject> [25]. На старті сцени система одноразово генерує необхідний ліміт неактивних капсул. Коли алгоритм спавну потребує новий об'єкт, він дістає його з черги, змінює координати та активує (SetActive(true)), а при підборі гравцем – просто деактивує

та повертає в пул. Це забезпечило нульове виділення пам'яті (Zero Allocation) під час активного ігрового процесу.

Управління часом генерації реалізовано за допомогою механізму співпрограм (Coroutines), який дозволяє виконувати асинхронні затримки без блокування основного ігрового потоку рендерингу. Алгоритм генерації працює циклічно з використанням конструкції yield return new WaitForSeconds(). Після завершення заданого часового інтервалу обчислюються випадкові координати в межах полігона ігрової платформи. Для запобігання появі капсул безпосередньо на ігровому персонажі алгоритм звертається до допоміжного модуля GetCharacterPosition. Програма розраховує дистанцію між точкою спавну та гравцем за допомогою методу Vector3.Distance(). Якщо розрахована точка спавну знаходиться ближче мінімально допустимого радіуса, координати рекурсивно перераховуються. Зі збільшенням набраних балів час затримки у співпрограмі динамічно зменшується за лінійною або логарифмічною функцією, що забезпечує поступове та безперервне підвищення складності гри.

Обробку подій взаємодії з ігровим оточенням зосереджено у скрипті PlayerCollision. У ньому реалізовано перевизначення системного методу OnTriggerEnter(), який спрацьовує при перетині колайдера гравця з тригерними зонами інших об'єктів. Програмний алгоритм здійснює ідентифікацію об'єкта за його строковим тегом, при цьому для оптимізації порівняння рядків використовується високоефективний метод CompareTag(), який не створює зайвого сміття в пам'яті. У разі підбору звичайної капсули модуль генерує подію (на базі делегатів мови C# Action), яку перехоплює GameManager для оновлення рахунку. Використання подійної архітектури (Event-Driven Architecture) повністю усуває жорстку залежність (Tight Coupling) між скриптом фізики та скриптом інтерфейсу. Якщо ж зафіксовано зіткнення з унікальним бонусом, ініціюється виклик відповідного спеціального ефекту.

Система унікальних бонусів побудована за принципами поліморфізму та високої модульності, спираючись на принцип відкритості/закритості

(Open/Closed Principle) з архітектури SOLID. Кожен бонус представлений окремим скриптом у просторі імен UniqueCapsuleBonuses та імплементує єдиний базовий інтерфейс. Наприклад, скрипт UniqueCapsuleBonusSlowedtime містить логіку маніпуляції глобальним масштабом часу. Він плавно зменшує значення Time.timeScale, тимчасово уповільнюючи всі динамічні процеси на сцені, водночас пропорційно збільшуючи силу AddMovement, щоб швидкість самого гравця залишалася незмінною. Скрипт UniqueCapsuleBonusMagnetism реалізує алгоритм поступового притягування всіх активних об'єктів до поточних координат гравця за допомогою математичної функції лінійної інтерполяції векторів Vector3.MoveTowards(), що виконується у циклі оновлення кадру.

Кожен такий скрипт самостійно керує життєвим циклом ефекту через внутрішні таймери співпрограм. Після їхнього завершення логіка скрипта автоматично скасовує всі внесені зміни (наприклад, повертає час до нормального стану або вимикає магнетизм) і повертає об'єкт бонуса до відповідного пулу. Такий високорівневий контроль стану гарантує стабільну роботу застосунку, унеможливлює виникнення багів при одночасній активації кількох різних бонусів та створює надійний фундамент для легкого масштабування гри шляхом додавання нових типів ігрових модифікаторів у майбутньому.

3.2 Програмна реалізація інтерфейсу користувача та аудіовізуальних ефектів

Розроблення графічного інтерфейсу користувача (UI) для проєкту «Move or Die» здійснено з використанням вбудованої підсистеми Unity UI (uGUI). Базовим структурним елементом інтерфейсу виступає об'єкт Canvas, який слугує середовищем для рендерингу всіх графічних компонентів (кнопок, тексту, панелей). Для забезпечення коректного, пропорційного та якісного відображення на мобільних пристроях із різною роздільною здатністю та співвідношенням

сторін екрана застосовано компонент Canvas Scaler у режимі Scale With Screen Size [26], який являється найбільш застосованим.

З метою оптимізації продуктивності на мобільних платформах та уникнення ресурсомісткого процесу повного перемалювання всього інтерфейсу (Canvas Rebuild) при зміні одного елемента, архітектуру інтерфейсу розділено на статичні та динамічні під-полотна (Sub-Canvas). Статичні елементи (фони, рамки) відмальовуються окремо від динамічних (шкала прогресу, таймери, лічильник очок) [27]. Як це реалізовано у застосунку відображено на Рисунку 3.1.

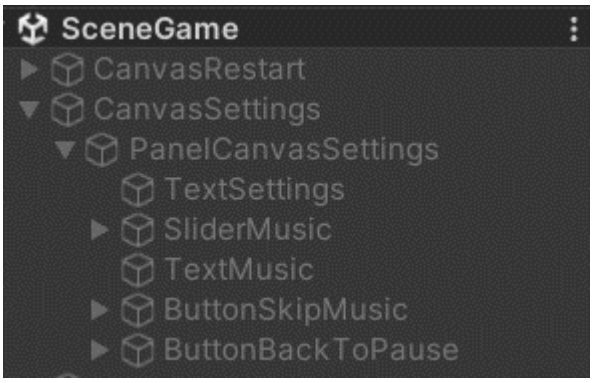


Рисунок 3.1 – Ієрархія об'єктів підсистеми UI в інспекторі Unity

Обробка користувацького введення (дотиків до екрана) реалізована за допомогою системного модуля EventSystem та компонента Graphic Raycaster. Для зменшення зв'язності (coupling) між логікою графічної оболонки та ігровим ядром застосовано подійно-орієнтований підхід: кнопки інтерфейсу не викликають методи ігрових класів безпосередньо, а генерують події (UnityEvents), на які підписані відповідні контролери. Наприклад, одна подія від кнопки інтерфейсу може одночасно перехоплюватися кількома незалежними підсистемами: аудіоменеджером для відтворення звуку натискання, менеджером анімацій та безпосередньо ігровим контролером. При цьому відсутність жорстких посилань запобігає виникненню критичних помилок у логіці (зокрема NullReferenceException) у випадку видалення чи тимчасового вимкнення певних елементів графічного інтерфейсу (UI). Реалізація зображена на Рисунку 3.2.

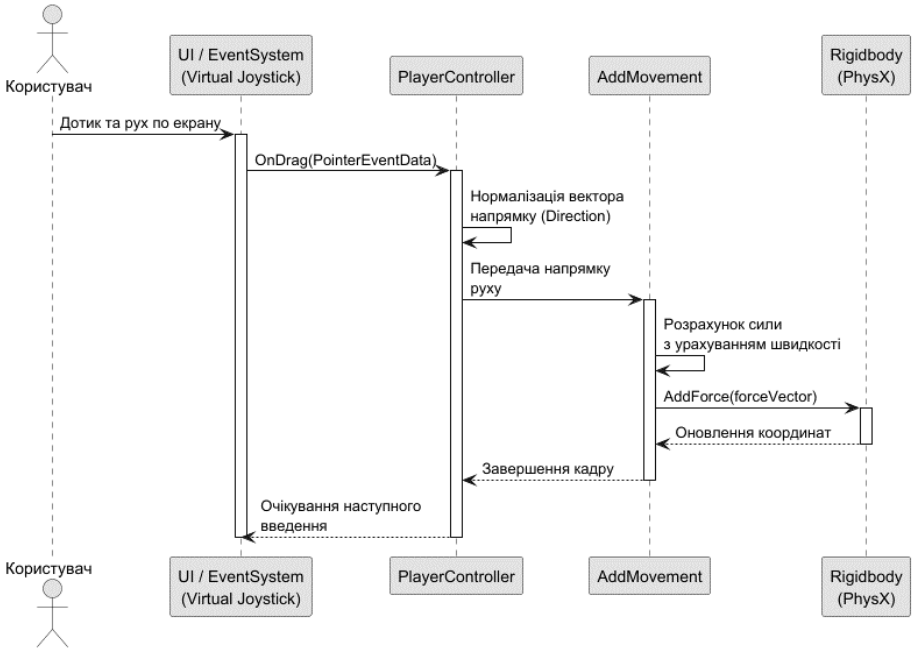


Рисунок 3.2 – Процес обробки користувацького введення та передачі команд до ігрового ядра

Головне вікно взаємодії з користувачем керується модулем StartMenu. Програмна логіка цього компонента інкапсулює процеси перемикання між активними станами меню: від початкового екрана до вікна налаштувань та таблиці лідерів. Для забезпечення безпеки користувацького досвіду та запобігання випадковому закриттю застосунку розроблено модуль GameLeaveConfirmation. Цей скрипт ініціює появу модального діалогового вікна, яке перехоплює фокус управління (блокує взаємодію з іншими елементами) до моменту отримання явного підтвердження або скасування дії від гравця. Реалізація цієї логіки побудована на принципі керування видимістю графічних об'єктів, що дозволяє миттєво перемикатися між екранами. Використання модальних вікон із перехопленням подій введення повністю відповідає сучасним стандартам UX-дизайну мобільних додатків, оскільки ефективно мінімізує ризик втрати ігрового прогресу через хибні або випадкові натискання. На Рисунку 3.3 зображено те, як виглядають переходи у самому застосунку.

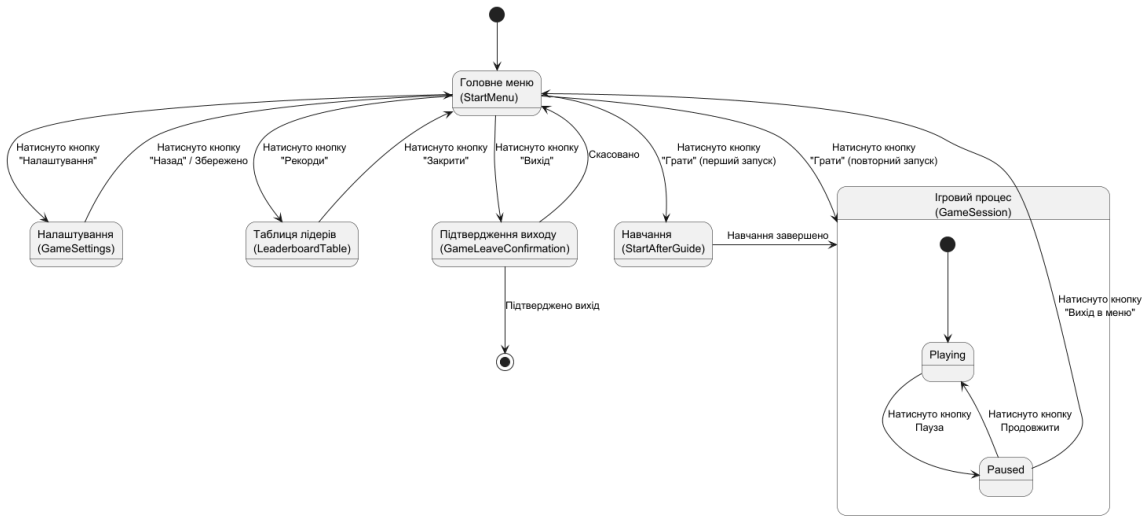


Рисунок 3.3 – Переходи між екранами навігаційного меню застосунку

Було спроєктовано макет головного меню, який буде використано в застосунку (Рисунок 3.4).

Рисунок 3.4 – Скріншот макета головного меню гри на екрані мобільного пристрою

Під час безпосереднього ігрового процесу оперативна інформація виводиться на екран за допомогою системи HUD (Head-Up Display). Ключовим візуальним індикатором є шкала прогресу, функціонування якої забезпечується

модулем ProgressBarController. Програмно цей клас взаємодіє з системним компонентом Slider. Алгоритм розраховує ступінь заповнення шкали як математичне відношення поточної кількості очок, переданих від GameManager, до порогового значення наступного рівня. Для забезпечення плавності анімації при нарахуванні балів застосовано математичну функцію лінійної інтерполяції (Mathf.Lerp). Ігровий інтерфейс застосунку зображено на Рисунку 3.5.

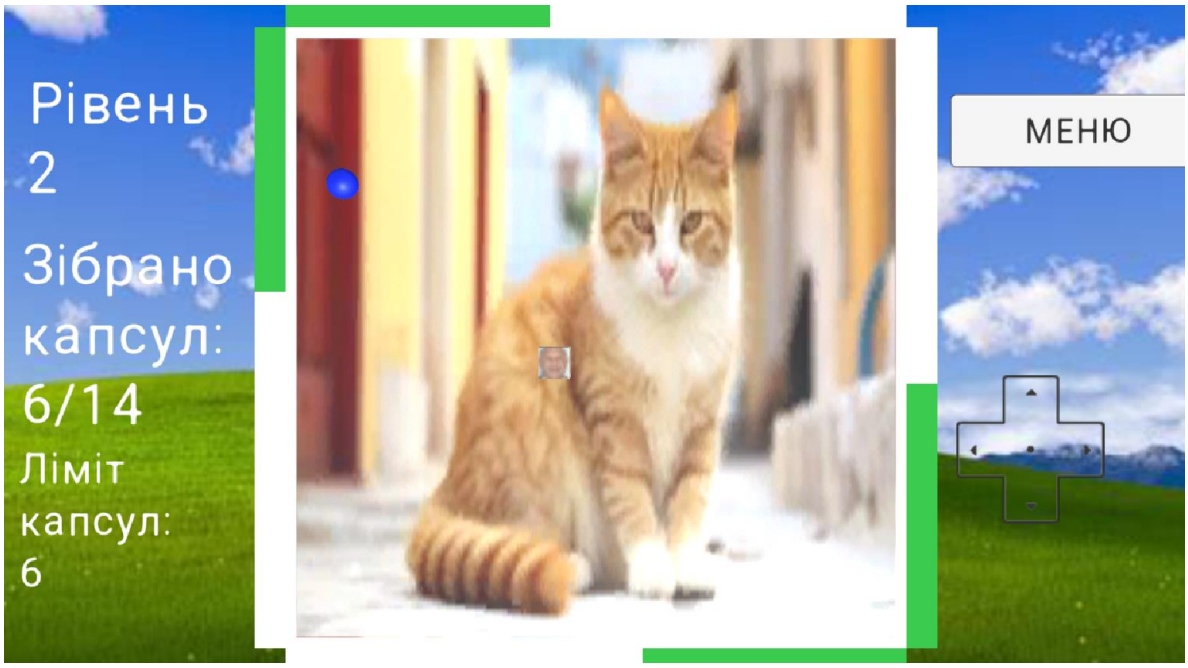


Рисунок 3.5 – Скріншот ігрового інтерфейсу із відображенням шкали прогресу, лічильника та віртуального джойстика

Для інформування гравця про локальні ігрові події (наприклад, підбір унікального бонусу «Магнетизм» чи «Уповільнення часу») без переривання ігрового процесу реалізовано підсистему динамічних сповіщень – модуль ToastManager, роботу якого зображено на Рисунку 3.6. Для їхньої оптимізації застосовано патерн «Об'єктний пул» (Object Pool). Скрипт керує життєвим циклом текстового префаба: при виклику методу ShowToast об'єкт активується, змінює координати по осі Y для ефекту спливання, після чого плавно деактивується через зміну прозорості (альфа-каналу) у компоненті CanvasGroup.

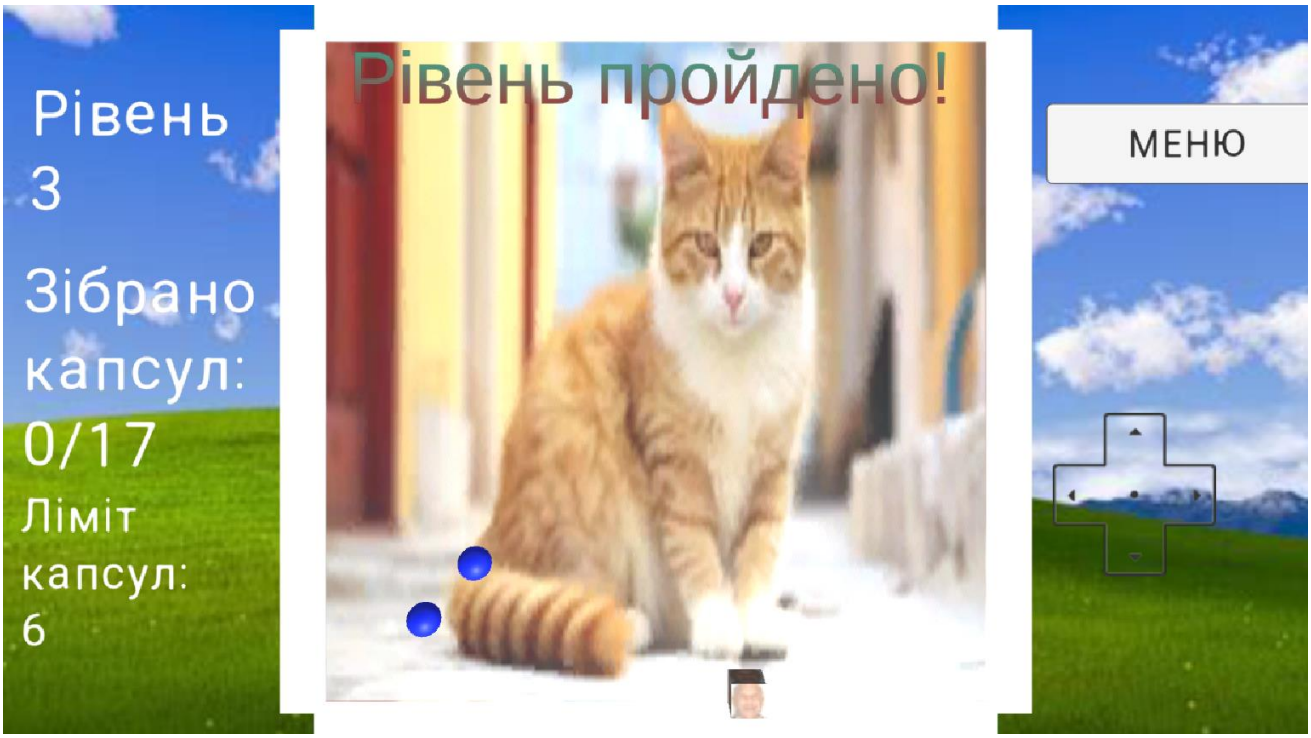


Рисунок 3.6 – Демонстрація роботи системи спливаючих сповіщень (Toast) під час активної гри

Аудіовізуальна складова гри інтегрована для посилення ефекту занурення. Звукова підсистема побудована на базі глобального компонента AudioMixer, що дозволило розділити аудіопотоки на незалежні канали (Master, Music, SFX) та централізовано керувати їхніми параметрами [28]. Модуль MusicRandomizer відповідає за циклічне та випадкове відтворення фонових треків із масиву AudioClip, виключаючи повторення поточного треку. Модуль SliderMusicChange зчитує положення повзунків у меню налаштувань, конвертує лінійні значення у логарифмічну шкалу децибел (dB) для коректного сприйняття звуку людським вухом та зберігає ці параметри у локальному сховищі PlayerPrefs. Використання AudioMixer надає гнучкість у маршрутизації звукових сигналі та забезпечує високу продуктивність за рахунок оптимізованої обробки аудіо. Така архітектура гарантує легке масштабування проєкту, а інтеграція нових звукових ефектів чи музичних композицій у майбутньому не вимагатиме втручання у базовий програмний код. Схема маршрутизації аудіосигналів відображена на Рисунку 3.7.

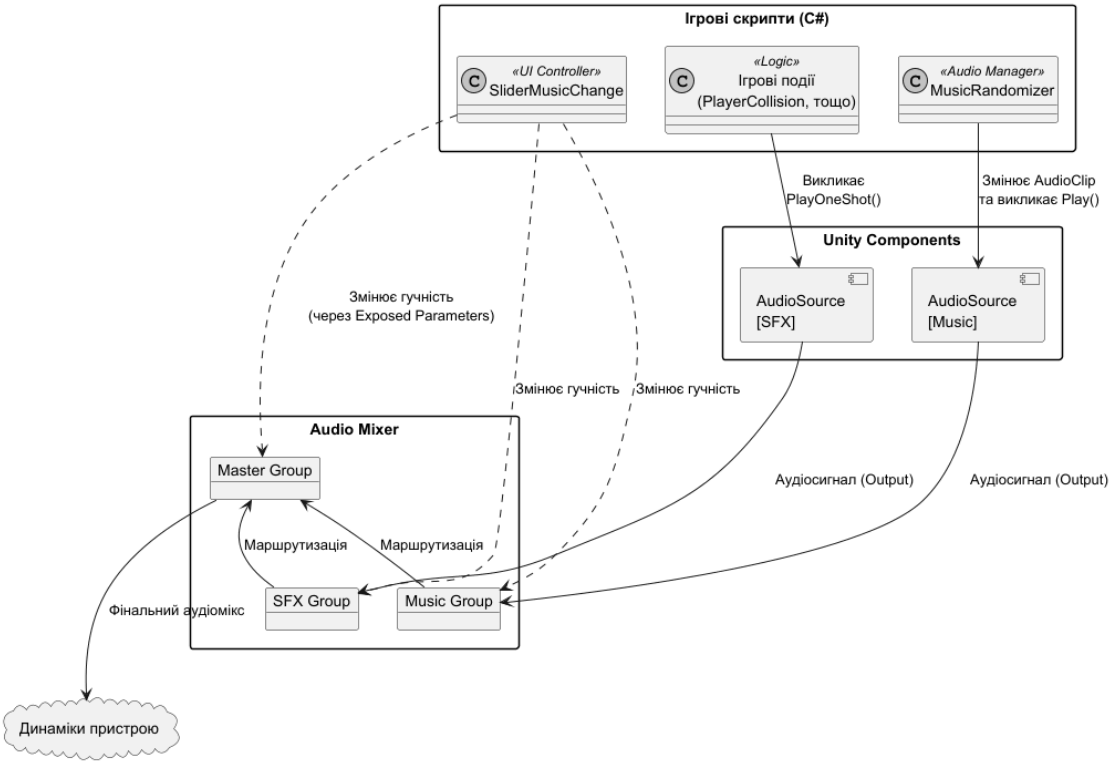


Рисунок 3.7 – Схема маршрутизації аудіосигналів: Структура груп Audio Mixer та їхній зв'язок зі скриптами

Візуальна стилістика цільових та бонусних об'єктів реалізована шляхом програмування користувацьких шейдерів. Модуль RainbowShaderForCapsule містить алгоритм, який використовує математичну функцію синуса залежно від ігрового часу (Time.time) для циклічної зміни колірних координат моделі (RGB-каналів). Це створює безперервний ефект переливання кольорів. Додатково, при ініціалізації події підвищення рівня у GameManager, викликається скрипт ChangeMaterialWhenLevelUp, який динамічно замінює посилання на матеріали у компонентах MeshRenderer ігрової платформи, забезпечуючи чіткий візуальний відгук на зміну складності. Такий підхід до реалізації візуальних ефектів є найбільш раціональним для мобільних платформ. Перенесення обчислень колірних переходів на рівень шейдерів дозволяє розвантажити центральний процесор та делегувати ці обчислення графічному процесору, що суттєво заощаджує ресурси пристрою та підтримує стабільну частоту кадрів. Візуальна складова зображена на Рисунку 3.8.

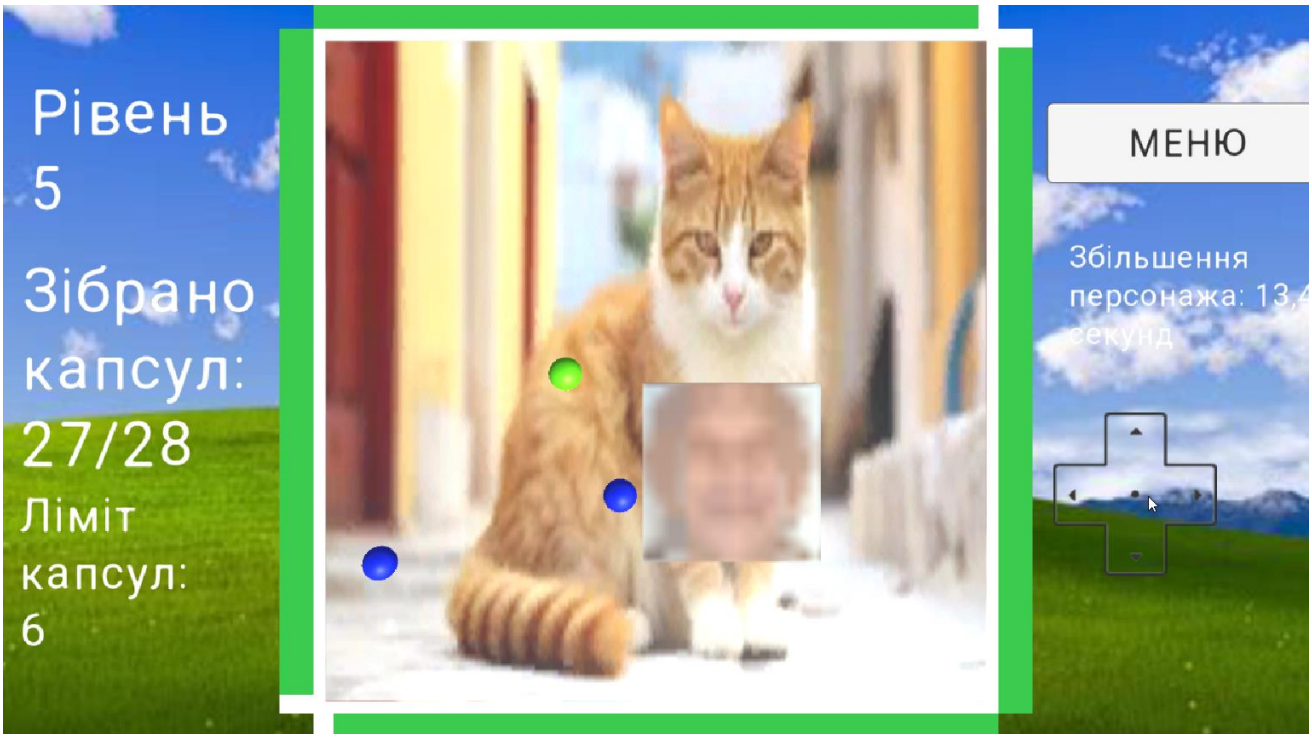


Рисунок 3.8 – Візуальні ефекти бонусних капсул та зміна матеріалу оточення при підвищенні рівня

Загалом, програмна реалізація інтерфейсу та аудіовізуальної підсистеми виконана з дотриманням принципів модульності та оптимізації, що гарантує високу продуктивність (стабільний FPS) та відсутність затримок керування на мобільних пристроях.

3.3 Вимоги до технічних та програмних засобів

Визначення вимог до технічних і програмних засобів є необхідним етапом підготовки програмного продукту до релізу. Оскільки розроблений ігровий застосунок «Move or Die» орієнтований на мобільні пристрої під управлінням операційної системи Android, критично важливо сформувати чіткі критерії апаратного забезпечення. Дотримання цих критеріїв гарантуватиме стабільну частоту кадрів, швидке завантаження сцен та відсутність збоїв через нестачу оперативної пам'яті.

У таблиці 3.1 наведено деталізовані мінімальні та рекомендовані системні вимоги до кінцевого пристрою користувача (смартфона або планшета) для забезпечення стабільного ігрового процесу.

Таблиця 3.1 – Мінімальні та рекомендовані системні вимоги до застосунку «Move or Die»

Характеристика	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Android 8.0 (Oreo)	Android 11.0 або новіша версія
Процесор (CPU)	4-ядерний, архітектура ARMv7 (32-bit)	8-ядерний, архітектура ARM64 (64-bit)
Оперативна пам'ять (RAM)	2 ГБ	4 ГБ або більше
Графічний прискорювач (GPU)	З підтримкою API OpenGL ES 3.0	З підтримкою Vulkan API або OpenGL ES 3.2
Вільний дисковий простір	50 МБ	100 МБ (внутрішній Flash-накопичувач)
Засоби введення	Сенсорний екран (Touchscreen)	Сенсорний екран із підтримкою Multi-touch
Доступ до мережі	Не вимагається (повний офлайн-режим)	Не вимагається

Завдяки використанню оптимізаційних патернів проєктування (зокрема, «Об'єктного пулу»), ефективному управлінню життєвим циклом об'єктів та застосуванню оптимізованих шейдерів, застосунок не є критично вимогливим до ресурсів системи. Це забезпечує широке охоплення цільової аудиторії, дозволяючи запускати гру навіть на бюджетних мобільних пристроях минулих поколінь.

Окрім вимог до цільового пристрою, для подальшої підтримки проєкту, модифікації вихідного коду та компіляції нових збірок (білдів) існують специфічні вимоги до робочої станції розробника.

Апаратне забезпечення розробника повинно включати багатоядерний процесор (рівня Intel Core i5 / AMD Ryzen 5 або вище), мінімум 8 ГБ оперативної пам'яті (рекомендовано 16 ГБ для комфортної роботи індексатора коду) та твердотільний накопичувач (SSD) для швидкого завантаження бібліотек проєкту. З програмного забезпечення необхідна наявність десктопної операційної системи (Windows 10/11 або macOS), встановленого ігрового рушія Unity (бажано версії з довгостроковою підтримкою – LTS), інтегрованого середовища розробки з підтримкою мови C# (JetBrains Rider або Microsoft Visual Studio), а також пакетів Android SDK та Android NDK для забезпечення можливості кросплатформної збірки продукту.

3.4 Тестування програмного забезпечення

3.4.1 Методи та засоби тестування

Тестування програмного забезпечення є невід’ємним та критично важливим етапом життєвого циклу розроблення ігрового застосунку. Його головна мета полягає у виявленні дефектів, перевірці відповідності реалізованого функціоналу початковим вимогам (валідація) та забезпеченні стабільної роботи продукту на цільових платформах (верифікація).

Для забезпечення високої якості кінцевого продукту під час переддипломної практики було обрано комплексну стратегію, яка включає декілька рівнів та методів тестування: функціональне, системне, тестування інтерфейсу користувача та продуктивності.

Функціональне тестування (Functional Testing) застосовується для перевірки коректності роботи окремих ігрових механік. Основним методом на цьому етапі є тестування «чорного ящика» (Black-box testing), коли перевіряється реакція системи на конкретні вхідні дані без втручання у внутрішній код [29]. Відповідно до архітектури застосунку, тестуванню підлягають:

- модуль переміщення гравця (AddMovement): перевірка плавності ходу та реакції колайдерів на зіткнення з межами платформи;
- система генерації (CapsuleSpawner): перевірка алгоритму обчислення безпечних координат та дотримання інтервалів таймера;
- система унікальних бонусів: валідація ефектів від кожного типу бонусу (уповільнення часу, магнетизм тощо) та коректності їхнього скасування після завершення дії таймера.

Системне тестування (System Testing) спрямоване на перевірку цілісної роботи застосунку як єдиного механізму. Перевіряється правильність переходів між станами гри: запуск головного меню (StartMenu), перехід до етапу навчання, ініціалізація активної гри, виклик паузи та логіка завершення ігрової сесії. Окремо перевіряється механізм збереження налаштувань користувача (гучність аудіо) і рекордів за допомогою вбудованого модуля PlayerPrefs.

Тестування інтерфейсу користувача (UI Testing) передбачає перевірку ергономіки та адаптивності графічної оболонки за допомогою інструмента Unity Device Simulator. Перевіряється робота компонента Canvas Scaler, читабельність текстових сповіщень (ToastManager) та зручність взаємодії з віртуальним джойстиком на екранах із різним співвідношенням сторін, оскільки мобільні пристрої бувають різних розмірів, і необхідно задовільнити максимальну кількість потенційних гравців (зокрема 16:9, 18:9 та 21:9).

Тестування продуктивності (Performance Testing) є ключовим для мобільних ігор. Його мета – забезпечити стабільну частоту кадрів (не менше 60 FPS) та уникнути витоків пам'яті, які можуть призвести до аварійного завершення програми. Для проведення цього виду тестування використовується вбудований інструмент Unity Profiler, який аналізує:

− навантаження на центральний процесор (CPU Usage) [30]: відстеження часу виконання ресурсомістких скриптів та обробки фізики (PhysX);

− споживання оперативної пам'яті (Memory Allocation): контроль за обсягом пам'яті, що виділяється під час активної ігрової сесії;

− роботу збирача сміття (Garbage Collector Allocation): перевірка ефективності реалізованого патерну «Об'єктний пул» (Object Pool) та відсутності неконтрольованого створення і знищення об'єктів.

3.4.2 Проведення тестування та аналіз результатів

Процес тестування розробленого ігрового застосунку здійснювався у два етапи: перевірка в середовищі розробки (Unity Editor) з використанням вбудованих засобів відлагодження та тестування скомпільованого продукту (APK-файлу) на реальному мобільному пристрої під управлінням операційної системи Android.

Під час тестування інтерфейсу у середовищі Unity Device Simulator було змодельовано відображення на різних пристроях. Компонент Canvas Scaler відпрацював штатно: кнопки меню та ігровий HUD залишалися у межах видимої зони без спотворень та накладання тексту. Успішні результати цього етапу тестування засвідчують готовність візуальної частини застосунку до стабільного функціонування на всьому спектрі цільових смартфонів. На Рисунку 3.9 зображено скріншот ігрового застосунку у режимі симуляції на пристроях із різним співвідношенням сторін екрана.

Рисунок 3.9 – Скріншот ігрового застосунку у режимі симуляції на пристроях із різним співвідношенням сторін екрана

Для перевірки коректності роботи ігрової логіки було розроблено набір тестових сценаріїв, результати виконання яких під час проходження переддипломної практики наведено у Таблиці 3.2. Структура тестів була сформована таким чином, щоб ізольовано перевірити кожен функціональний блок застосунку та зафіксувати фактичну реакцію системи на визначені вхідні параметри порівняно з очікуваним результатом, оскільки мобільні застосунки мають тенденцію мати різні неочікувані проблеми, і тому дуже важливо ретельно провести тестування.

Таблиця 3.2 – Результати функціонального тестування ігрових механік

Ідентифікатор	Опис дії (кроки тестування)	Очікуваний результат	Фактичний результат	Статус
ТС-01	Запуск сцени. Очікування роботи CapsuleSpawner.	Капсули генеруються у випадкових координатах, не ближче заданого радіуса від гравця.	Генерація відбувається коректно, на безпечній відстані.	Успішно
ТС-02	Переміщення гравця за межі платформи.	Рух блокується фізичним колайдером, об'єкт не випадає за межі сцени.	Колайдери відпрацьовують штатно.	Успішно
ТС-03	Зіткнення гравця зі звичайною капсулою.	До лічильника додається 1 бал, оновлюється шкала, капсула деактивується.	Бал нараховується, капсула повертається у пул.	Успішно
ТС-04	Зіткнення з бонусом «Уповільнення часу».	Швидкість гри зменшується на визначений час, виводиться спливаюче сповіщення.	Час уповільнюється та відновлюється коректно.	Успішно
ТС-05	Зіткнення з бонусом «Магнетизм».	Усі капсули починають рухатися до координат гравця.	Капсули притягуються без порушення фізики оточення.	Успішно

Перевірка продуктивності проводилася за допомогою Unity Profiler. Аналіз модуля CPU Usage показав, що основне навантаження припадає на рендеринг та розрахунки підсистеми PhysX. Час обробки одного кадру не перевищує 16 мілісекунд, що забезпечує стабільну частоту 60 FPS на тестовому мобільному пристрої. Критичних витоків пам'яті (memory leaks) або різких падінь кадрової частоти протягом усього періоду профілювання виявлено не було, що підтверджує готовність проєкту до релізу. Аналіз відображено на Рисунку 3.10.

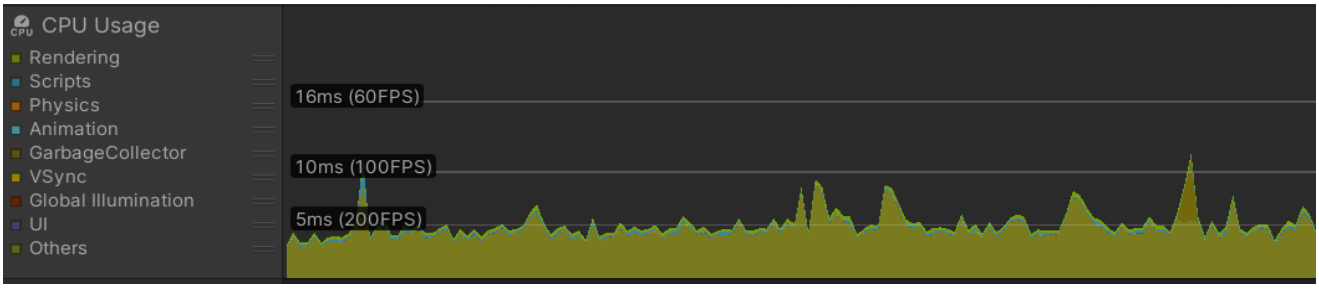


Рисунок 3.10 – Графік CPU Usage у вікні Unity Profiler, що демонструє стабільний час обробки кадру під час ігрової сесії

Завдяки впровадженню архітектурного шаблону «Об'єктний пул» для циклічної генерації капсул вдалося повністю уникнути ресурсомістких викликів методів Instantiate та Destroy. Аналіз графіка Garbage Collector Allocations у підсистемі Memory Profiler продемонстрував відсутність різких стрибків виділення пам'яті, що підтверджує відсутність мікрозатримок (фрізів) під час тривалих ігрових сесій, що свідчить про коректну роботу застосунку. Отриманий графік зображено на Рисунку 3.11.

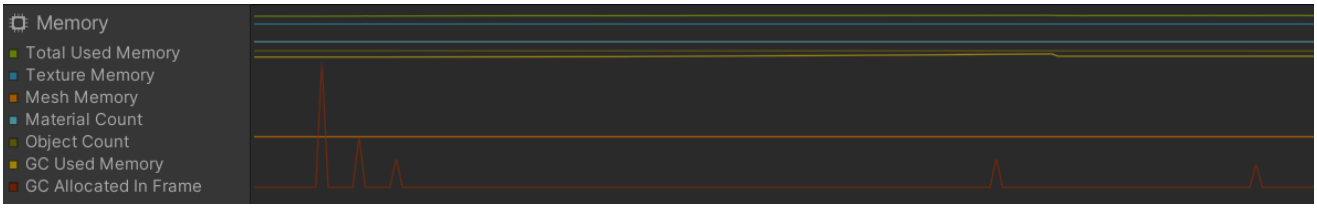


Рисунок 3.11 – Графік Memory Allocation у Unity Profiler, що підтверджує ефективну роботу об'єктного пулу та відсутність витоків пам'яті

Виявлені дефекти та їхнє усунення:

- логічний дефект: при активації меню паузи об'єкти, рух яких контролювався через співпрограми (Coroutines), продовжували переміщення, попри зупинку ігрового часу. Помилку виправлено шляхом впровадження перевірки статичного прапорця isPaused у циклах співпрограм.
- дефект інтерфейсу: на екранах із великим співвідношенням сторін (21:9) віртуальний джойстик частково виходив за межі видимої зони. Проблему розв'язано через переналаштування якорів (Anchors) відносно нижніх кутів SafeArea.

У підсумку, проведене комплексне тестування підтвердило, що розроблений ігровий застосунок функціонує згідно з вимогами технічного завдання та демонструє високий рівень оптимізації програмного коду.

3.5 Висновки до третього розділу

Під час розробки та тестування було виконано повний цикл програмної реалізації та верифікації ігрового застосунку «Move or Die». На основі обраної архітектури було успішно впроваджено ключові ігрові механіки, використовуючи можливості об’єктно-орієнтованого програмування на мові C# та інструментарій ігрового рушія Unity. Особливу увагу приділено оптимізації продуктивності шляхом використання патерну «Об’єктний пул», що дозволило суттєво знизити навантаження на апаратні ресурси мобільних пристроїв та забезпечити стабільну роботу системи без критичних затримок під час інтенсивного ігрового процесу.

Графічний інтерфейс користувача та аудіовізуальна складова були реалізовані з урахуванням сучасних стандартів ергономіки та адаптивності. Завдяки коректному налаштуванню системи якорів та використанню компонентів динамічного масштабування, вдалося досягти правильного відображення

елементів керування на пристроях із різним співвідношенням сторін екрана. Побудована система маршрутизації звуку через Audio Mixer забезпечила гнучке керування звуковим супроводом, що значно покращує рівень занурення гравця в ігрову атмосферу.

Проведене комплексне тестування, яке охоплювало функціональний, системний та продуктивнісний аспекти, підтвердило повну працездатність усіх модулів застосунку. Використання засобів профілювання Unity Profiler дало змогу верифікувати відсутність витоків оперативної пам’яті та підтвердити досягнення цільового показника частоти кадрів на рівні 60 FPS. У ході тестування було оперативно виявлено та усунуто незначні дефекти в роботі співпрограм та логіці паузи, що дозволило підвищити загальну стабільність програмного коду та надійність продукту.

Підсумовуючи результати практичного етапу, можна стверджувати, що створений ігровий продукт є технічно завершеним та повністю відповідає вимогам технічного завдання. Реалізовані архітектурні рішення довели свою ефективність на практиці, а проведена оптимізація гарантує комфортну експлуатацію застосунку на широкому спектрі мобільних пристроїв під управлінням операційної системи Android. Таким чином, усі поставлені завдання третього розділу виконані в повному обсязі, що дає змогу перейти до фінального етапу оцінки результатів роботи.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання, яке полягає у проєктуванні та програмній реалізації оптимізованого мобільного ігрового застосунку «Move or Die» жанру 3D-аркада для операційної системи Android. Мета роботи, що полягала у створенні стабільного, продуктивного та конкурентоспроможного ігрового продукту із застосуванням сучасних архітектурних шаблонів, досягнута в повному обсязі.

За результатами виконання роботи можна зробити наступні висновки:

− проведено системний аналіз предметної області та ринку мобільних ігор. Дослідження існуючих програмних аналогів (Cube Surfer, Pac-Man 256 тощо) дозволило виявити їхні архітектурні та геймдизайнерські недоліки, такі як перевантаженість інтерфейсу або надмірна лінійність управління. На основі цих даних було сформовано концепцію розроблюваного застосунку та складено деталізовані функціональні й нефункціональні вимоги, орієнтовані на забезпечення високої динаміки (Core Gameplay Loop) та реіграбельності.

− спроєктовано гнучку та масштабовану архітектуру програмного забезпечення. Для забезпечення низької зв'язності (Low Coupling) модулів та дотримання принципів SOLID було обрано змішану архітектуру, що поєднує компонентну модель Unity з подійно-орієнтованою парадигмою. Розроблено комплекс UML-діаграм для наочного представлення системи. Визначено та впроваджено оптимальні шаблони проєктування, зокрема Singleton для централізованого управління сесією та Observer для ізоляції логіки користувацького інтерфейсу від математичних розрахунків.

− здійснено програмну реалізацію всіх ігрових модулів мовою C# в середовищі Unity. Успішно розроблено алгоритми фізичного переміщення персонажа, обробки колізій та інтерполяції векторів. Особливу увагу було приділено оптимізації використання оперативної пам'яті: для уникнення мікрозависань, викликаних роботою збирача сміття, процедурну генерацію

об'єктів реалізовано на базі патерну Object Pool. Створено модульну, розширювану систему унікальних бонусів (магнетизм, уповільнення часу) та імплементовано систему локального збереження даних на базі PlayerPrefs як ресурсоефективну альтернативу реляційним базам даних.

− проведено комплексне тестування та профілювання ігрового застосунку. Аналіз результатів підтвердив повну відповідність розробленого ПЗ висунутим нефункціональним вимогам. Застосунок демонструє стабільну частоту зміни кадрів на рівні 60 FPS, відсутність витоків пам'яті (Memory Leaks) та швидкий час холодного старту. Гра коректно функціонує на цільових мобільних пристроях під керуванням ОС Android (версії 8.0 та вище), забезпечуючи коректну обробку системних переривань та адаптивність інтерфейсу під різні пропорції екранів.

Практичне значення одержаних результатів полягає в тому, що розроблений мобільний ігровий застосунок «Move or Die» є повністю завершеним, оптимізованим та автономним програмним продуктом. Застосовані під час його створення архітектурні підходи та методи оптимізації можуть бути використані як методична база для подальшої розробки ресурсоємних мобільних додатків в умовах обмежених апаратних можливостей кінцевих пристроїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Newzoo. The Global Games Market Report 2023. Amsterdam : Newzoo, 2023. 45 p. URL: https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023 (дата звернення: 20.05.2026).
2. Engelstein G. Achievement Relocked: Loss Aversion and Game Design. Cambridge : MIT Press, 2020. 216 p.
3. Bycer J. Game Design Deep Dive: Free-to-Play. Boca Raton : CRC Press, 2023. 182 p.
4. Horton J. P. Unity 2020 Mobile Game Development. 2nd ed. Birmingham : Packt Publishing, 2020. 458 p.
5. Annander M. The Game Design Toolbox. Boca Raton : CRC Press, 2024. 238 p.
6. Somberg G. Game Audio Programming 4: Principles and Practices. Boca Raton : CRC Press, 2024. 356 p.
7. Summers T. The Cambridge Companion to Video Game Music. Cambridge : Cambridge University Press, 2021. 420 p.
8. Phillips B., Stewart C., Hardy B., Marsicano K. Android Programming: The Big Nerd Ranch Guide. 4th ed. Boston : Addison-Wesley Professional, 2022. 704 p.
9. Martin R. C. Functional Design: Principles, Patterns, and Practices. Boston : Addison-Wesley Professional, 2023. 352 p.
10. Summers T. The Cambridge Companion to Video Game Music. Cambridge : Cambridge University Press, 2021. 420 p.
11. Baron D. Game Development Patterns with Unity 2021. Birmingham : Packt Publishing, 2021. 338 p.
12. Albahari J. C# 10 in a Nutshell: The Definitive Reference. Sebastopol : O'Reilly Media, 2022. 1082 p.
13. Ferrone H. Learning C# by Developing Games with Unity 2021. 6th ed. Birmingham : Packt Publishing, 2021. 370 p.

14. N. Ford, M. Richards, S. Pramod, Z. Dehghani Software Architecture: The Hard Parts. Sebastopol : O'Reilly Media, 2021. 432 p.
15. Sarcar V. Design Patterns in C#: A Hands-on Guide with Real-World Examples. 2nd ed. New York : Apress, 2020. 504 p.
16. Freeman A. Pro C# 9 with .NET 5. 10th ed. New York : Apress, 2021. 1290 p.
17. Price M. J. C# 11 and .NET 7 - Modern Cross-Platform Development Fundamentals. 7th ed. Birmingham : Packt Publishing, 2022. 822 p.
18. Smith M., O'Dowd R. Unity 2021 Cookbook. 4th ed. Birmingham : Packt Publishing, 2021. 628 p.
19. Borromeo N. A. Hands-On Unity 2022 Game Development. 3rd ed. Birmingham : Packt Publishing, 2022. 574 p.
20. Yablonski J. Laws of UX: Using Psychology to Design Better Products & Services. Sebastopol : O'Reilly Media, 2020. 152 p.
21. Order of Execution for MonoBehaviour // Unity Documentation. 2024. URL: https://docs.unity3d.com/Manual/ExecutionOrder.html (дата звернення: 20.05.2026).
22. Troelsen A., Japikse P. Pro C# 10 with .NET 6. New York : Apress, 2022. 1614 p.
23. Understand the Activity Lifecycle // Android Developers. 2026. URL: https://developer.android.com/guide/components/activities/activity-lifecycle (дата звернення: 20.05.2026).
24. Chiu C. C# Game Programming Cookbook. Boca Raton : CRC Press, 2020. 418 p.
25. Introduction to Object Pooling // Unity Learn. 2023. URL: https://learn.unity.com/tutorial/introduction-to-object-pooling (дата звернення: 20.05.2026).
26. Murray J. Building Games with Unity and Blender. New York : Springer, 2021. 290 p.

27. Introduction to UI Toolkit // Unity Documentation. 2025. URL: https://docs.unity3d.com/Manual/UIElements.html (дата звернення: 20.05.2026).
28. Somberg G. Game Audio Programming 3: Principles and Practices. Boca Raton : CRC Press, 2020. 412 p.
29. Black R. Foundations of Software Testing: ISTQB Certification. 5th ed. London : Cengage Learning, 2020. 280 p.
30. Profiler Window // Unity Documentation. 2025. URL: https://docs.unity3d.com/Manual/ProfilerWindow.html (дата звернення: 20.05.2026).
31. Бедратюк Л. П., Радельчук Г. І., Форкун Ю. В., Яшина О. М. Дипломний проект: методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення». Хмельницький : ХНУ, 2020. 77 с.

ДОДАТОК А

(обов’язковий)

ГРАФІЧНІ МАТЕРІЛИ

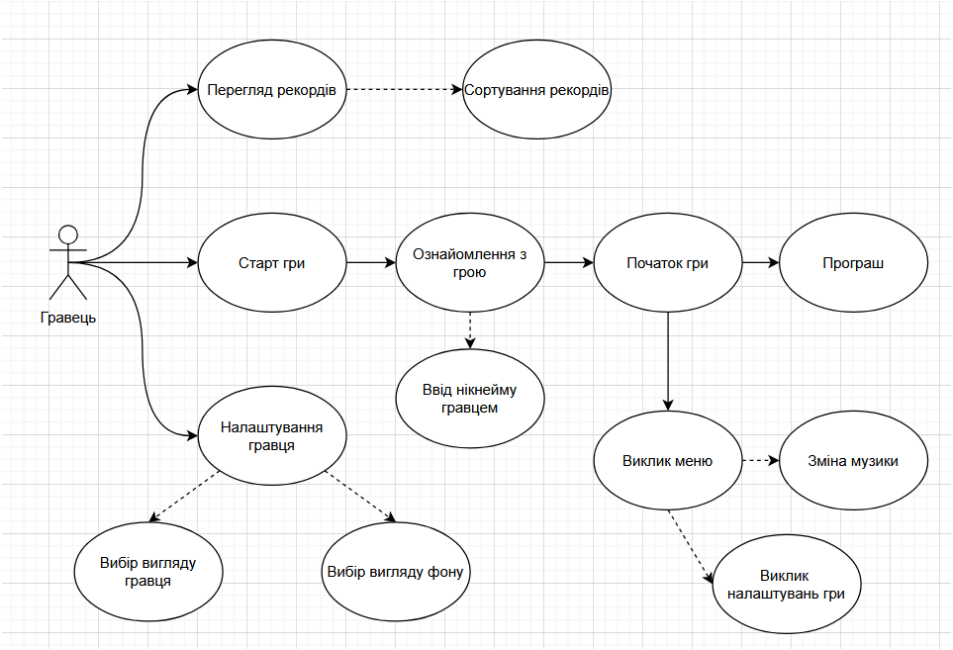


Рисунок А.1 – Діаграма варіантів використання для гравця «Move or Die»

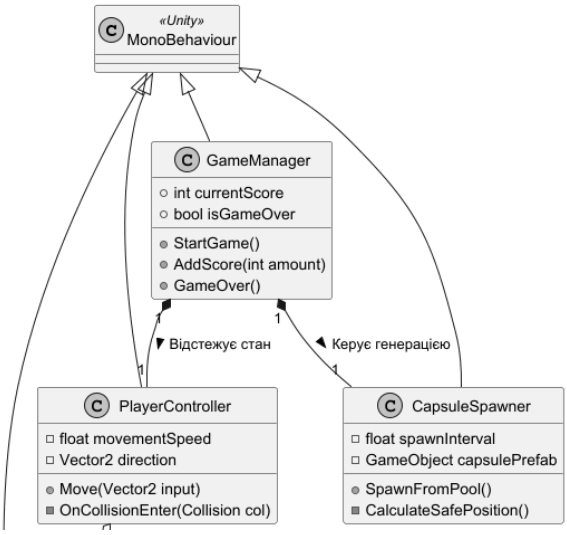


Рисунок А.2 – Загальна UML-діаграма класів застосунку

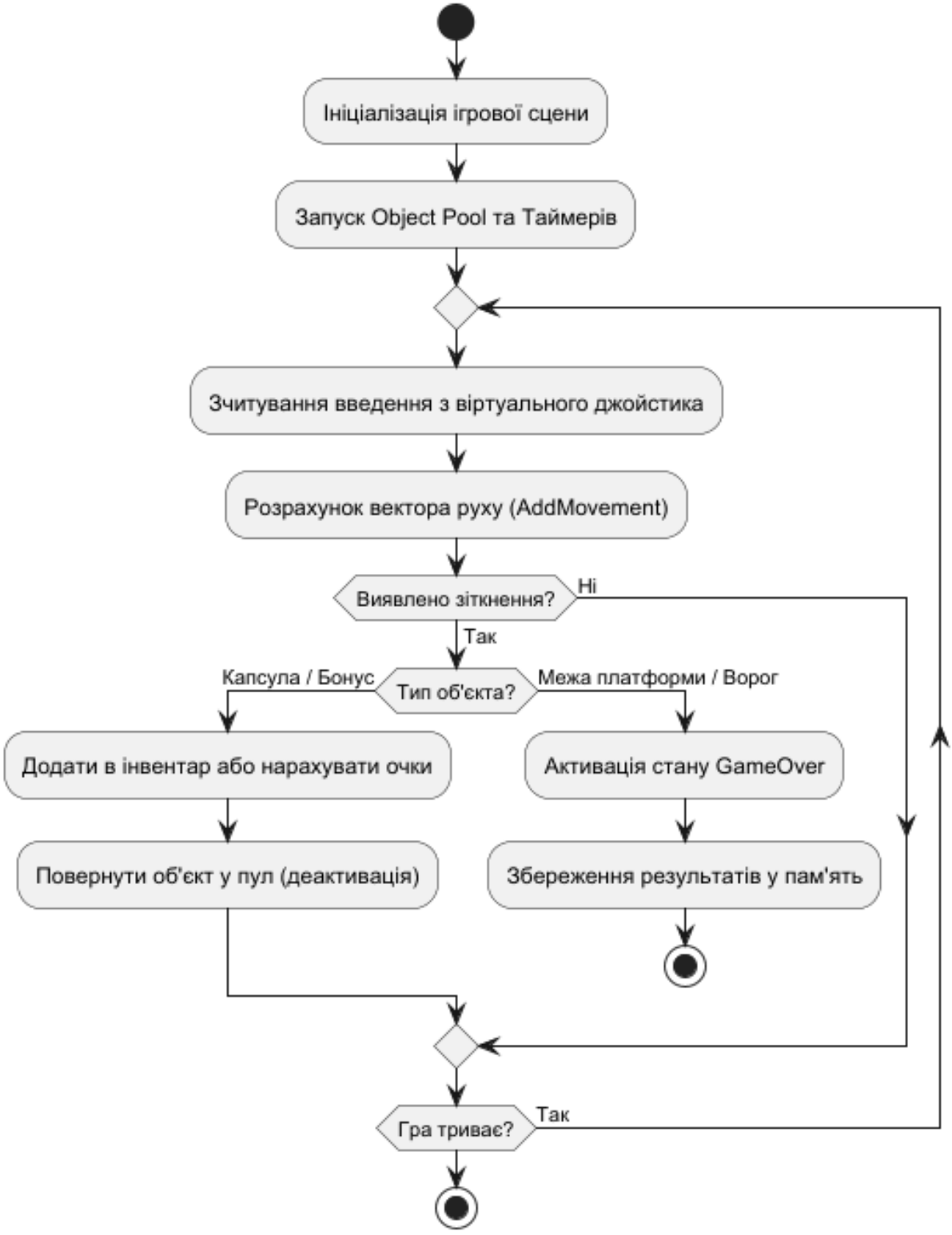


Рисунок А.3 – Блок-схема алгоритму генерації цільових об'єктів

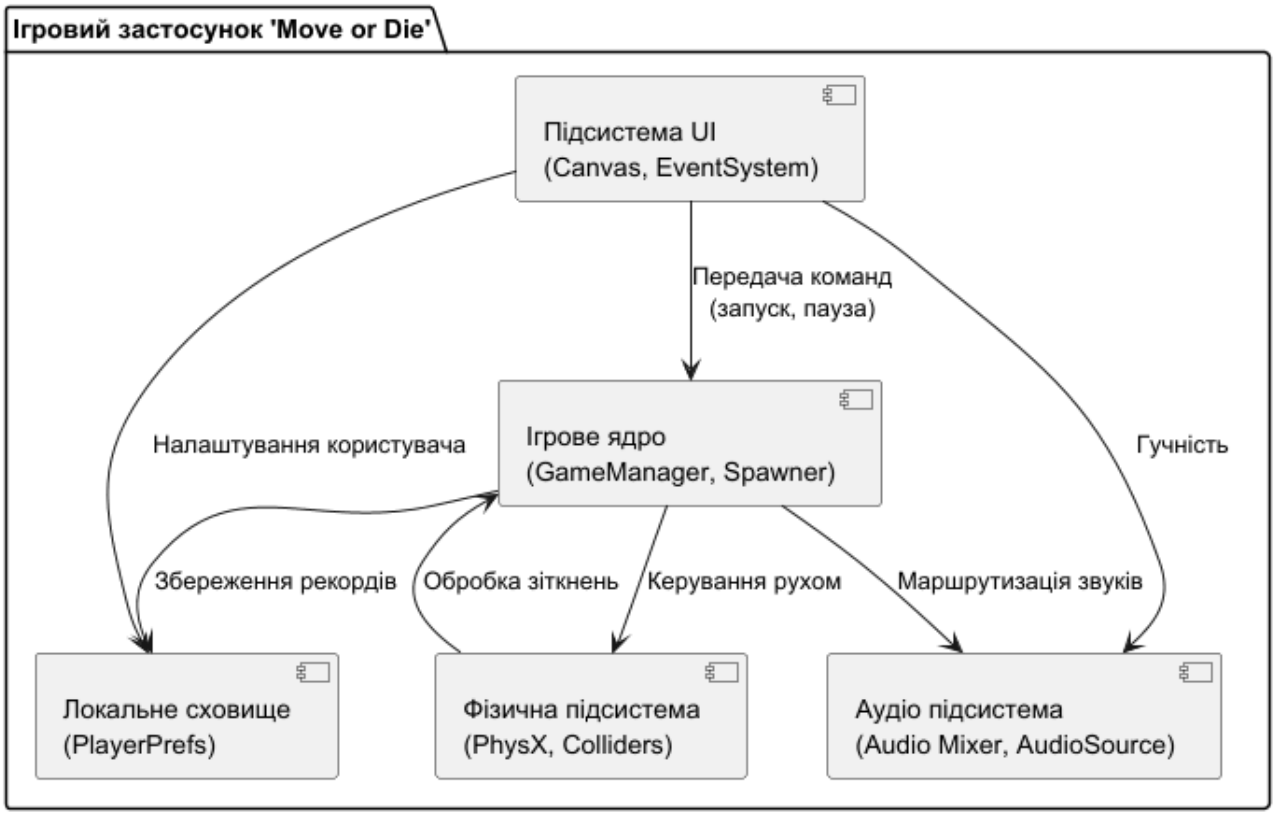


Рисунок А.4 – Загальна архітектура системи

Рисунок А.5 – Структура механізму збереження локальних даних

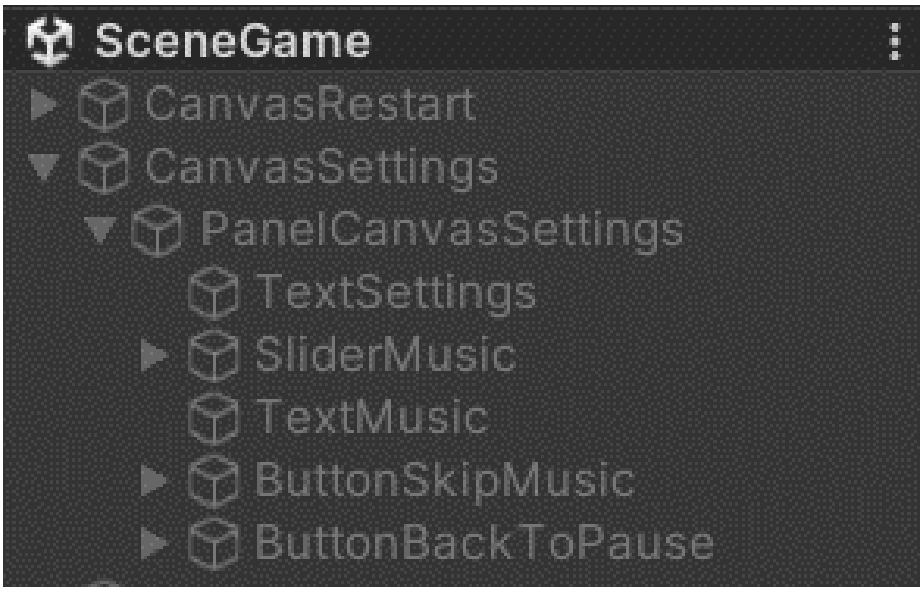


Рисунок А.6 – Ієрархія об'єктів підсистеми UI в інспекторі Unity

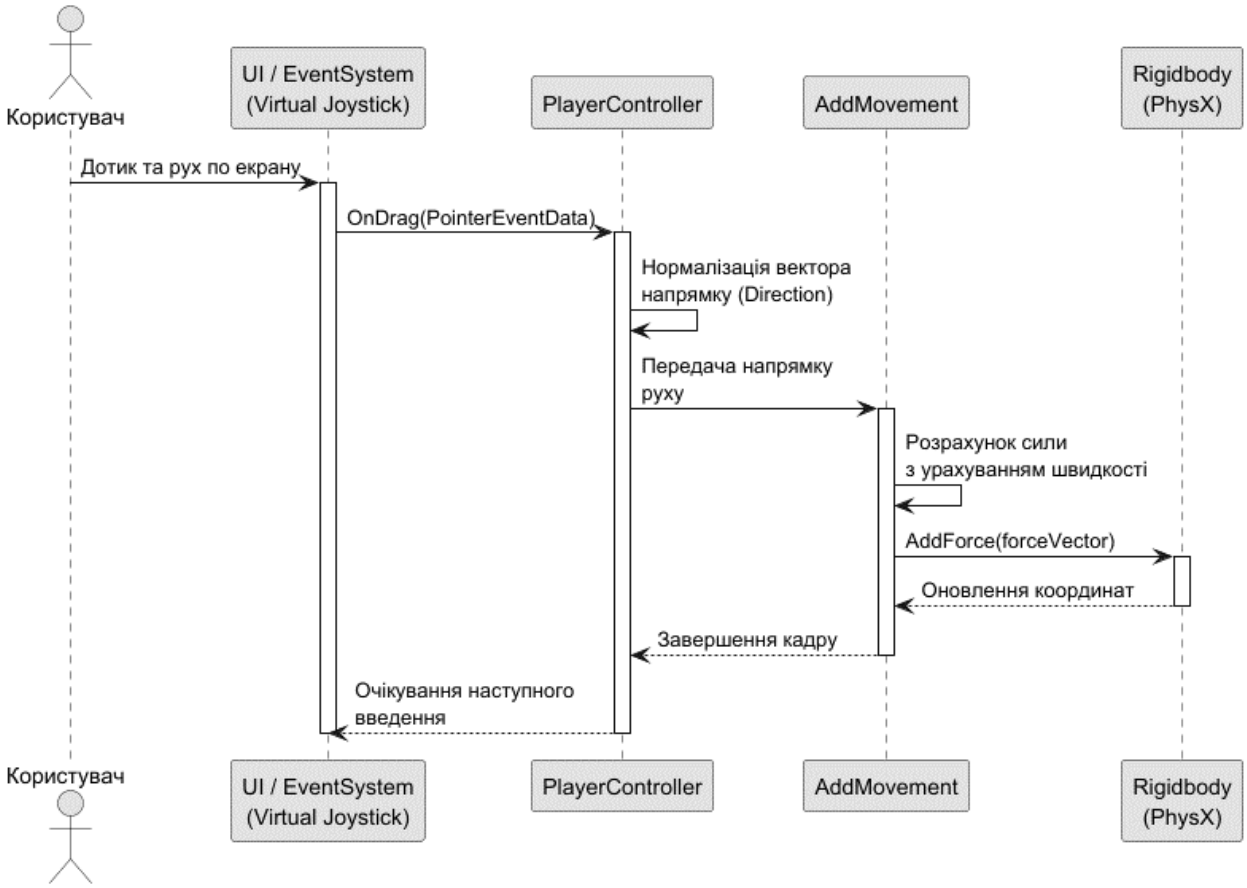


Рисунок А.7 – Процес обробки користувацького введення та передачі команд до ігрового ядра

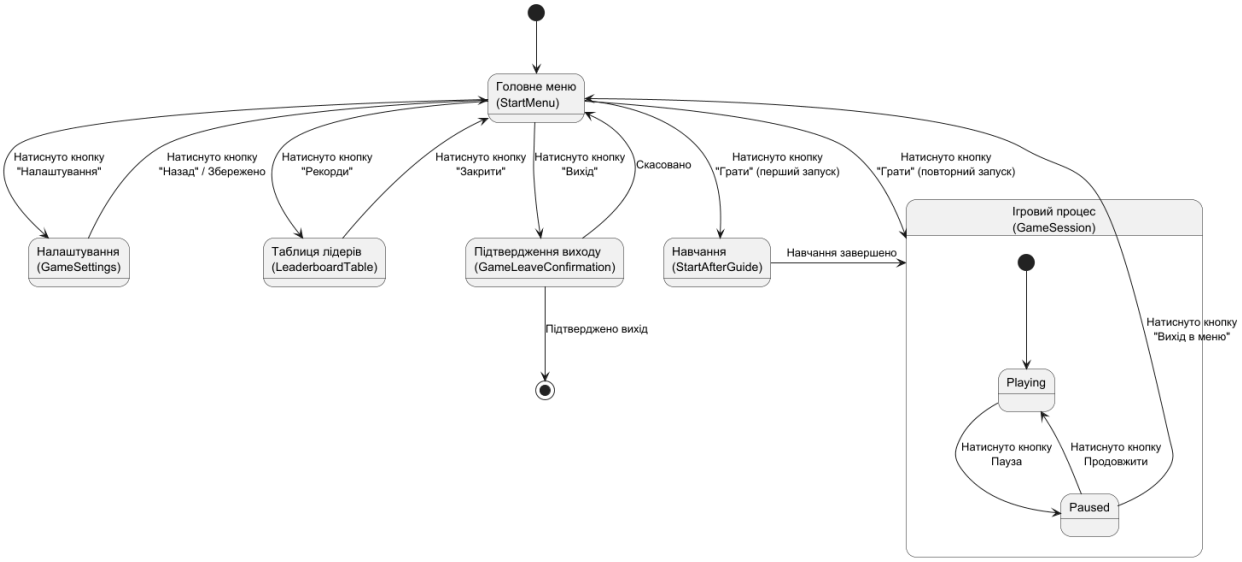


Рисунок А.8 – Переходи між екранами навігаційного меню застосунку

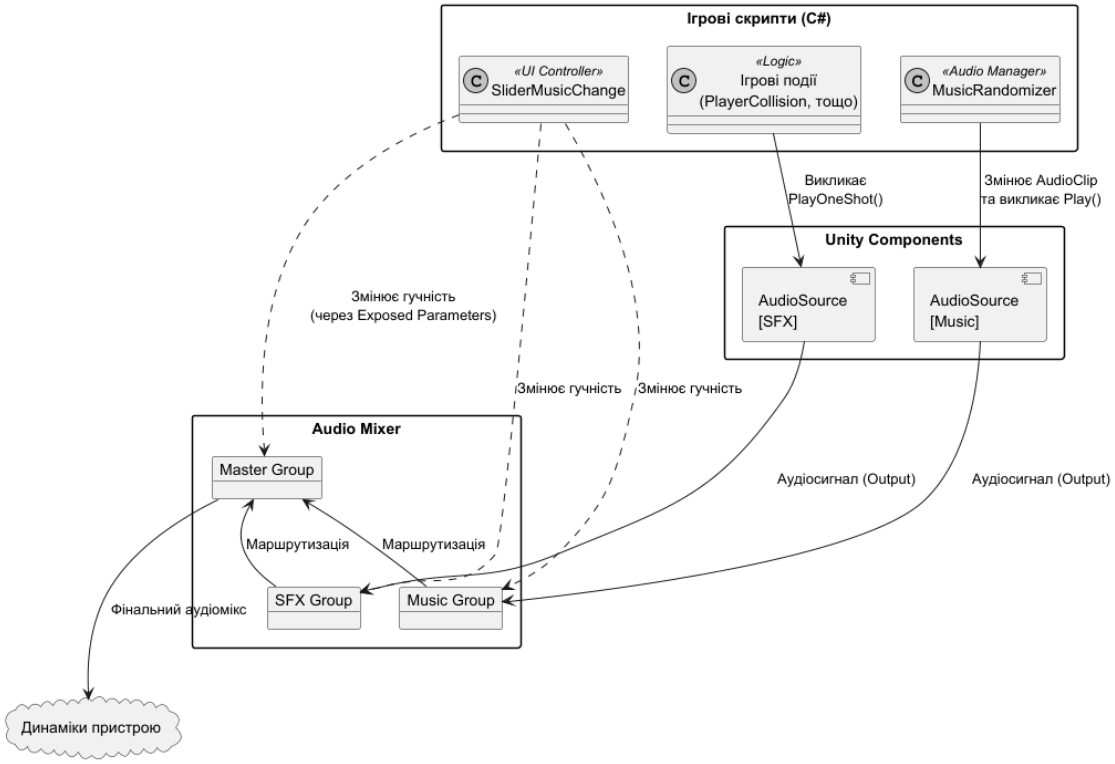


Рисунок А.9 – Схема маршрутизації аудіосигналів: Структура груп Audio Mixer та їхній зв'язок зі скриптами

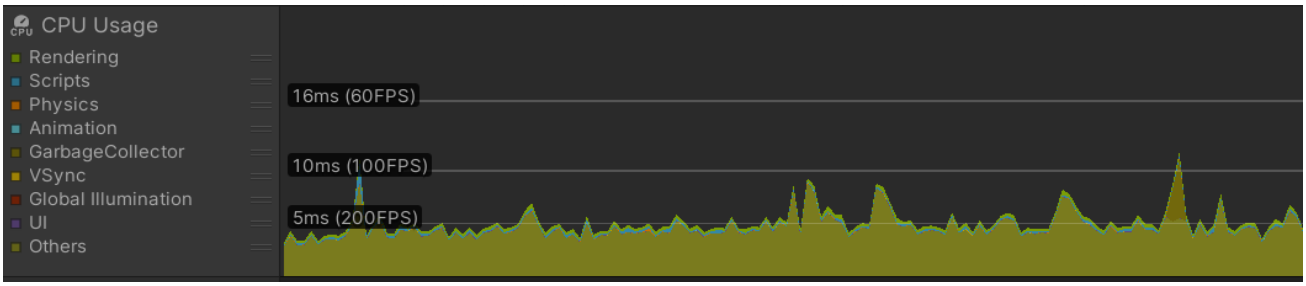


Рисунок А.10 – Графік CPU Usage у вікні Unity Profiler, що демонструє стабільний час обробки кадру під час ігрової сесії

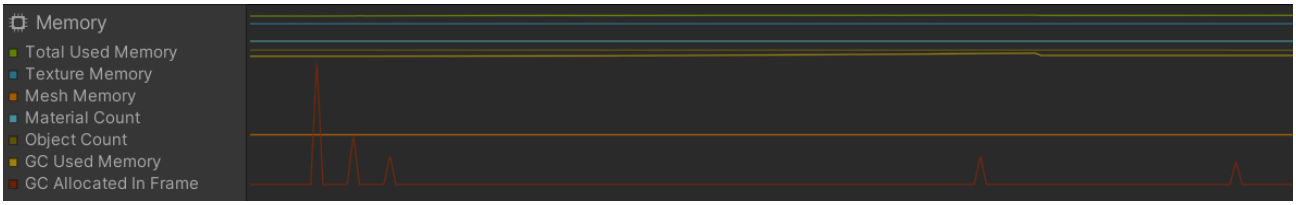


Рисунок А.11 – Графік Memory Allocation у Unity Profiler, що підтверджує ефективну роботу об'єктного пулу та відсутність витоків пам'яті

ДОДАТОК Б

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Б.1 Менеджер ігрової сесії

```
using System;
using UnityEngine;
using UnityEngine.SceneManagement;
using Random = UnityEngine.Random;

public class GameManager : MonoBehaviour
{
    public static GameManager GameManagerInstance;

    public ChangeMaterialWhenLevelUp changeMaterialWhenLevelUp;
    public GamePausing gamePausing;
    public ProgressBarController progressBarController;
    public PlayerCollision playerCollision;
    public ToastManager toastManager;

    private bool isPressed = false;

    public int amountOfCapsules;
    public int amountToNextLevel;
    public int currentLevel;
    public float basetimeToSpawn;
    public float newLevelTimeAddition;
    public int randomFloorGetStatically;

    public Canvas canvasSettings;
    public AudioSource audioSource;

    // Difficulty Settings
    public enum GameDifficulty { Easy = 0, Medium = 1, Hard = 2 }
    public GameDifficulty currentDifficulty;
    private float difficultyTimeMultiplier;

    void Awake()
    {
```

```
        if (GameManagerInstance == null)
        {
            GameManagerInstance = this;
        }
        Time.timeScale = 1;
        changeMaterialWhenLevelUp.ChangeFloor();
        audioSource.Play();
        progressBarController.SetFillAmount(amountToNextLevel);
    }

    private void Start()
    {
        audioSource.volume = PlayerPrefs.GetFloat("musicVolume", 1f);

        // Load difficulty (Defaults to Medium if not set yet)
        currentDifficulty = (GameDifficulty)PlayerPrefs.GetInt("GameDifficulty", 1);
        ApplyDifficultySettings();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape) &&
canvasSettings.isActiveAndEnabled == false)
        {
            gamePausing.TogglePause();
        }

        // Cheat code logic
        if (Input.GetKey(KeyCode.H) && Input.GetKeyDown(KeyCode.G))
        {
            isPressed = !isPressed;
            if (isPressed)
            {
                CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
+= 1000;
            }
            else
            {
                CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
-= 1000;
            }
        }
        WinCondition();
```

```
    }

    public void ApplyDifficultySettings()
    {
        switch (currentDifficulty)
        {
            case GameDifficulty.Easy:
                CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
= 7;
                difficultyTimeMultiplier = 0.50f;
                break;
            case GameDifficulty.Medium:
                CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
= 6;
                difficultyTimeMultiplier = 0.25f;
                break;
            case GameDifficulty.Hard:
                CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
= 5;
                difficultyTimeMultiplier = 0.10f;
                break;
        }
    }

    public void LevelCompletion()
    {
        progressBarController.ResetFilling();
        CapsuleSpawner.CapsuleSpawnerInstance.availableToSpawn = false;
        currentLevel += 1;
        randomFloorGetStatically += 1;
        amountToNextLevel += currentLevel + Random.Range(0, 3);
        RandomproofMaterialChanger();

        if (currentLevel % 5 == 1)
        {
            CapsuleSpawner.CapsuleSpawnerInstance.maxAmountOfCapsulesOnField
+= 1;
        }

        progressBarController.SetFillAmount(amountToNextLevel);
        if (playerCollision.backupCapsulesIfOverflow > 0)
        {
            for (int amountOfOverflow = 0; amountOfOverflow <
```

```
playerCollision.backupCapsulesIfOverflow; amountOfOverflow++)
            {
                progressBarController.CollectItem();
            }
        }
        NextLevel();
    }

    public void RandomproofMaterialChanger()
    {
        var randomFloorGetRandomly = Random.Range(0, 4);
        if (randomFloorGetRandomly == 0)
        {
            changeMaterialWhenLevelUp.ChangeFloor();
            randomFloorGetStatically = 0;
        }
        if (randomFloorGetStatically == 5)
        {
            changeMaterialWhenLevelUp.ChangeFloor();
            randomFloorGetStatically = 0;
        }
    }

    public void WinCondition()
    {
        if (playerCollision.capsulesCollected >= amountToNextLevel)
        {
            LevelCompletion();
            if (!ToastManager.instance.gameObject.activeInHierarchy)
            {
                ToastManager.instance.gameObject.SetActive(true);
            }
            ToastManager.instance.ShowToast("Рівень пройдено!");
        }
    }

    public void NextLevel()
    {
        newLevelTimeAddition += difficultyTimeMultiplier * (currentLevel - 1);
        CapsuleSpawner.CapsuleSpawnerInstance.availableToSpawn = true;
        CapsuleSpawner.CapsuleSpawnerInstance.timeToSpawn = basetimeToSpawn;
        PlayerCollision.PlayerCollisionInstance.capsulesCollected = 0;
        PlayerCollision.PlayerCollisionInstance.capsulesCollected +=
```

```
            PlayerCollision.PlayerCollisionInstance.backupCapsulesIfOverflow;
        PlayerCollision.PlayerCollisionInstance.backupCapsulesIfOverflow = 0;
    }
}
```

Б.2 Спавнер капсул

```
using System;
using System.Collections;
using System.Threading;
using Unity.VisualScripting;
using UnityEngine;
using Random = UnityEngine.Random;

public class CapsuleSpawner : MonoBehaviour
{
    public static CapsuleSpawner CapsuleSpawnerInstance;

    public GameManager gameManager;
    public GetCharacterPosition getCharacterPosition;
    public PlatformManipulation platformManipulation;
    public GameRestart gameRestart;

    public int amountOfCapsulesNow;
    public int maxAmountOfCapsulesOnField;
    public float timeToSpawn;
    public float borderLimitFromSpawningFromWalls;
    public float borderLimitFromSpawningAroundCharacter;
    public bool availableToSpawn;
    public int determinationOfCapsuleType;
```

```
    public float removeTimeToSpawnNormal;
    public float removeTimeToSpawnBonusTime;
    public float removeBonusTime;

    public GameObject[] objectsToSpawn;
    public Vector3 sizeOfPlatform;
    public Vector3 randomPositionOnPlatform;
    public Vector3 randomCapsuleSpawnPosition;



    void Awake()
    {
        if (CapsuleSpawnerInstance == null)
        {
            CapsuleSpawnerInstance = this;
        }
        StartCoroutine(TimedSpawner());
        sizeOfPlatform = platformManipulation.GetPlatformSize();
    }


    void Update()
    {
        if (Input.GetKeyDown(KeyCode.P))
        {
            CapsuleSpawn();
        }
        if (GameObject.Find("Capsule(Clone)") == null && GameObject.Find("Rare
Capsule(Clone)") == null)
        {
```

```
            if (timeToSpawn > 1f)
            {
                CapsuleSpawn();
                timeToSpawn -= removeTimeToSpawnBonusTime;
            }
            else
            {
                CapsuleSpawn();
            }
        }
    }

    IEnumerator TimedSpawner()
    {
        while (availableToSpawn)
        {
            CapsuleSpawn();
            yield return new WaitForSeconds(timeToSpawn);
            if (gameManager.newLevelTimeAddition > 0f)
            {
                gameManager.newLevelTimeAddition -= removeBonusTime;
                gameManager.newLevelTimeAddition =
(float)Math.Round(gameManager.newLevelTimeAddition, 3);
                timeToSpawn -= removeTimeToSpawnBonusTime;
                timeToSpawn = (float)Math.Round(timeToSpawn, 3);
            }
            else
            {
                if (timeToSpawn >= 1f)
```

```
            {
                timeToSpawn -= removeTimeToSpawnNormal;
            }
        }
    }
}

void CapsuleSpawn()
{
    if (amountOfCapsulesNow >= maxAmountOfCapsulesOnField)
    {
        gameRestart.Restart();
    }
    else
    {
        determinationOfCapsuleType = Random.Range(0, 100);
        randomCapsuleSpawnPosition = GetRandomSpawnPosition();
        if (determinationOfCapsuleType % 20 == 0)
        {
            Instantiate(objectsToSpawn[2], randomCapsuleSpawnPosition,
Quaternion.identity); // spawn unique capsule in 1/20 probability
        }
        else if (determinationOfCapsuleType % 6 == 0)
        {
            Instantiate(objectsToSpawn[1], randomCapsuleSpawnPosition,
Quaternion.identity); // spawn rare capsule in ~1/6 probability
            amountOfCapsulesNow += 1;
        }
        else
        {
```

```
            Instantiate(objectsToSpawn[0], randomCapsuleSpawnPosition,
Quaternion.identity); // spawn regular capsule
            amountOfCapsulesNow += 1;
        }
    }
}

Vector3 GetRandomSpawnPosition()
{
    var characterPosition = getCharacterPosition.GetCharacterCurrentPosition();
    var platformPosition = platformManipulation.GetPlatformPosition();
    do
    {
        randomPositionOnPlatform = new Vector3(
            Random.Range(platformPosition.x - sizeOfPlatform.x / 2 +
borderLimitFromSpawningFromWalls,
                platformPosition.x + sizeOfPlatform.x / 2 -
borderLimitFromSpawningFromWalls + 1),
            platformPosition.y + 2,
            Random.Range(platformPosition.z - sizeOfPlatform.z / 2 +
borderLimitFromSpawningFromWalls,
                platformPosition.z + sizeOfPlatform.z / 2 -
borderLimitFromSpawningFromWalls + 1));
    } while (IsWithinExclusionArea(randomPositionOnPlatform,
characterPosition));
    return randomPositionOnPlatform;
}

bool IsWithinExclusionArea(Vector3 capsulePosition, Vector3
characterPositionForExclusion)
{
```

```
        return Mathf.Abs(capsulePosition.x - characterPositionForExclusion.x) <=
borderLimitFromSpawningAroundCharacter &&
                Mathf.Abs(capsulePosition.z - characterPositionForExclusion.z) <=
borderLimitFromSpawningAroundCharacter;
    }
}
```

Б.3 Керування персонажом

```
using System;
using UnityEngine;
using UnityEngine.InputSystem;

public class AddMovement : MonoBehaviour
{
    public Rigidbody rb;
    public float forwardForce;

    public DefaultInputActions defaultInputActions;
    void FixedUpdate()
    {
        Vector2 inputedButtons =
defaultInputActions.Player.Move.ReadValue<Vector2>();
        Vector3 movement = new Vector3(inputedButtons.x, 0, inputedButtons.y) *
forwardForce;
        Time.timeScale = 1;
        rb.velocity = movement;
        // rb.AddForce(movement, ForceMode.VelocityChange);
    }
```

```
    private void Awake()
    {
        defaultInputActions = new DefaultInputActions();
    }

    private void OnEnable()
    {
        defaultInputActions.Enable();
    }

    private void OnDisable()
    {
        defaultInputActions.Disable();
    }
}
```

Б.4 Шкала прогресу рівня

```
using System;
using UnityEngine;

public class ProgressBarController : MonoBehaviour
{
    public Renderer[] targetRenderers;
    public float fillAmountNow;
    public float amountStepFill;

    private Material[] _progressBarMaterials;
    private static readonly int Fill = Shader.PropertyToID("_Fill");
```

```
void Start()
{
    // Initialize materials for all target renderers
    _progressBarMaterials = new Material[targetRenderers.Length];
    for (int i = 0; i < targetRenderers.Length; i++)
    {
        _progressBarMaterials[i] = targetRenderers[i].material;
    }
}

void Update()
{
    // Update the fill amount in all materials
    foreach (Material material in _progressBarMaterials)
    {
        material.SetFloat(Fill, fillAmountNow);
    }
}

public void SetFillAmount(float capsuleAmount)
{
    amountStepFill = (float)Math.Round(1 / capsuleAmount, 3);
}

public void CollectItem()
{
    // Increase fill amount when an item is collected
    fillAmountNow += amountStepFill;
```

```
        if (fillAmountNow > 1f) fillAmountNow = 1f;
    }

    public void OnItemCollected()
    {
        // Find all ProgressBarController instances in the scene and call CollectItem
        ProgressBarController[] progressBars =
FindObjectsOfType<ProgressBarController>();
        foreach (ProgressBarController progressBar in progressBars)
        {
            progressBar.CollectItem();
        }
    }

    public void ResetFilling()
    {
        fillAmountNow = 0f;
    }
}
```

Б.5 Рандомізатор музики на рівні

```
using System.Collections;
using System.Linq;
using UnityEngine;

public class MusicRandomizer : MonoBehaviour
{
    public static MusicRandomizer MusicRandomizerInstance;
```

```
    public PlatformManipulation platformManipulation;

    private static System.Random _rng = new System.Random();
    private Coroutine _playMusicCoroutine;
    public AudioClip[] musicList;
    public AudioClip specificMusicClip;
    public AudioSource musicToPlay;
    public AudioClip[] randomizedListOfMusic;
    public AudioClip currentAudioClip;

    public int currentNumberOfMusic;

    void Awake()
    {
        if (MusicRandomizerInstance == null)
        {
            MusicRandomizerInstance = this;
        }
        randomizedListOfMusic = musicList.OrderBy(audioClip =>
_rng.Next()).ToArray();
    }

    void Start()
    {
        if (_playMusicCoroutine == null)
        {
            _playMusicCoroutine = StartCoroutine(PlayMusic());
        }
```

```
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.K))
        {
            StopCurrentMusicAndPlayNext();
        }
    }
    IEnumerator PlayMusic()
    {
        while (true)
        {
            if (platformManipulation.videoPlayer.isPlaying)
            {
                platformManipulation.videoPlayer.Stop();
            }
            if (currentNumberOfMusic >= musicList.Length)
            {
                currentNumberOfMusic = 0;
                randomizedListOfMusic = musicList;
                randomizedListOfMusic = randomizedListOfMusic.OrderBy(audioClip =>
_rng.Next()).ToArray();
            }
            currentAudioClip = randomizedListOfMusic[currentNumberOfMusic];
            platformManipulation.CheckMusicAndPlayVideo();
            musicToPlay.PlayOneShot(currentAudioClip);
            currentNumberOfMusic += 1;
            yield return new WaitForSeconds(currentAudioClip.length);
```

```
            while (musicToPlay.isPlaying)
            {
                yield return null;
            }
        }
    }

    public void StopCurrentMusicAndPlayNext()
    {
        if (musicToPlay.isPlaying)
        {
            musicToPlay.Stop();
        }

        if (_playMusicCoroutine != null)
        {
            StopCoroutine(_playMusicCoroutine);
        }
        _playMusicCoroutine = StartCoroutine(PlayMusic());
    }
}
```

Б. 6 Функціонал колізії персонажа

```
using System;
using UnityEngine;
using Random = UnityEngine.Random;

public class PlayerCollision : MonoBehaviour
```

```
{
    public static PlayerCollision PlayerCollisionInstance;

    public AmountOfCapsulesCollectedText
amountOfCapsulesCollectedTextToChange;
    public GameManager gameManager;
    public ProgressBarController progressBarController;
    public UniqueCapsuleBonusMagnetism uniqueCapsuleBonusMagnetism;
    public UniqueCapsuleBonusInstapoints uniqueCapsuleBonusInstapoints;
    public UniqueCapsuleBonusBiggersize uniqueCapsuleBonusBiggersize;
    public UniqueCapsuleBonusMultiplier uniqueCapsuleBonusMultiplier;
    public UniqueCapsuleBonusInstalevelup uniqueCapsuleBonusInstalevelup;
    public UniqueCapsuleBonusSlowedTime uniqueCapsuleBonusSlowedTime;
    public CurrentBonusText currentBonusText;

    public int capsulesCollected;
    public int allCapsulesCollectedAmount;
    public int backupCapsulesIfOverflow;
    public int normalCapsulePoints;
    public int rareCapsulePoints;

    void Awake()
    {
        PlayerCollisionInstance = this;
    }

    void OnCollisionEnter(Collision collisionInfo)
    {
        if (collisionInfo.collider.name is "Capsule(Clone)")
```

```
        {
            capsulesCollected += normalCapsulePoints;
            progressBarController.OnItemCollected();
            allCapsulesCollectedAmount += normalCapsulePoints;
            CapsuleSpawner.CapsuleSpawnerInstance.amountOfCapsulesNow -= 1;
            Destroy(collisionInfo.collider.gameObject);
        }

        if (collisionInfo.collider.name is "Rare Capsule(Clone)")
        {
            capsulesCollected += rareCapsulePoints;
            for (int capsulePoints = 0; capsulePoints < rareCapsulePoints;
capsulePoints++)
            {
                progressBarController.OnItemCollected();
            }
            allCapsulesCollectedAmount += rareCapsulePoints;
            if (capsulesCollected > gameManager.amountToNextLevel)
            {
                backupCapsulesIfOverflow = capsulesCollected -
gameManager.amountToNextLevel;
            }
            CapsuleSpawner.CapsuleSpawnerInstance.amountOfCapsulesNow -= 1;
            Destroy(collisionInfo.collider.gameObject);
        }

        if (collisionInfo.collider.name is "Unique Capsule(Clone)")
        {
            int randomizedUniqueCapsuleBonus = Random.Range(0, 6);
            switch (randomizedUniqueCapsuleBonus)
```

```
        {
            case 0:
                uniqueCapsuleBonusInstapoints.AddPoints();
                if (capsulesCollected > gameManager.amountToNextLevel)
                {
                    backupCapsulesIfOverflow = capsulesCollected -
gameManager.amountToNextLevel;
                }

currentBonusText.AddBuff($"+{uniqueCapsuleBonusInstapoints.amountToAddUpo
nUniqueCapsule} очків", 3f, false);
                break;
            case 1:
                StartCoroutine(uniqueCapsuleBonusMagnetism.Magnetism());
                currentBonusText.AddBuff("Магнетизм",
uniqueCapsuleBonusMagnetism.bonusDurationTimeMagnetism);
                break;
            case 2:
                StartCoroutine(uniqueCapsuleBonusBiggersize.MakeCharacterBigger());
                currentBonusText.AddBuff("Збільшення персонажа",
uniqueCapsuleBonusBiggersize.bonusDurationTimeBiggersize);
                break;
            case 3:
                StartCoroutine(uniqueCapsuleBonusMultiplier.PointMultiplication());
                currentBonusText.AddBuff($"Мультиплікатор
x{uniqueCapsuleBonusMultiplier.multiplier}",
uniqueCapsuleBonusMultiplier.bonusDurationTimeMultiplier);
                break;
            case 4:
                uniqueCapsuleBonusInstalevelup.InstaLevelUp();
                currentBonusText.AddBuff("Моментальне підвищення рівня", 3f,
false);
```

```
                break;
            case 5:
                StartCoroutine(uniqueCapsuleBonusSlowedTime.SlowingTime());
                currentBonusText.AddBuff("Уповільнення часу",
uniqueCapsuleBonusSlowedTime.bonusDurationTimeSlowedtime);
                break;
        }
        Destroy(collisionInfo.collider.gameObject);
    }
  }
}
```

ДОДАТОК В

(обов’язковий)

КЕРІВНИЦТВО КОРИСТУВАЧА

Призначення програми: застосунок «Move or Die» – це мобільна динамічна 3D-аркада, розроблена для пристроїв під керуванням операційної системи Android. Головною метою користувача є керування ігровим персонажем (кубом) на ізометричній платформі, збір цільових об'єктів (капсул) для отримання балів та уникнення перешкод в умовах постійного зростання темпу гри.

Системні вимоги та встановлення: для коректної роботи застосунку мобільний пристрій повинен відповідати таким мінімальним системним вимогам:

Операційна система: Android 8.0 (Oreo) або новіша версія.

Оперативна пам’ять (RAM): 2 ГБ або більше.

Вільний простір на накопичувачі: 50 МБ.

Доступ до мережі Інтернет: не вимагається (повна підтримка офлайн-режиму).

Порядок встановлення:

- завантажте інсталяційний файл у форматі .apk на ваш мобільний пристрій;
- у налаштуваннях безпеки пристрою (Налаштування, далі Безпека або Біометрія та безпека) дозвольте встановлення додатків із невідомих джерел;
- знайдіть завантажений файл через файловий менеджер, натисніть на нього та дочекайтеся завершення процесу встановлення;
- запустіть застосунок за допомогою іконки «Move or Die», яка з'явиться на робочому екрані;

Інтерфейс головного меню: після запуску гри відкривається головне меню, яке містить інтуїтивно зрозумілі елементи керування:

- кнопка «Почати гру»: ініціює запуск нової ігрової сесії;
- кнопка «Рекорди»: відкриває панель конфігурації;
- кнопка «Вийти з гри»: безпечно закриває програму.

У розділі «Налаштування» користувач має можливість регулювати загальну гучність фонової музики (Music Volume) за допомогою повзунка.

Правила гри та керування:

- основна механіка: одразу після натискання кнопки «Грати» персонаж з'являється на ігровій платформі;
- гра не має логічного фіналу (Endless Runner) – сесія триває доти, доки гравець не припуститься помилки;
- на платформі процедурно з'являються цільові об'єкти (капсули)**;**
- кожна підібрана капсула додає бали до поточного рахунку.

Система керування: керування ігровим персонажем здійснюється за допомогою віртуального джойстика. Доторкніться пальцем до екрана в зоні керування (права нижня частина екрана). На місці дотику з'явиться візуальний інтерфейс джойстика.

Не відриваючи палець, потягніть його у бажаному напрямку. Персонаж миттєво почне рух у відповідну сторону (підтримується 360-градусний рух простором). Для зупинки персонажа достатньо відпустити палець від екрана.

Динаміка та складності: геймплей побудовано на постійному наростанні напруги:

- з кожною підібраною капсулою час до появи наступної зменшується;
- у разі досягнення певного порогу балів гра автоматично переводить гравця на наступний рівень (змінюється візуальне оформлення платформи, а базовий темп гри прискорюється).

Унікальні бонуси: окрім звичайних капсул, на платформі періодично з'являються бонусні об'єкти, які тимчасово полегшують ігровий процес:

- уповільнення часу (Slow Time): на кілька секунд загальний темп гри знижується, даючи час на прийняття рішень;
- магнетизм (Magnetism): дозволяє притягувати об'єкти до гравця на відстані без необхідності точного наближення до них.

Інтерфейс ігрової сесії (HUD): під час гри на екрані відображаються:

- поточний рахунок (у верхній частині екрана);
- кнопка паузи (у верхньому правому куті). Її натискання зупиняє ігровий час та відкриває меню з можливостями продовжити гру (Resume) або повернутися до головного меню (Main Menu).

Завершення ігрової сесії (Game Over): ігрова сесія переривається у випадку, якщо персонаж стикається з критичною перешкодою (ворожим об'єктом) або гравець не встигає підібрати капсулу за відведений таймером час (якщо ця механіка активована). Після цього на екрані з'являється вікно з поточним рахунком і кнопкою перезапуску рівня (Restart). Якщо поточний рахунок перевищує попередній рекорд, система автоматично збереже його.

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький національний університет
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота на тему:
Ігровий застосунок жанру аркади на рушії Unity “Move or Die”

Виконав:
Студент 4 курсу групи ІПЗ-22-1
Кашевар Костянтин Сергійович

Керівник:
Канд. пед. наук, доцент
Онишко Оксана Григорівна

Рисунок В.1 – Слайд 1

Актуальність теми

Ринок мобільних ігор: Понад 50% світової ігрової індустрії припадає на мобільний сегмент з щорічним зростанням.

Попит на казуальність: Користувачі віддають перевагу іграм з короткою ігровою сесією та низьким порогом входу.

Технічна складність: Вимога стабільної продуктивності (60 FPS) на широкому спектрі Android-пристроїв.

Оптимізація: Необхідність впровадження сучасних патернів проєктування для економії ресурсів батареї та RAM.

Рисунок В.2 – Слайд 2

Мета та завдання

Мета кваліфікаційної роботи:
Спроєктувати та програмно реалізувати оптимізований мобільний ігровий застосунок «Move or Die» жанру аркади, використовуючи двигун Unity з використанням сучасних архітектурних шаблонів та забезпеченням стабільної продуктивності.

Завдання для досягнення мети:
1.Провести аналіз предметної області, цільової аудиторії та існуючих програмних аналогів.
2.Спроєктувати компонентну архітектуру, UML-моделі, структуру збереження даних та макети інтерфейсу (UI).
3.Здійснити програмну реалізацію базових та поглиблених механік середовищі Unity (C#).
4.Провести комплексне тестування ПЗ (функціональне та профілювання продуктивності) з подальшим аналізом результатів.

Рисунок В.3 – Слайд 3

Аналіз предметної області

Аркадний жанр

Жанр характеризується швидким геймплеєм, інтуїтивним керуванням та зростаючою складністю. Основний фокус — утримання уваги гравця (retention).

- Психологія: миттєва винагорода.
- Цільова аудиторія: масовий споживач.

Рисунок В.4 – Слайд 4

Аналіз аналогів ПЗ

Назва гри	Тип управління	Графічний стиль	Механіка складності	Основні недоліки
Color Bump 3D	Свайпи вліво/вправо (автоматичний рух вперед)	Мінімалістичний 3D	Статична складність рівнів, відсутність таймерів	Обмежений контроль, монотонність геймплею
PAC-MAN 256	Свайпи у 4-х напрямках	Ізометричний воксельний 3D	Наростання швидкості переслідувачів (ворогів)	Перевантажений інтерфейс, обмежена плавність рухів
Cube Surfer	Свайпи по горизонталі	Яскравий 3D	Збільшення кількості перешкод	Лінійність рівнів, рух лише по прямій
Move or Die	Віртуальний джойстик (повний контроль)	Ізометричний 2D, яскраві контрастні кольори	Динамічне зменшення таймеру появи капсул	Знаходиться на стадії розробки

Рисунок В.5 – Слайд 5

Вимоги до програмного забезпечення

Функціональні

- Точне керування джойстиком.
- Генерація бонусів за ймовірністю.
- Збереження рекордів PlayerPrefs.

Нефункціональні

- Стабільна частота 60 FPS.
- Енергоефективність (Battery life).
- Час завантаження сцени < 3с.

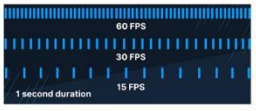



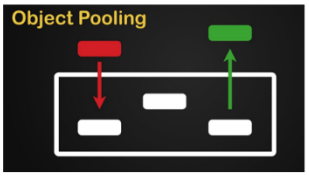

Рисунок В.6 – Слайд 6

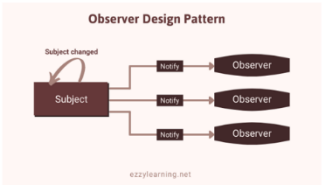
Архітектура та патерни

Компонентна модель

Використання вбудованої архітектури Unity для незалежності модулів. Обмін даними через Singleton та подійну модель.

Object Pool: Економія RAM.
Observer: Реактивний UI.
State Machine: Логіка станів гри.





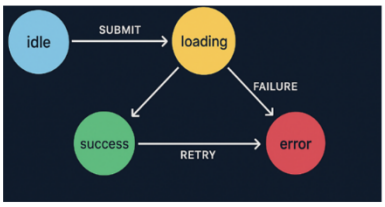


Рисунок В.7 – Слайд 7

Декомпозиція та інтерфейси

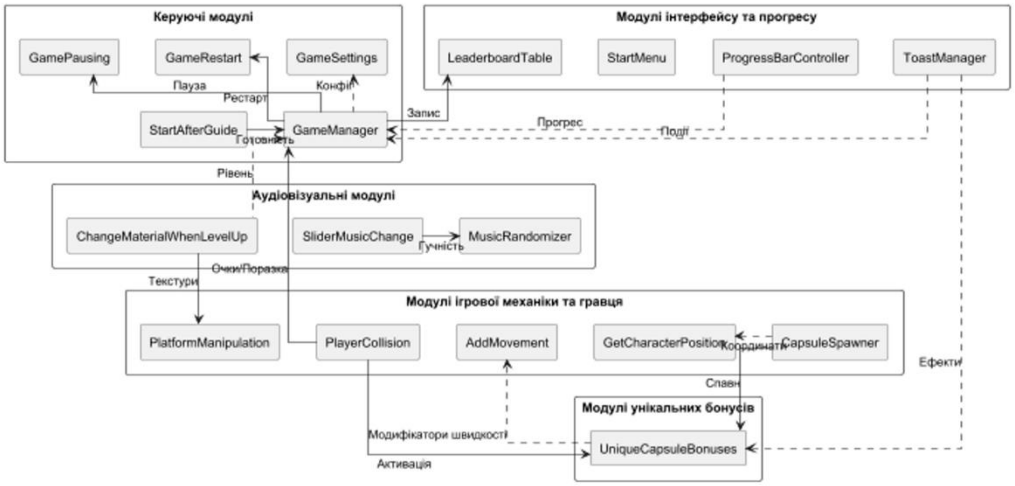
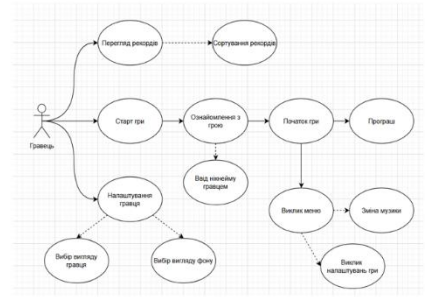


Рисунок В.8 – Слайд 8

Проєктування модулів і даних



Збереження даних

Використання ключової системи PlayerPrefs для локального зберігання прогресу:

- **BestScore:** int (рекорд очок).
- **VolumeConfig:** float (рівень звуку).
- **TutorialFlags:** bool (стан навчання).

Рисунок В.9 – Слайд 9

Аналіз та вибір технологій

Unity LTS

C# Language

Rider IDE

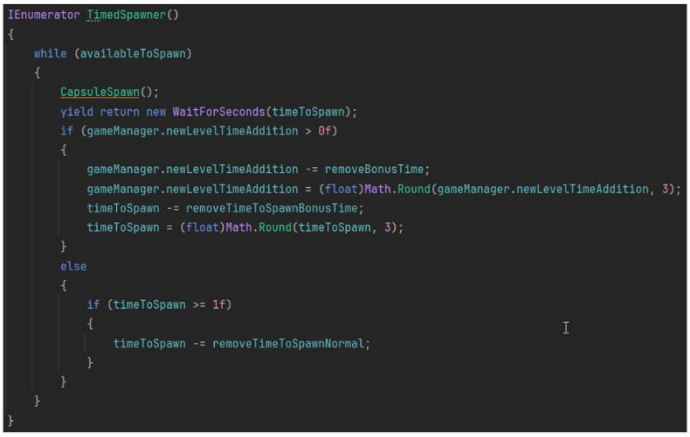
Git / GitHub

Аргументація вибору:

Кросплатформенність, потужна фізична підсистема PhysX, розвинена спільнота та низький поріг входу для мобільної розробки.

Рисунок В.10 – Слайд 10

Програмна реалізація



```
IEnumerator TimedSpawner()
{
    while (availableToSpawn)
    {
        CapsuleSpawn();
        yield return new WaitForSeconds(timeToSpawn);
        if (gameManager.newLevelTimeAddition > 0f)
        {
            gameManager.newLevelTimeAddition -= removeBonusTime;
            gameManager.newLevelTimeAddition = (float)Math.Round(gameManager.newLevelTimeAddition, 3);
            timeToSpawn -= removeTimeToSpawnBonusTime;
            timeToSpawn = (float)Math.Round(timeToSpawn, 3);
        }
        else
        {
            if (timeToSpawn >= 1f)
            {
                timeToSpawn -= removeTimeToSpawnNormal;
            }
        }
    }
}
```

```
public void ChangingMusicVolume(float val)
{
    _musicVolume = val;
    PlayerPrefs.SetFloat("musicVolume", _musicVolume);

    if (audioSource != null)
    {
        audioSource.volume = _musicVolume;
    }
}
```

```
if (PlayerPrefs.HasKey("musicVolume"))
{
    _musicVolume = PlayerPrefs.GetFloat(key: "musicVolume");
}
```

Рисунок В.11 – Слайд 11

Вимоги до технічних засобів

Характеристика	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Android 8.0 (Oreo)	Android 11.0 або новіша
Процесор (CPU)	4-ядерний, архітектура ARMv7 (32-bit)	8-ядерний, архітектура ARM64 (64-bit)
Оперативна пам'ять (RAM)	2 ГБ	4 ГБ або більше
Графічний прискорювач (GPU)	З підтримкою API OpenGL ES 3.0	З підтримкою Vulkan API або OpenGL ES 3.2
Вільний дисковий простір	50 МБ	100 МБ+ (внутрішній Flash-накопичувач)
Засоби введення	Сенсорний екран (Touchscreen)	Сенсорний екран із підтримкою Multi-touch
Доступ до мережі	Не вимагається (повний офлайн-режим)	Не вимагається

Рисунок В.12 – Слайд 12

Аналіз результатів тестування

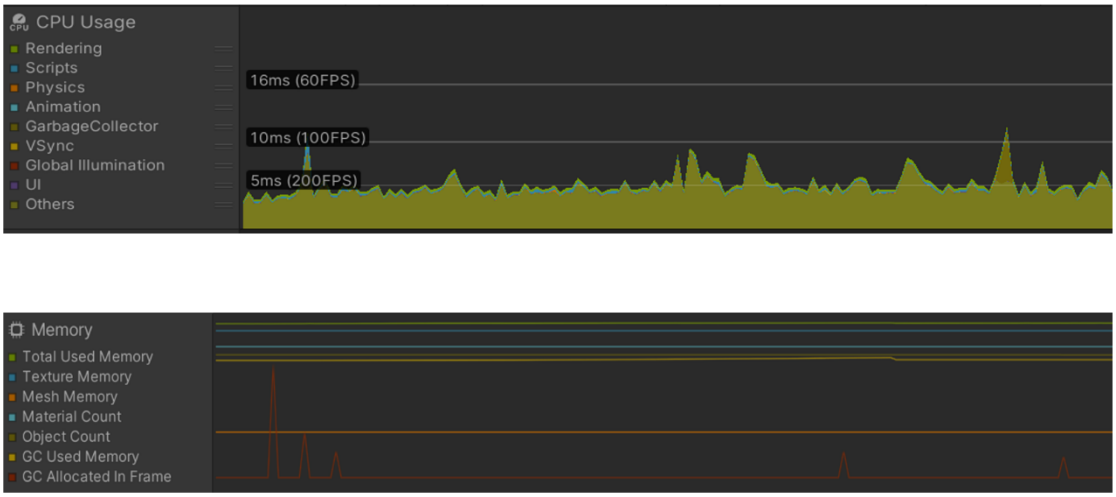


Рисунок В.13 – Слайд 13

Висновки та виконання завдань

Поставлене завдання	Статус / Результат
Аналіз предметної області	Виконано: виявлено основні тренди та аналоги.
Проєктування архітектури	Виконано: спроєктовано компонентно-подійну систему.
Реалізація ігрових механік	Виконано: розроблено Core Loop та систему бонусів.
Тестування ПЗ	Виконано: підтверджено 60 FPS та стабільність RAM.

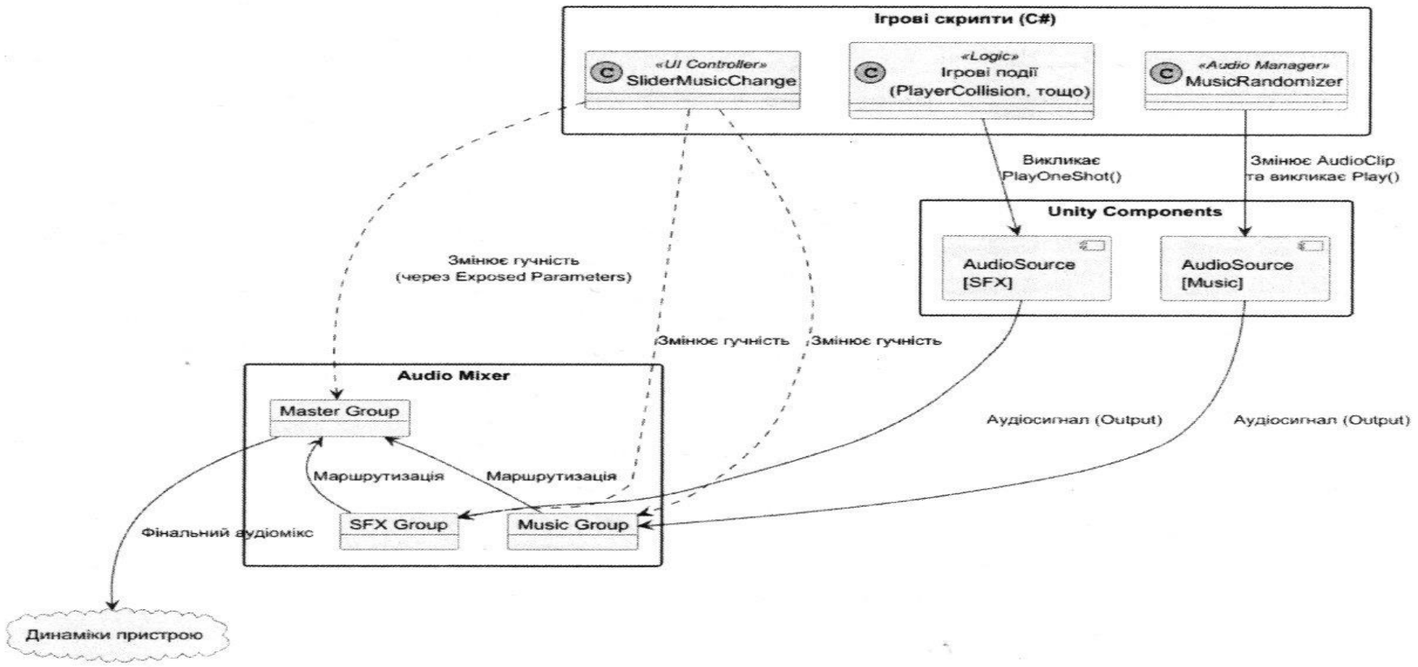
Висновки: Мета роботи досягнута. Розроблений застосунок «Move or Die» є завершеним програмним продуктом, що відповідає сучасним вимогам до мобільного ігрового ПЗ для мобільних операційних систем.

Рисунок В.14 – Слайд 14

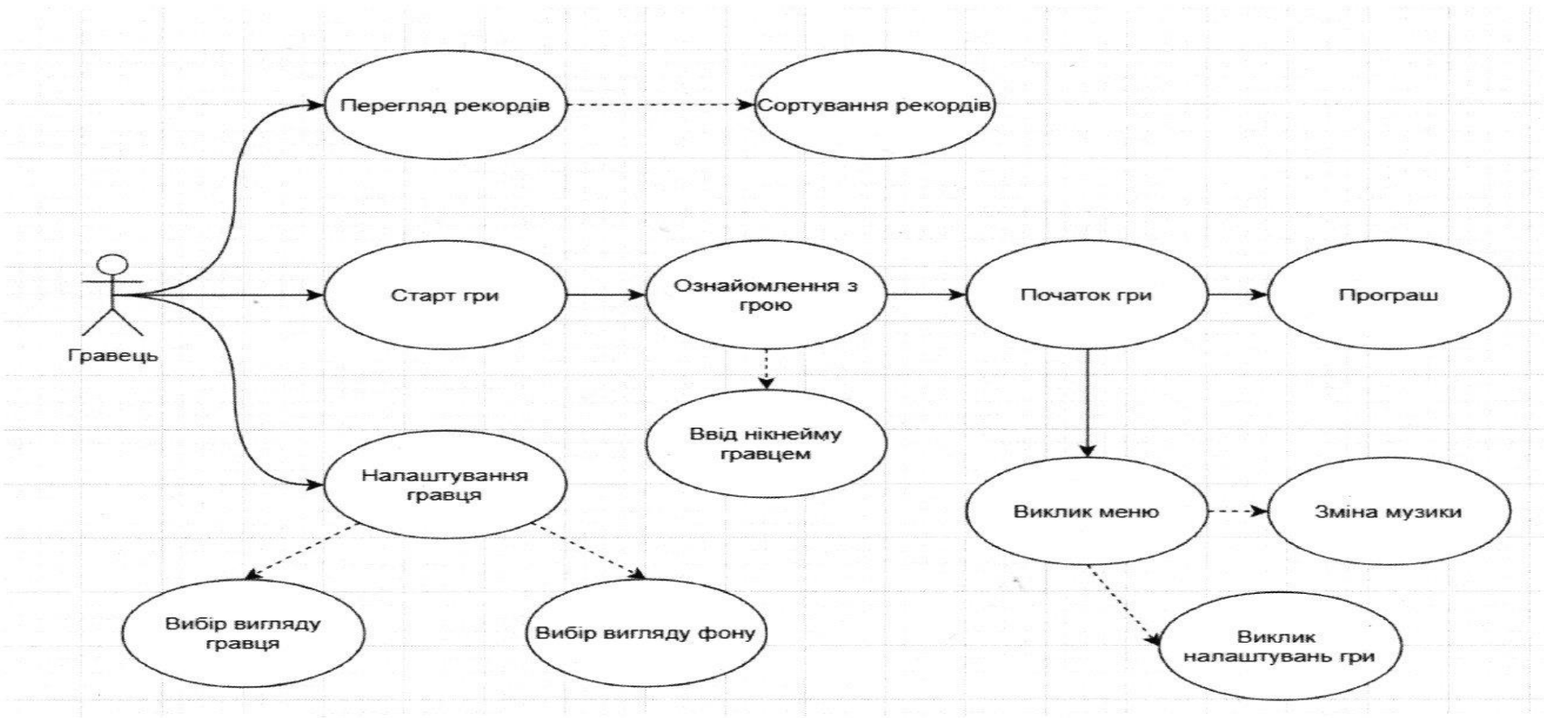
Дякую за вашу увагу та гарного дня!

Рисунок В.15 – Слайд 15

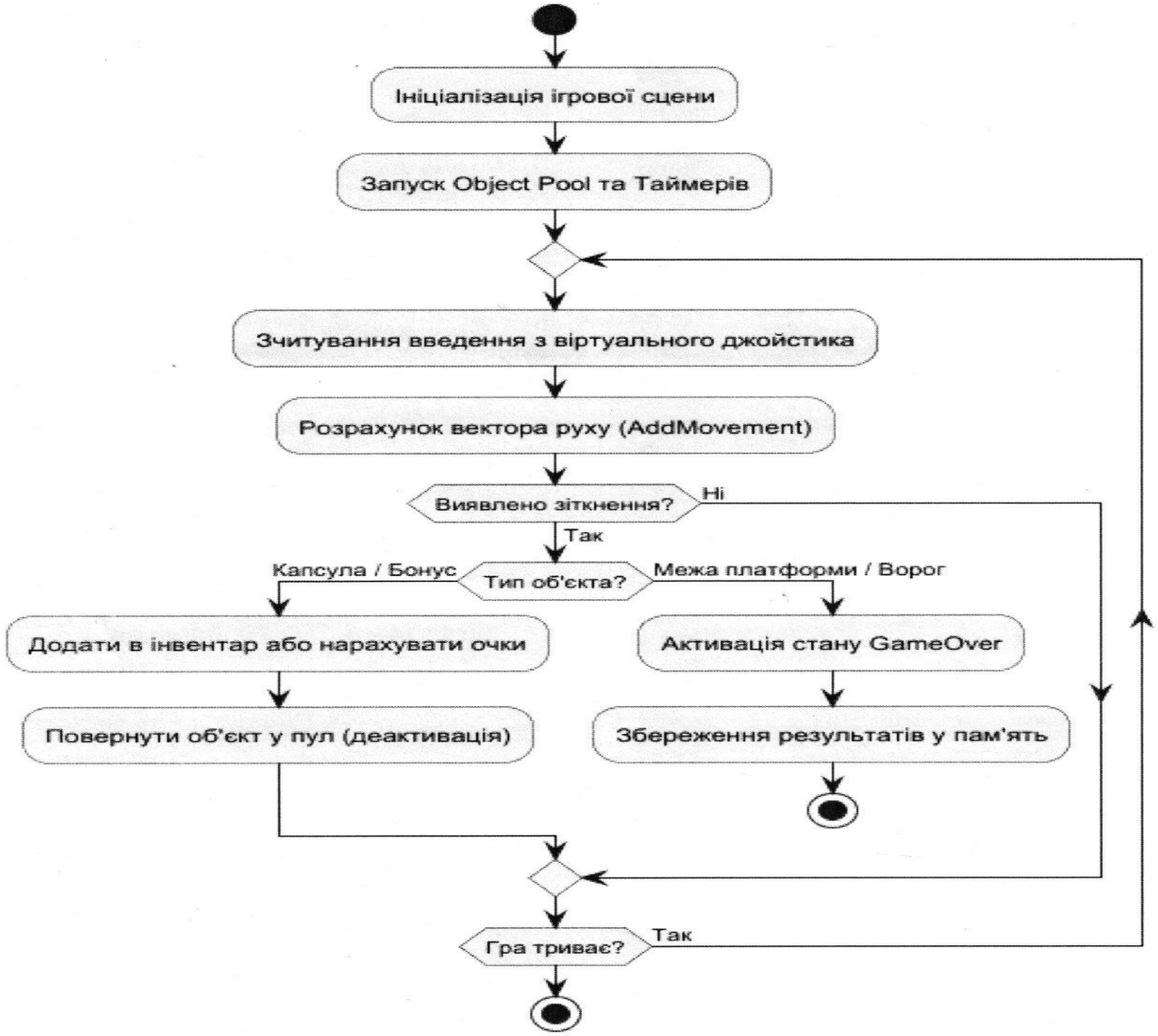
ГРАФІЧНІ МАТЕРІАЛИ



Змн.	Арк.	№ докум.	Підпис	Дат	КвРІПЗ.2301166.01.07.Е8		
Виконав		Кашевар К.С.		25.05	Схема маршрутизації аудіосигналів	Літ.	Арк. 1 / Аркушів 3
Керівник		Онишко О. Г.		25.05			
Рецензент						ХНУ, ІПЗ-22-1	
Н. контр.		Форкун Ю.В.		25.05			
Зав. каф.		Бедратюк Л.П.		25.05			



					КвРІПЗ.2301166.01.07.Е8	
Змн.	Арк.	№ докум.	Підпис	Дат		
Виконав		Кашевар К.С.		25.05	Діаграма варіантів використання	Літ. / Арк. 2 / Аркушів 3
Керівник		Онишко О. Г.		25.05		
Рецензент						ХНУ, ІПЗ-22-1
Н. контр.		Форкун Ю.В.		25.05		
Зав. каф.		Бедратюк Л.П.		25.05		



Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2301166.01.07.Е8		
Виконав		Кашевар К.С.		25.05	Блок-схема алгоритму генерації цільових об'єктів	Літ.	Арк. 3 / Аркушів 3
Керівник		Онишко О. Г.		25.05			
Рецензент						ХНУ, ІПЗ-22-1	
Н. контр.		Форкун Ю.В.		25.05			
Зав. каф.		Бедратюк Л.П.		25.05			

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Кашевара Костянтина Сергійовича
факультет ІТ, IV курс, 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.

21.05.26
дата

підпис

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: Костянтин КАШЕВАР

Співавтор:

Назва: Ігровий застосунок жанру аркади на рушії Unity “Move or Die”

Науковий керівник: канд. пед. наук, доцент Оксана ОНИШКО

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.63%

Коефіцієнт подібності 2:1.22%

Мікропробіли: 36

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-21 10:31:05.0

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 25.05.26 експерт (Бурцун Ю.В.)

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

Максимальне співпадіння з одним документом 12.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 14%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271874 Назва: БКР Ігровий застосунок жанру аркади на рушії Unity “Move or Die” Додано в БД: 2026-05-21 Автора: Костянтин КАШЕВАР Керівники: канд. пед. наук, доцент Оксана ОНИШКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	89349	610	14267 (16%)	124 (20%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269634	Назва: Переддипломна практика Додано в БД: 2026-03-02 Автора: К. С. Кашевар Керівники: Онишко О. Г., канд. пед. наук, доцент Консультанти: Опоненти:	10538 (12.0%)	79 (13.0%)

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Кашевар Костянтин Сергійович

Тема Ігровий застосунок жанру аркади на рушії Unity “Move or Die”

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 63

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено предметну область індустрії мобільного геймінгу, проведено ґрунтовний аналіз ринку аркадних ігор та існуючих аналогів. Визначено ключові функціональні та нефункціональні вимоги до створюваного ігрового застосунку. Обґрунтовано вибір ігрового рушія Unity та мови програмування C#. Спроєктовано гнучку архітектуру на базі компонентної моделі та сучасних патернів проєктування. Виконано програмну реалізацію ігрового ядра, процедурної генерації об'єктів та системи збереження даних. Проведено комплексне профілювання та тестування, яке підтвердило високу продуктивність, стабільність роботи та готовність розробленої мобільної гри «Move or Die» до релізу.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана в повному обсязі, повністю відповідає поставленому завданню тарозроблена з дотриманням усіх вимог до створення мобільного програмного забезпечення.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, визначено мету та завдання розробки. У першому розділі проаналізовано предметну область, розглянуто наявні рішення на ринку (аналоги), виявлено їхні архітектурні недоліки та сформовано вимоги до розроблюваної гри. У другому розділі обґрунтовано вибір технологій, спроєктовано архітектуру застосунку із застосуванням UML-моделювання та визначено оптимальні структури збереження даних (PlayerPrefs). У третьому розділі детально описано процес програмної реалізації модулів ігрової логіки, впровадження патернів (Singleton, Object Pool) для оптимізації продуктивності, а також наведено результати тестування, які доводять забезпечення стабільної частоти кадрів (60 FPS).

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є надзвичайно актуальною в умовах стрімкого розвитку ринку мобільного геймінгу. Головною перевагою проєкту є глибока технічна оптимізація: вдале використання архітектурного патерну Object Pool для уникнення витоків пам'яті та усунення

мікрозависань, викликаних роботою Garbage Collector. Також позитивно відзначається реалізація математичного алгоритму динамічного масштабування складності та зручне просторове управління за допомогою віртуального джойстика, що вигідно виділяє гру серед конкурентів.

5. Негативні сторони роботи У роботі збереження прогресу гравця (локальних рекордів) реалізовано виключно на рівні локальної пам'яті пристрою. У майбутньому варто додати інтеграцію з хмарними сервісами (наприклад, Google Play Games Services) для синхронізації результатів та створення глобальної таблиці лідерів для підвищення залученості гравців. Також можна розширити різноманітність процедурно генерованих перешкод.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано на високому рівні, повністю відповідає темі кваліфікаційної роботи. Архітектурні рішення наочно проілюстровано за допомогою UML-діаграм (діаграми варіантів використання, послідовності, станів) та скріншотів ігрового процесу. Пояснювальна записка структурована логічно та оформлена згідно з вимогами чинних стандартів (ДСТУ).

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є самостійним, логічно завершеним інженерним дослідженням і безумовно заслуговує на позитивну оцінку. Матеріал викладено структуровано, послідовно та технічно грамотно. Програмний продукт створено на високому рівні.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі, демонструє високий рівень підготовки здобувача та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Савенко Олег Станіславович, д.т.н., проф., проф. кафедри КІІС, ХНУ

"27" березня 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 жанру аркади на рушії Unity “Move or Die"»
Автор: Кашевар Костянтин Сергійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, кандидат педагогічних наук, доцент
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 у роботі, є законними і не є плагіатом, оскільки:
1. У тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2. Запозичення, виявлені у тексті роботи, є фрагментарними. Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 12.0% з одного джерела. Загальна сумарна подібність у базі даних складає 16% за символами та 20% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 3.63%, коефіцієнт подібності 2 – 1.22%. Виявлено 36 мікропробілів, не виявлено зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 28.05.26

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО