НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___"_____ 20 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Моделювання об'єктів в кросплатформеному середовищі розробки Unity

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

Старший викладач _____ Семко О. В.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Туровський В. М.

(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 20 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Туровському Владиславу Миколайовичу

(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи Моделювання об'єктів в кросплатформеному середовищі розробки Unity

затверджена наказом ректора НУБіП України від 08.04.2025 № 575 «С»

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи

_____

Перелік питань, які потрібно розробити:

Системний аналіз предметної області, Аналіз вимог до програмної системи, Моделювання структури та функціонування системи, Проєктування інформаційного та програмного забезпечення, Розробка логічної моделі даних та архітектури системи, Реалізація інформаційної бази та прикладного ПЗ, Тестування, впровадження та експлуатація системи.

Перелік графічних документів (за потреби)

Дата видачі завдання "03" квітня 2025 р.

Керівник бакалаврської кваліфікаційної роботи

Старший викладач _____ Семко О. В.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Туровський В. М.

(підпис) (ПІБ студента)

ЗМІСТ

Вступ

У сучасних умовах розвитку цифрових технологій все більша популярність сприяють інтерактивній візуалізації та симуляції тривимірних об'єктів. Їх застосовують у таких сферах, як архітектура, віртуальна та доповнена реальність, ігрова індустрія, промисловий дизайн, освіта та інші. Моделювання об'єктів у тривимірному середовищі дає змогу користувачам не тільки переглядати об'єкти, але й взаємодіяти з ними, оцінюючи їхні властивості, структуру та поведінку у змодельованому середовищі. Такий підхід суттєво розширює можливості представлення інформації та ефективність візуального сприйняття. Актуальність даного завдання виникає у потребі створення гнучких, адаптивних та реалістичних інструментів для моделювання об'єктів, які поєднують візуальні, фізичні та інтерактивні характеристики.

Метою розробки програмного додатку є створення інтерактивного 3D-середовища в ігровому рушії Unity, у якому змодельовані об'єкти володіють візуальними і фізичними властивостями, а користувач має змогу взаємодіяти з ними в режимі реального часу. Такий додаток може бути використаний як прототип для подальшої розробки візуалізацій архітектурних просторів, навчальних симуляцій або демонстраційних систем. Враховуючи широке застосування таких рішень, їхнє впровадження відповідає поточним вимогам ринку програмних продуктів та освітніх інструментів.

У процесі розробки програмного додатку застосовуються сучасні методи тривимірного моделювання, побудови сцени, застосування фізичних властивостей об'єктів – гравітація, зіткнення, матеріали. а також методи розробки інтерактивного інтерфейсу Основною технологією є Unity - потужне кросплатформене середовище розробки, що підтримує фізичні рушії, систему обробки подій, а також гнучкі інструменти для створення користувацьких сценаріїв. Для створення моделей можна використовувати сторонні

редактори, такі як Blender, з подальшим імпортом об'єктів у середовище Unity. У проекті також використовуються мова програмування C# для створення логіки взаємодії та обробки подій.

Основні завдання, які вирішуються у даній роботі, це аналіз предметної області, дослідження моделювання, візуалізації та симуляції об’єктів у інтерактивному середовищі. Проектування структури додатку, визначити типи об’єктів, їх властивості. Створення моделей об’єктів, реалізація фізичних властивостей об’єктів, розробка інтерактивних елементів.

Методи дослідження включають моделювання системи за допомогою UML-діаграм, аналіз літературних джерел та розробку прототипу.

У результаті дослідженої теми було створенно програмне забезпечення, яке відповідає стандартам та вивчено принципи побудови тривимірних об’єктів, особливості їх візуалізації, а також реалізації фізичної взаємодії між об’єктами. Проведено аналіз технологій, які дозволяють створювати інтерактивні сцени, обробки колізій, освітлення та управління камерою. Вивчено підходи до імпорту моделей із зовнішніх редакторів, таких як Blender, і інтеграції їх у середовище Unity. Здійснено тестування функціональності сцени та оцінено практичну придатність для використання в навчальних та демонстраційних цілях.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

1.1.1 Всуп до предметної області. У сучасному цифровому середовищі зростає потреба у створенні реалістичних віртуальних об'єктів та середовищ, які дозволяють користувачам не лише спостерігати, а й активно взаємодіяти з тривимірним контентом. Такий підхід є особливо важливим у

сфері візуалізації архітектурних просторів, створення навчальних симуляцій, розробки ігор, віртуальних турів, дизайну та інженерного моделювання. Моделювання об'єктів у тривимірному середовищі дозволяє відтворити як зовнішній вигляд елементів, так і їхню поведінку у фізичному просторі, що значно розширює можливості для їх використання.

Середовище розробки Unity займає провідні позиції серед платформ для створення інтерактивних 3D-додатків завдяки своїй гнучкості, підтримці фізичних рушіїв, інструментам роботи з анімацією, світлом і матеріалами. Предметна область даної роботи охоплює створення об'єктів з реалістичними геометричними, текстурними та фізичними характеристиками, їх розміщення у віртуальному просторі, а також забезпечення інтерактивної взаємодії з користувачем. Особлива увага приділяється використанню фізики, яка дозволяє досягти ефекту реалізму в поведінці об'єктів, таких як падіння, зіткнення, ковзання чи переміщення під дією сил.

1.1.2 Огляд поточного стану предметної області. На сьогодні моделювання тривимірних об'єктів є однією з ключових технологій у таких галузях, як комп'ютерні ігри, архітектурна візуалізація, віртуальна реальність, інженерне проєктування та освіта. Із розвитком апаратного забезпечення та програмних платформ розширюються можливості створення реалістичних 3D-сцен, симуляцій фізичних процесів та забезпечення інтерактивної взаємодії з об'єктами у віртуальному середовищі.

Одним із найпопулярніших інструментів для реалізації подібних рішень є Unity, багатофункціональний ігровий рушій, який підтримує розробку як для настільних, так і мобільних та VR/AR платформ. Unity активно використовується як в індустрії розваг, так і в наукових, навчальних і промислових проектах завдяки своїй відкритості, наявності великої спільноти, підтримці фізичного моделювання, систем анімації, освітлення, матеріалів та гнучкому C# орієнтованому API.

Популярними є також інструменти імпорту 3D-моделей, такі як Blender, 3ds Max, Maya приклад однієї із платформ показані на рис.1, що

дозволяють створювати складні об'єкти й передавати їх у Unity без втрати якості. Крім того, поширюються методи генеративного моделювання та інтеграції з AI/ML для динамічного створення контенту. Загалом, сучасний стан предметної області характеризується високим рівнем розвитку інструментальних засобів, широкою доступністю навчальних матеріалів, активною підтримкою спільнот і наявністю численних прикладів застосування моделювання об'єктів у Unity у різних сферах.

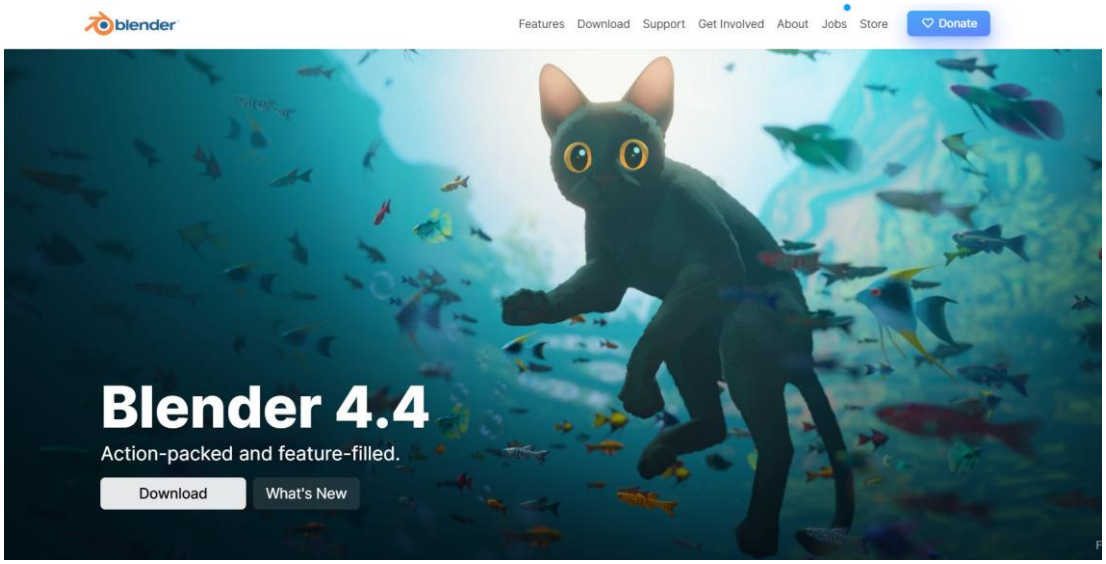

Рис1 Головна сторінка платформи Blender

Незважаючи на активний розвиток середовища розробки Unity та широке впровадження 3D-моделювання в різних галузях, предметна область все ще стикається з низкою проблем. Однією з основних є досить високий поріг входу для початківців: для ефективної роботи над повноцінним 3D-проєктом розробнику необхідно володіти знаннями з програмування, фізики, тривимірного моделювання та принципів побудови інтерактивної взаємодії. Це ускладнює швидке залучення нових спеціалістів або студентів до створення подібних систем.

Крім того, фізичний рушій Unity має певні обмеження: він забезпечує базову симуляцію фізики, але не придатний для складного моделювання рідин, тканин або деформацій, що обмежує реалізм у проєктах, які потребують високої точності. Також розробники часто стикаються з проблемами продуктивності при роботі з великими або деталізованими сценами, особливо

на пристроях з обмеженими ресурсами. Це вимагає додаткових знань в області оптимізації, зокрема налаштування рівнів деталізації, baking освітлення чи використання механізмів відсікання невидимих об'єктів (Occlusion Culling).

Ще однією складністю є процес інтеграції моделей, створених у сторонніх 3D-редакторах, таких як Blender чи 3ds Max. При імпорті часто виникають помилки зі зміною масштабу, втратою матеріалів або порушенням UV-розгортки. Водночас багато сучасних функцій Unity реалізовано лише у рамках певних графічних пайплайнів (URP або HDRP), що знижує гнучкість і вимагає уважного підбору версії рушія ще до початку розробки.

Також варто зазначити, що хоча Unity має власний магазин готових рішень - Asset Store — значна частина корисних ресурсів є платною, що створює додаткові обмеження для навчальних або некомерційних проєктів.

1.2 Аналіз Вимог до програмної системи

1.2.1 Функціональні вимоги. Опис основних функцій які повинна виконувати програма.

Функціональні вимоги визначають перелік основних дій і сценаріїв, які має забезпечувати програмна система для досягнення поставленої мети. У контексті моделювання об'єктів у середовищі Unity, система повинна виконувати такі функції:

- Візуалізація тривимірного середовища. Програма повинна відображати сцену з 3D-об’єктами, які мають задану геометрію, матеріали, освітлення та позицію у просторі. Всі об’єкти мають бути доступними для огляду в режимі реального часу.

- Можливість інтерактивної навігації сценою. Користувач має змогу переміщуватись сценою за допомогою камери (від першої або третьої особи) — ходити, оглядатися, наближати або віддаляти об’єкти.
- Інтерактивна взаємодія з об’єктами. Програма повинна реагувати на дії користувача — наприклад, при натисканні на об’єкт він може

відкриватися, переміщуватися або змінювати стан (відчинення дверей, активація освітлення тощо).

- Фізична симуляція поведінки об’єктів. Об’єкти мають взаємодіяти відповідно до фізичних законів — падати під дією сили тяжіння, відбиватись при зіткненні, ковзати, повертатись у початковий стан тощо. Це забезпечується використанням компонентів Rigidbody та Collider.
- Відтворення анімацій. Система повинна мати можливість запускати прості анімації для певних об'єктів, наприклад, анімоване відкривання дверей, рух механізмів чи реакція на дії користувача.
- Інформаційне супроводження об'єктів. Користувач має можливість отримати текстову або графічну інформацію про об’єкт при наведенні чи взаємодії з ним (наприклад, назва, опис, інструкція).
- Користувацький інтерфейс (UI). Програма повинна містити базовий інтерфейс — головне меню, кнопки виходу, підказки, індикатори стану або режимів, доступ до налаштувань (наприклад, режим переміщення або взаємодії).
- Імпорт 3D-об’єктів у сцену. Система повинна мати можливість підтримувати завантаження заздалегідь змодельованих 3D-об’єктів у форматах, сумісних із Unity (наприклад, .fbx, .obj).
- Реагування на події користувача. Обробка вводу з клавіатури та миші — переміщення, клацання, утримання клавіш, взаємодія з елементами сцени та інтерфейсу.

1.2.2 Опис конкретних сценаріїв використання.

Сценарії використання описують типові дії користувача, які відповідають основним функціональним можливостям системи. Вони визначають логіку взаємодії з програмним середовищем, включаючи ініціацію дій, умови виконання та очікуваний результат.

Сценарії використання (use cases) є невід'ємним елементом аналізу вимог до програмної системи, оскільки вони дозволяють детально описати взаємодію користувача з системою в рамках конкретних завдань. Сценарії демонструють, яким чином кінцевий користувач здійснює операції з 3D-об’єктами — наприклад, створення, редагування, збереження або візуалізація моделей.

Опис сценаріїв дозволяє структурувати функціональні вимоги до системи, визначити ролі користувачів (акторів) та встановити чіткі межі системи. Наприклад, типовими сценаріями є запуск додатку, завантаження сцени з об'єктами, зміна параметрів моделей, взаємодія з фізичними властивостями об’єктів (гравітація, колізії) або застосування анімаційних ефектів. Кожен сценарій включає послідовність кроків, які ілюструють логіку використання додатку з точки зору користувача.

Сценарії також допомагають визначити виняткові ситуації та вимоги до надійності — наприклад, як система поводиться при неправильному введенні параметрів або при спробі завантаження неіснуючого ресурсу. Це сприяє підвищенню якості системи та її зручності для кінцевих користувачів.

Основні елементи для побудови моделі прецедентів на діаграмі:

- Межа системи – прямокутник, який окреслює прецеденти для умовного позначення краю, або межі, модельованої системи. У UML 2 цю межу називають суб’єктом (контекстом або контекстним класифікатором) системи.
- Ектор (actor) – елемент, що позначає ролі користувача, який взаємодіє з компонентами системи;
- Прецедент – елемент, що відображає дії, а також варіанти дій, які виконує система, і які призводять до відповідних результатів.
- Сценарій (екземпляр прецеденту) – послідовність дій або взаємодій між виконавцем та системою, це конкретний випадок використання системи.

- Відношення – значущі відношення між акторами і прецедентами.

1. Сценарій 1: Огляд 3D-сцени

Актор: Користувач

Передумова: Програма запущена, сцена завантажена.

Опис:

- Користувач починає з фіксованої стартової позиції.
- За допомогою клавіш WASD (або стрілок) і миші користувач переміщується сценою.
- Камера слідує за поглядом користувача, дозволяючи огляд об'єктів у просторі.

Результат: Користувач вільно навігує середовищем, не змінюючи стан об’єктів.

2. Сценарій 2: Взаємодія з об'єктом

Актор: Користувач

Передумова: Користувач знаходиться поруч з об’єктом, що підтримує взаємодію.

Опис:

- Користувач наближається до інтерактивного об’єкта (наприклад, двері).
- Програма підказує (текстовим повідомленням або іконкою), що можливе натискання клавіші для взаємодії (наприклад, клавіша E).
- Користувач натискає клавішу.
- Програма виконує дію — наприклад, анімацію відкриття дверей або активацію механізму.

Результат: Стан об’єкта змінено, взаємодія відбулася.

3. Сценарій 3: Переміщення об’єкта в просторі

Актор: Користувач

Передумова: Користувач у режимі взаємодії, активований дозволений об’єкт.

Опис:

- Користувач підходить до предмета (наприклад, ящика).
- З’являється інструкція: “Утримуйте LMB для переміщення”.
- Користувач натискає і утримує ліву кнопку миші.
- Об’єкт слідує за рухами миші у межах дозволеного простору.

Результат: Об’єкт змінює позицію згідно з діями користувача.

4. Сценарій 4: Завершення роботи з програмою

Актор: Користувач

Передумова: Користувач знаходиться в головному інтерфейсі або в сцені.

Опис:

- Користувач натискає клавішу Esc або кнопку “Вихід” у меню.
- Програма виводить запит на підтвердження завершення роботи.
- Користувач підтверджує вихід.

Результат: Додаток закривається, або повертається до головного меню.

1.2.3 Нефункціональні вимоги. Нефункціональні вимоги визначають якісні характеристики програмної системи, які забезпечують її стабільну, безпечну та ефективну роботу в межах заданого середовища. Вони охоплюють такі аспекти:

1. Продуктивність системи

- Програма повинна працювати із стабільною частотою кадрів (не менше 30 FPS) на середньостатистичних комп’ютерах.
- Оптимізація 3D-моделей, текстур та освітлення має бути проведена з урахуванням зменшення навантаження на GPU та CPU.

2. Масштабованість

- Архітектура додатку повинна дозволяти розширення функціональності — додавання нових об’єктів, анімацій, сценаріїв взаємодії без суттєвих змін основного коду.
- Можливість оновлення або імпорту нових 3D-моделей без необхідності зміни ядра сцени.

3. Надійність

- Програма повинна функціонувати без збоїв протягом тривалого часу.
- Всі основні взаємодії користувача повинні мати обробку помилок або виключних ситуацій (наприклад, відсутність об’єкта, недоступна дія тощо).
- Всі об’єкти сцени повинні завантажуватись коректно і в повному складі при кожному запуску програми.

4. Портативність

- Розроблений додаток має бути сумісним як мінімум з платформою Windows 10.

1.2.4 Обмеження та припущення. У процесі розробки програмного додатку та його подальшого використання можуть виникати певні обмеження, пов’язані з технічними, програмними, апаратними та ресурсними чинниками. Крім того, для успішної реалізації системи були прийняті деякі спрощувальні припущення, що дозволили зосередитися на ключовій функціональності.

Обмеження:

- Апаратні обмеження: Програма розрахована на запуск на персональних комп’ютерах із середнім рівнем продуктивності.
- Обмеження платформи: Реалізація орієнтована на середовище Windows, без повної адаптації під інші операційні системи (Linux, macOS, мобільні ОС) на етапі дипломної розробки.

- Використання базового фізичного рушія: У проєкті застосовується стандартний фізичний рушій Unity (PhysX), що має обмежену точність та не підтримує складні моделі рідин, газів або деформацій.
- Відсутність підтримки багатокористувацького режиму: Поточна версія не передбачає мережевої взаємодії або кооперативного використання.
- Статичність більшості елементів сцени: Не всі об’єкти сцени підтримують динамічну зміну станів або положення — деякі залишаються фіксованими у межах моделі.
- Обмеження на кількість одночасно активних об’єктів: Для уникнення просідання продуктивності кількість активних об’єктів із фізичними компонентами обмежена певним значенням.

Припущення:

- Користувач має базові навички навігації в 3D-просторі: Передбачається, що кінцевий користувач зможе використовувати клавіатуру та мишу для переміщення камерою, взаємодії з об’єктами та користування інтерфейсом.
- Імпортовані 3D-моделі є коректно підготовленими:Передбачається, що всі моделі мають коректну топологію, UV-розгортку та оптимізовану кількість полігонів.
- Система використовується в локальному середовищі:Очікується, що додаток не залежить від зовнішніх серверів чи підключення до Інтернету під час виконання.

- Розширення функціональності можливе у майбутньому: Архітектура системи будується з урахуванням потенційної підтримки нових об'єктів, сценаріїв, UI-елементів або платформ без радикальної перебудови структури проєкту.

1.3 Моделювання предметної області

Моделювання предметної області є одним із ключових етапів у процесі створення програмного забезпечення, оскільки воно дозволяє формалізувати знання про об’єкти реального або уявного світу, що відтворюються в системі. У межах даної дипломної роботи предметною областю виступає система моделювання тривимірних об'єктів у середовищі Unity з урахуванням їхніх фізичних властивостей, взаємодій і керування на сцені.

Метою моделювання є побудова структурованого уявлення про систему, що дозволяє ефективно проектувати архітектуру, логіку та функціонал майбутнього програмного продукту. Для досягнення цієї мети використовуються сучасні методи моделювання, зокрема UML-діаграми, які наочно і формально відображають структуру класів, взаємодію акторів із системою, послідовності дій та компонування елементів.

У цьому розділі буде проведено побудову й аналіз діаграм, які описують основні компоненти системи: класи, послідовності, абстракції, акторів. Ці моделі є необхідною основою для подальшої реалізації програмного продукту в середовищі Unity з використанням мови програмування C#.

1.3.1 Діаграма класів.

Діаграма класів є фундаментальним інструментом об’єктно-орієнтованого моделювання та одним з основних засобів представлення структури системи в нотації UML (Unified Modeling Language). Вона дозволяє візуалізувати ключові сутності проєкту (класи), їх атрибути, методи та взаємозв’язки, що виникають між ними. Побудова діаграми класів є важливим етапом при проєктуванні архітектури системи, оскільки саме класи визначають логіку поведінки об'єктів у програмі, структуру даних та способи їхньої обробки.

У данній роботі, присвяченої моделюванню об'єктів у середовищі розробки Unity, діаграма класів відображає структуру програмних

компонентів, пов’язаних із створенням, керуванням і візуалізацією 3D-об’єктів, реалізацією фізичних властивостей, зберіганням параметрів, а також взаємодією з користувачем. Вона формує основу для подальшої реалізації класів у коді на мові програмування C# та слугує дороговказом для логічної організації системи.

У цьому пункті буде представлено UML-діаграму класів, де описано основні елементи системи: класи, їхні атрибути, методи, а також типи зв’язків між ними. Такий підхід дозволяє забезпечити модульність, масштабованість та розширюваність застосунку, що є критично важливими як для розробника, так і для кінцевого користувача.

Основні елементи діаграми класів на рис. 2:

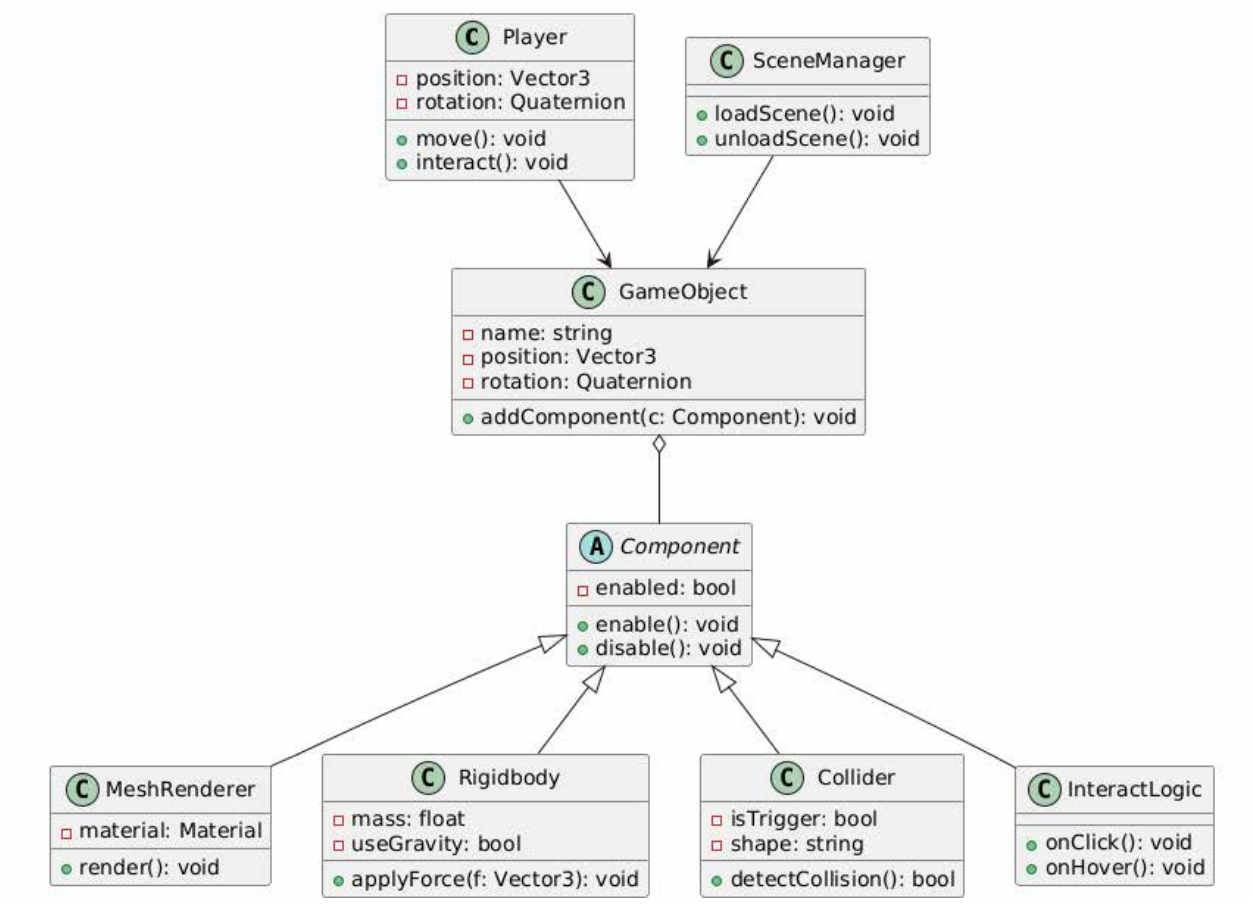

Рис. 2 Діаграма класів

1. Player

Призначення: Моделює користувача або камеру в сцені, яка дозволяє оглядати та взаємодіяти з об’єктами.

Атрибути:

- position: Vector3 — координати у просторі.
- rotation: Quaternion — орієнтація у просторі.

Методи:

- move() — переміщення гравця/камери.
- interact() — взаємодія з об’єктами сцени.

2. GameObject

Призначення: Універсальний об’єкт сцени — базовий блок, до якого прикріплюються компоненти (фізика, візуалізація, логіка).

Атрибути:

- name: string — ім’я об’єкта.
- position: Vector3 — положення в просторі.
- rotation: Quaternion — обертання.

Методи:

- addComponent(c: Component) — додавання нового компонента до об’єкта.

3. Component (абстрактний клас)

Призначення: Базовий клас для всіх типів компонентів, які можуть бути додані до GameObject.

Атрибути:

- enabled: bool — стан активності компонента.

Методи:

- enable() — увімкнення компонента.
- disable() — вимкнення компонента.

4. MeshRenderer (наслідує Component)

Призначення: Відповідає за візуалізацію 3D-моделі.

Атрибути:

- material: Material — матеріал поверхні об’єкта.

Методи:

- render() — обробка відображення на екрані.

5. Rigidbody (наслідує Component)

Призначення: Забезпечує фізичні властивості об’єкта: масу, рух, гравітацію.

Атрибути:

- mass: float — маса об’єкта.
- useGravity: bool — чи діє гравітація.

Методи:

- applyForce(f: Vector3) — застосування сили.

6. Collider (наслідує Component)

Призначення: Дозволяє об'єктам взаємодіяти через зіткнення.

Атрибути:

- isTrigger: bool — визначає, чи є колайдер тригером.
- shape: string — форма колайдера (сфера, коробка тощо).

Методи:

- detectCollision() — перевірка на зіткнення.

7. InteractLogic (наслідує Component)

Призначення: Містить логіку користувацької взаємодії з об’єктом.

Методи:

- onClick() — дія при натисканні.
- onHover() — дія при наведенні.

8. SceneManager

Призначення: Керує завантаженням і вивантаженням сцени, ініціалізує об'єкти.

Методи:

- loadScene() — завантаження сцени.
- unloadScene() — очищення/вивантаження сцени.

1.3.2 Діаграма послідовностей.

Діаграма послідовностей є ключовим елементом динамічного моделювання програмної системи в UML. Вона описує, як об’єкти системи взаємодіють між собою у часі під час виконання певного сценарію або функції. На відміну від діаграми класів, яка фокусується на статичній структурі, діаграма послідовностей дозволяє відстежити хронологію викликів методів, обмін повідомленнями та реакції об’єктів на події в системі.

Діаграма послідовностей є ефективним інструментом для формалізації сценаріїв використання програмної системи. Її основна перевага полягає в тому, що вже на етапі проектування можна визначити всі ключові елементи, що взаємодіють між собою, а також встановити логіку обміну повідомленнями між цими елементами. Надалі ці взаємодії трансформуються у відповідні класи, методи та об’єкти — згідно з концепцією об’єктно-орієнтованого програмування.

Завдяки цій діаграмі формується модель поведінки системи: усі події, що виникають у процесі виконання певного варіанту використання, будуть підтримуватись і оброблятись відповідними об’єктами. Діаграма дозволяє наочно відобразити взаємодії між об’єктами як ланцюжок подій у часі — починаючи від ініціації процесу до досягнення певної мети або результату.

На схемі фіксуються лише ті об’єкти, які беруть безпосередню участь у реалізації сценарію, залишаючи поза увагою другорядні або непричетні елементи, що дозволяє зосередитись на головній логіці виконання завдання.

В даній дипломній роботі, діаграма послідовностей відіграє важливу роль у формуванні логіки взаємодії між користувачем, елементами інтерфейсу, 3D-об’єктами та керуючими компонентами. Вона демонструє, як система реагує на дії користувача (наприклад, створення об’єкта, застосування трансформації чи запуск симуляції) та які об’єкти відповідають за ті чи інші етапи обробки подій.

Побудова такої діаграми дозволяє:

- виявити приховані залежності між компонентами;
- оцінити складність логіки взаємодії;
- проєктувати коректні методи комунікації між класами;
- спростити подальше тестування та реалізацію.

У цьому пункті буде представлено одну діаграму послідовностей, що описують типовий сценарій використання системи. Ця діаграма стане важливою частиною моделі архітектури застосунку, а також слугуватимуть основою для реалізації логіки обміну повідомленнями між об’єктами в середовищі Unity з використанням мови C#.

Основні елементи діаграми послідовностей на рис. 3:

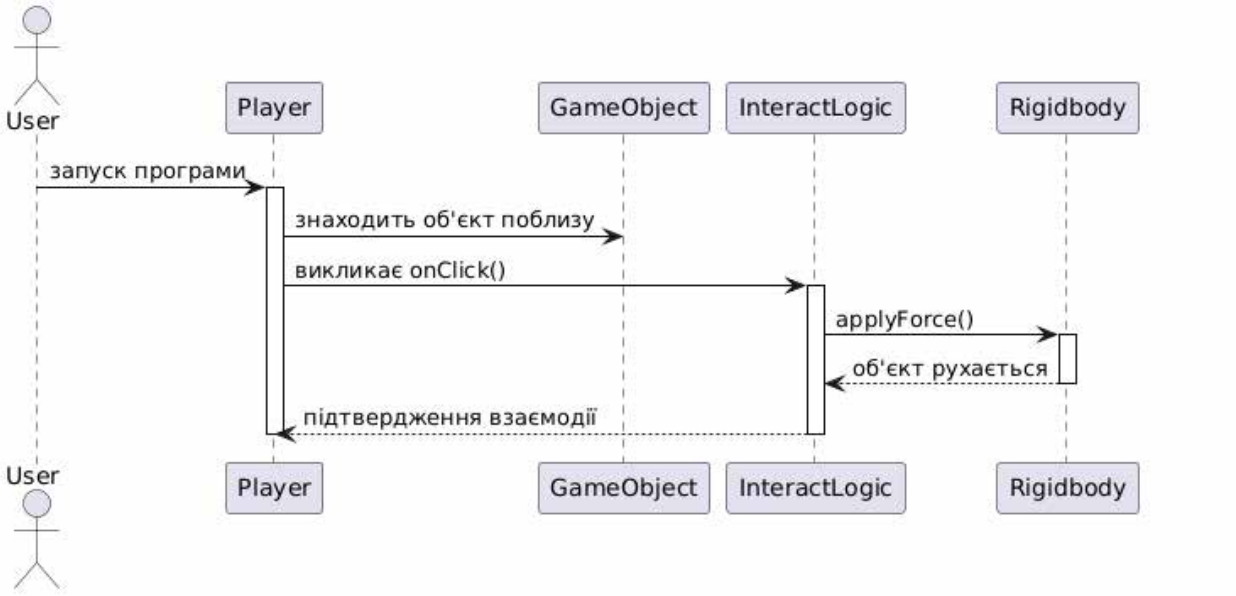

Рис. 3 Діаграма послідовностей

Опис послідовності:

- Користувач запускає гру (через Player);
- Гравець знаходить об'єкт (GameObject);
- Викликається метод взаємодії (onClick());
- Логіка передає команду фізиці (applyForce() у Rigidbody);
- Повертається результат дії.

1.3.3 Абстракція. Абстракції показують ключові властивості та

обов’язки кожного елемента системи. Вона допомагає зрозуміти основні характеристики та функції елементів системи.

У процесі розробки складної програмної системи важливо не лише деталізувати окремі компоненти, а й вміти уявити їх на вищому рівні узагальнення. Діаграма абстракції виступає засобом, який дозволяє відобразити логіку системи у спрощеній формі, де увага зосереджується не на внутрішній реалізації, а на сутності об’єктів та їхніх функціональних призначеннях.

На відміну від технічних діаграм, таких як діаграми класів або послідовностей, діаграма абстракції допомагає зосередитися на концептуальних одиницях — ролях, які виконують об’єкти в системі, зв’язках між ними та загальних принципах їхньої взаємодії. Такий підхід є особливо корисним на ранніх етапах проєктування або при комунікації з не технічними учасниками команди, адже він дозволяє зрозуміти логіку без необхідності заглиблюватися в код.

Діаграма абстракції відіграє роль «містка» між предметною областю і програмною реалізацією. Вона демонструє, як узагальнені компоненти — такі як модуль управління сценою, модуль користувача, або генератор моделей — взаємодіють між собою для досягнення кінцевого функціонального результату.

Діаграма абстракції є важливим аналітичним інструментом, що допомагає краще зрозуміти структуру й логіку проєкту до моменту глибокого технічного занурення.

Основні елементи діаграми абстракцій на рис 4:

Рис. 4 Абстракції

1. Об’єкт

Абстракція сцени, яка описує базову одиницю в Unity — GameObject.

Важливі властивості:

- Назва — унікальне ім’я об’єкта у сцені;
- Позиція — координати у 3D-просторі;
- Ротація — обертання об’єкта в просторі.

Обов’язки:

- Бути частиною сцени;
- Зберігати та поєднувати компоненти, що додають функціональність.

2. Користувач

Абстракція актора, який взаємодіє з віртуальним середовищем. Це може бути гравець або камера.

Важливі властивості:

- Положення — де знаходиться користувач у сцені;
- Камера — віртуальний “погляд” користувача;

- Інтерфейс управління — засоби керування (клавіатура, миша, UI).

Обов’язки:

- Навігація сценою;
- Ініціювання взаємодії з об’єктами (натискання, наближення тощо).

3. Компонент

Модуль, що реалізує поведінку об’єкта. В Unity кожен GameObject містить компоненти.

Важливі властивості:

- Активність — увімкнений чи вимкнений стан;
- Тип — візуалізація, фізика, логіка;
- Дані — властивості компонента (матеріали, маса тощо).

Обов’язки:

- Додавати функціональність до об'єкта;
- Обробляти графіку, фізику, анімацію або взаємодію.

4. Фізика

Спеціалізований компонент для фізичної симуляції, пов’язаний із Rigidbody.

Важливі властивості:

- Маса — фізична вага об’єкта;
- Гравітація — чи діє на об’єкт сила тяжіння;
- Швидкість — параметри руху.

Обов’язки:

- Реалістична симуляція руху в просторі;
- Реакція на сили, зіткнення, падіння.

5. Логіка

Компонент, який описує реакції об’єкта на дії користувача (тригери, події).

Важливі властивості:

- Події — взаємодії, які система може відстежити (наприклад, клік, наведення);
- Тригери — умови, які викликають дію.

Обов’язки:

- Реагування на події;
- Запуск певних дій або сценаріїв (анімація, переміщення, зміна стану).

1.3.4 Діаграма акторів. Мета діаграми показати, які дії можуть ініціювати актори, що взаємодіють із системою, а також які саме компоненти відповідають за обробку цих дій. На рис 5 наведена діаграма акторів, яка демонструє взаємодію.

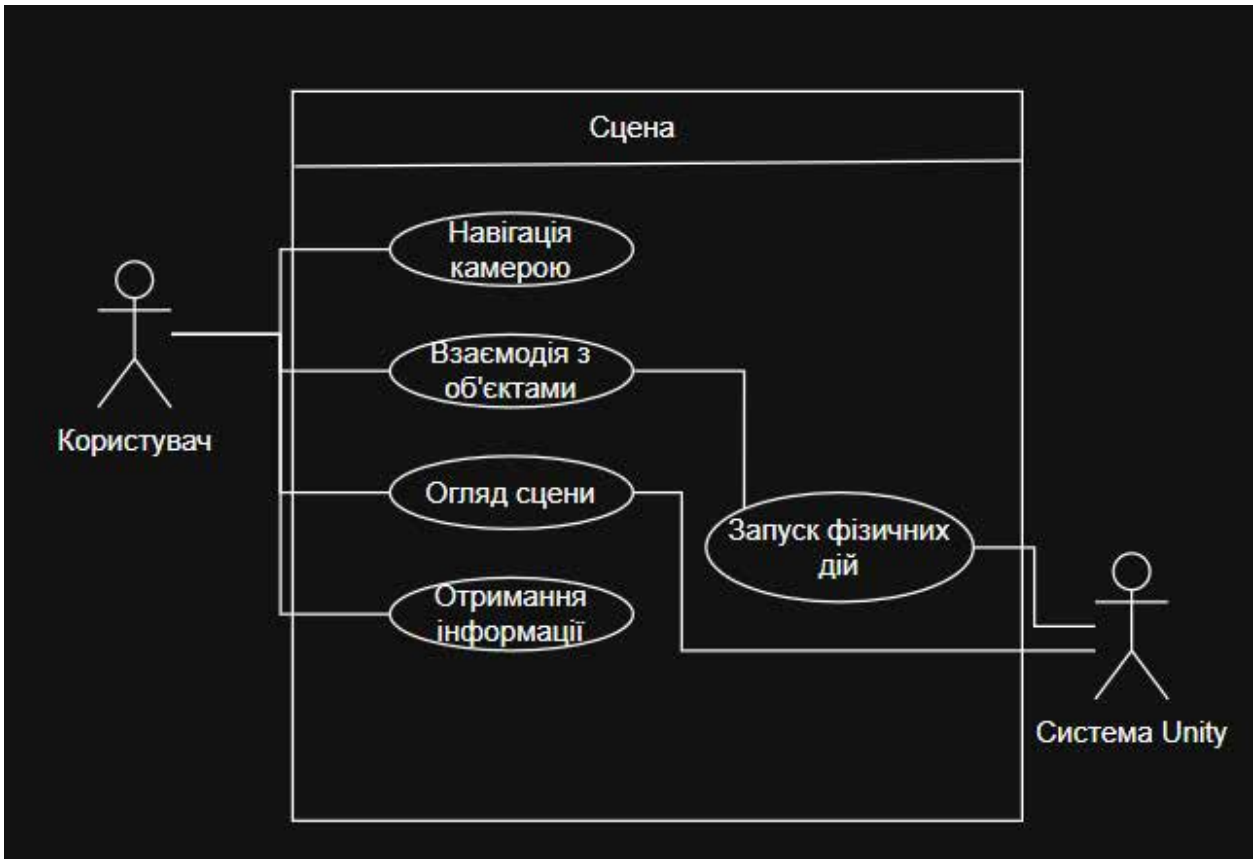

Рис 5 Діаграма акторів

Пояснення до діаграми:

Користувач — це основний актор-юдина, яка запускає додаток та безпосередньо взаємодіє з віртуальним середовищем Unity. Його роль — навігація по сцені, ініціація подій, ознайомлення з вмістом об’єктів.

Виконує такі дії:

- Огляд сцени — пересування камерою або вільний режим огляду.
- Взаємодія з об’єктами — наприклад, клік на об’єкт, відчинення дверей, натискання кнопок.

- Отримання інформації — перегляд підказок, описів або параметрів об’єктів.
- Навігація камерою — управління оглядом (обертання, зум, переміщення).

Система Unity — це умовний актор, який представляє рушій Unity, а саме його автоматизовані компоненти: фізичний рушій, система рендерингу та обробки взаємодій.

Реагує на дії користувача:

- Запуск фізичних дій — симуляція падіння, зіткнень або переміщень при взаємодії з об’єктами.
- Огляд сцени — відтворення сцени в реальному часі (рендеринг, світло, тіні тощо).

1.3.5 Діаграма діяльності

Діаграма діяльності є наочною формою представлення логіки роботи системи й нагадує блок-схему алгоритму, але замість традиційних процедур на ній зображено окремі дії або варіанти використання. Вона особливо корисна при моделюванні поведінки, яка включає багато паралельних або взаємопов'язаних процесів.

Такі діаграми подаються у вигляді орієнтованого графа: вузли цього графа відповідають окремим діям, а дуги — переходам між ними. Дія може бути виконана один або кілька разів протягом життєвого циклу процесу, залежно від умови або сценарію. Дії можуть приймати, обробляти або трансформувати дані, і в багатьох випадках мають виконуватись у чітко визначеній послідовності.

Завдяки діаграмам діяльності можна наочно описувати алгоритми, які лежать в основі функціонування майбутньої інформаційної системи. Перед початком розробки важливо детально описати ці алгоритми, щоб забезпечити правильну автоматизацію процесів.

Розробник має мати повне уявлення про всі робочі сценарії, які буде обробляти система, інакше можна упустити важливі функціональні аспекти. Також варто враховувати, що впровадження нової системи може бути складним, тому можливі труднощі потрібно передбачити ще на етапі проектування.

Ключовим елементом діаграми діяльності є активність — це стан, у якому система виконує певну дію. Після завершення дії система переходить до наступної активності. Перехід між активностями може відбуватись не лише послідовно, а й за певною подією. Діаграма обов’язково містить початок і кінець процесу, а також може включати розгалуження (паралельні гілки) і злиття потоків через відповідні конструкції для логічних умов і синхронізації.

Діаграма ілюструє типовий робочий процес користувача в середовищі Unity для моделювання 3D-об’єктів — від запуску програми до збереження сцени. Вона відображає основні етапи, дії та варіанти розвитку подій, включаючи умовні розгалуження та можливість паралельного виконання окремих процесів.(рис 6)

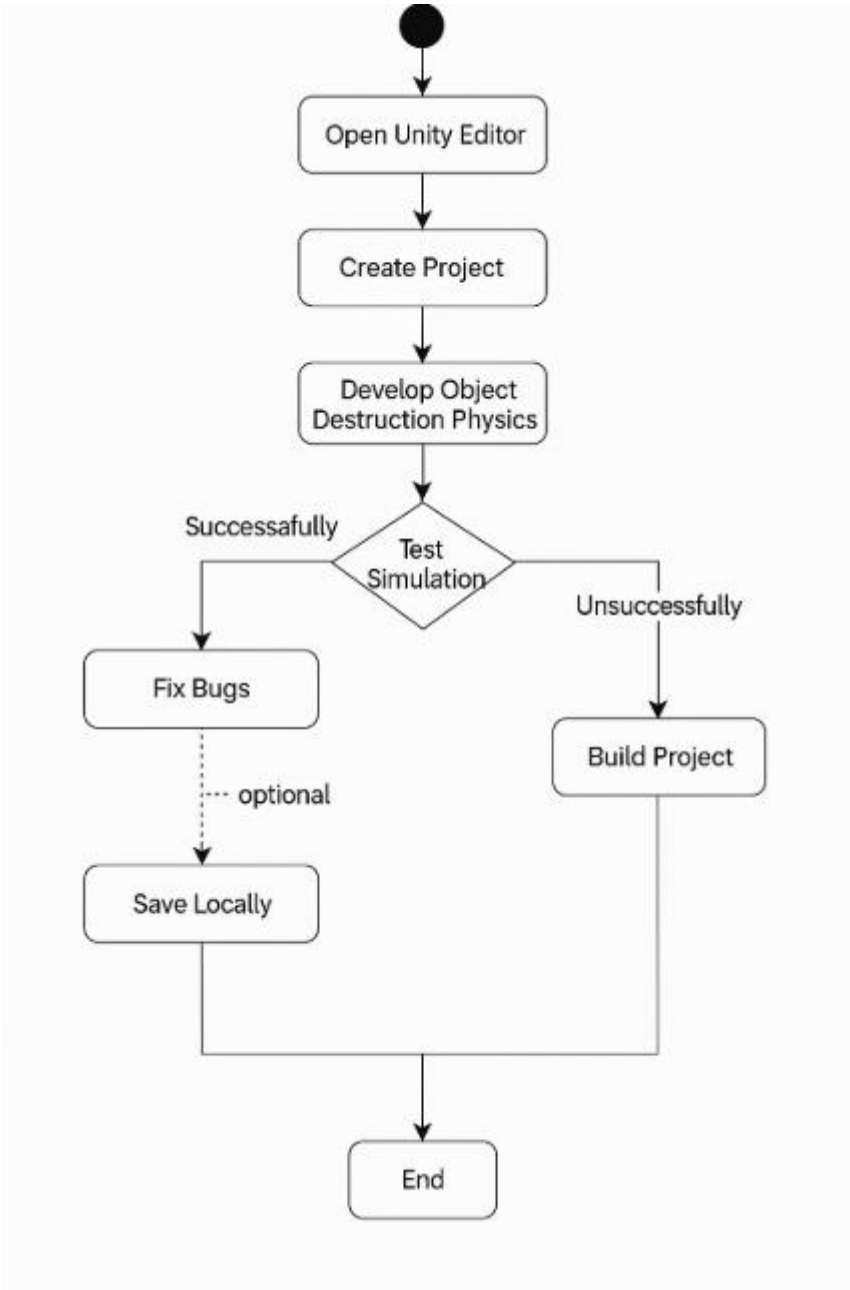

Рис 6 діаграма діяльності

Основні активності, зображені на діаграмі:

- Початок процесу – стартова точка роботи системи.
- Запуск Unity – ініціалізація середовища розробки.
- Створення нового проєкту або відкриття існуючого – користувач може почати нову сцену або редагувати вже наявну.
- Імпорт 3D-моделей або створення примітивів – додавання об’єктів до сцени.
- Запуск симуляції – тестування сцени в режимі Play.

Умовне розгалуження:

- Якщо сцена працює коректно — перехід до збереження.
- Якщо виявлено помилки — повернення до редагування скриптів або властивостей.
- Збереження проєкту – фінальний етап.
- Завершення роботи – вихід із Unity або перехід до іншої задачі.

1.4 Огляд інформаційних джерел та існуючих рішень

Для розробки якісного програмного продукту, що реалізує моделювання об’єктів у середовищі Unity, необхідно провести аналіз науково-технічної літератури, інструкцій, методичних матеріалів, документації до програмних інструментів та прикладів реалізації подібних рішень. Такий огляд дозволяє виявити сучасні підходи, інструменти та практики, які вже використовуються в сфері тривимірного моделювання та інтерактивних симуляцій.

Особливу увагу приділено вивченню архітектурних підходів до побудови об’єктно-компонентних систем, реалізації фізики в Unity, а також механізмам взаємодії користувача з 3D-середовищем. Крім того, аналіз існуючих проєктів, публікацій у спеціалізованих ресурсах та прикладів з відкритим кодом дозволив виділити ефективні рішення, які можуть бути адаптовані або використані як основа для реалізації власного додатку.

У цьому підрозділі представлено порівняння популярних платформ і бібліотек, огляд навчальних курсів, офіційної документації Unity, а також прикладів реалізації схожих функціональних систем.

1.4.1 Аналіз поточних досліджень та їх результатів.

У ряді сучасних досліджень розглядаються практичні та технологічні аспекти використання середовища Unity для моделювання та візуалізації тривимірних

об'єктів. Так, у статті, опублікованій в ISPRS Annals, автори дослідили можливості Unity у контексті моделювання міського середовища та інтеграції з великими геопросторовими наборами даних, зокрема CityGML. На прикладі візуалізації сонячного потенціалу та підземної інфраструктури Стамбула було показано переваги рушія Unity для створення інтерактивних 3D-сцен. Разом з тим, було виявлено низку викликів, пов’язаних із масштабністю моделей та продуктивністю при відображенні великої кількості об’єктів.

Інше дослідження, розміщене на ResearchGate, зосереджене на техніках моделювання 3D-об'єктів для використання в Unity. У ньому автори детально аналізують етапи створення моделей — від проєктування геометрії до текстурування — та наголошують на необхідності оптимізації кількості полігонів і правильному налаштуванні UV-розгортки. Окрема увага приділяється процесу імпорту з Blender, що дозволяє ефективно готувати моделі для подальшого використання в ігровому рушії.

Додатково в блозі CrossComm було представлено приклад реального застосування Unity для створення VR-середовищ з використанням 3D-моделей. Автори публікації акцентують увагу на важливості налаштування матеріалів, PBR-текстур та освітлення для досягнення високого рівня візуального реалізму. Також розглядаються практичні рекомендації з оптимізації моделей з метою забезпечення продуктивної роботи VR-додатків.

Зокрема, у дослідженні “FastPoints: A State-of-the-Art Point Cloud Renderer for Unity” запропоновано ефективний рендерер хмар точок для Unity, який здатний обробляти великі обсяги даних без попередньої обробки. Це рішення забезпечує високу продуктивність та реалістичність візуалізації, що є критично важливим для інтерактивних систем реального часу.

Аналогічно, у роботі “Research on the Application of Products Based on Unity3D” акцентується увага на широкому спектрі застосування Unity3D у промисловості, освіті та розважальній сфері. Автори підкреслюють гнучкість рушія у створенні інтерактивних 3D-продуктів — від віртуальних турів до симуляцій і навчальних систем. В обох дослідженнях Unity розглядається як

універсальний інструмент, здатний підтримувати високоточне моделювання об'єктів і надання користувачеві реалістичного досвіду взаємодії з цифровим середовищем.

1.4.2 Огляд існуючих програмних рішень.

У сучасній практиці моделювання тривимірних об’єктів та створення інтерактивних візуальних середовищ існує низка програмних рішень, які дозволяють ефективно реалізовувати проєкти різної складності. Найбільш поширеними серед них є Unity, Unreal Engine, Godot Engine та CryEngine. Кожен з цих інструментів має свої переваги, обмеження та сферу застосування.

Unity є одним із найпопулярніших рушіїв для розробки інтерактивних 3D-додатків і симуляцій. Його компонентно-орієнтована архітектура дозволяє легко створювати складні сцени, а підтримка C# і численних плагінів значно розширює функціональні можливості. Завдяки потужному редактору та активній спільноті Unity часто використовується у VR/AR-проєктах, мобільних додатках та освітніх симуляціях. Його недоліками є дещо обмежені можливості у фотореалізмі та менша ефективність при симуляціях високого фізичного навантаження.

Unreal Engine, у свою чергу, вирізняється високим рівнем графіки, підтримкою реалістичного освітлення та візуального програмування через систему Blueprint. Це робить його надзвичайно привабливим для розробки AAA-ігор або симуляцій з фотореалістичною графікою. Проте, він має більш складний поріг входу та високі вимоги до апаратного забезпечення, що може бути критичним для індивідуальних розробників або студентських проєктів.

Godot Engine — це легкий та відкритий рушій, який підходить для невеликих 3D-проєктів і навчальних завдань. Хоча він забезпечує простоту використання та невеликі системні вимоги, його функціональність у сфері складного 3D-моделювання та фізики поки що поступається Unity та Unreal.

CryEngine орієнтований на створення високоякісних 3D-сцен з реалістичним освітленням і просунутою симуляцією руйнувань, однак через

складність освоєння, нестачу документації та слабку підтримку спільноти використовується обмежено.

1.4.3 Оцінка ефективності існуючих рішень

Unreal Engine демонструє високу ефективність у створенні графічно насичених сцен завдяки фотореалістичному рендерингу, сучасним системам освітлення (Lumen) та геометрії (Nanite). Однак його складність, ресурсоємність та високий поріг входження обмежують використання в освітніх і навчальних розробках, де гнучкість і простота мають перевагу.

Godot Engine, хоча й забезпечує базову підтримку 3D-графіки та фізики, виявляється менш ефективним у реалізації складних симуляцій або високонавантажених сцен, що робить його менш придатним для моделювання складної поведінки об'єктів у динамічному середовищі.

У цьому контексті Unity показав найвищу ефективність у поєднанні гнучкого моделювання об'єктів, налаштування їхньої взаємодії, візуалізації в реальному часі та широкої підтримки фізичних ефектів. Його переваги полягають у наявності великої кількості готових компонентів, інтуїтивному редакторі, зручній інтеграції з моделями з Blender та підтримці багатьох платформ. Окрім того, Unity має низький поріг входу, що дозволяє використовувати його як у професійних, так і в навчальних цілях.

Загалом, ефективність існуючих рішень значною мірою залежить від поставлених цілей. Для реалізації завдань, що передбачають створення інтерактивного середовища з об’єктами, які мають фізичні властивості, візуальні ефекти та логіку взаємодії — Unity є оптимальним і найбільш збалансованим варіантом.

1.5 Постановка завдання

У сучасному світі комп’ютерні технології стрімко розвиваються, а візуалізація об'єктів та середовищ стає однією з ключових складових в багатьох галузях — від комп’ютерних ігор до віртуальної та доповненої

реальності, архітектурного дизайну, симуляцій, освіти й наукових візуалізацій. Однією з найпопулярніших платформ для створення інтерактивних 3D-сцен є Unity — потужне середовище розробки, яке дозволяє створювати кросплатформенні додатки із високим рівнем інтерактивності.

Моделювання об'єктів у такому середовищі включає не лише створення геометричних форм, а й налаштування їхніх фізичних властивостей, взаємодії, анімацій, освітлення та поведінки в реальному часі. Така діяльність вимагає комплексного підходу — від розробки структури даних і логіки взаємодії об’єктів до реалізації ефективного візуального представлення та оптимізації продуктивності.

Актуальність теми зумовлена зростаючим попитом на якісні інтерактивні 3D-сцени, які потребують точного моделювання та реалізації фізичних процесів, зокрема для створення ігор, симуляторів або навчальних програм. Unity забезпечує гнучке середовище для реалізації таких задач завдяки своїй підтримці фізики, скриптової інтеграції через C#, багатої бібліотеки готових компонентів та можливостей швидкого прототипування.

Цілю є розробка прикладного програмного рішення, яке дозволяє створювати та керувати 3D-об’єктами з урахуванням їхніх фізичних і візуальних властивостей, використовуючи інструментарій Unity. Під час виконання проєкту передбачається дослідити основні підходи до моделювання в Unity, реалізувати структуру взаємодії об'єктів на сцені, застосувати скриптові механізми для управління поведінкою елементів та забезпечити користувацький інтерфейс для взаємодії з моделями.

Основні цілі програми:

- Створення 3D-моделей об'єктів з можливістю їх редагування та масштабування;
- Інтеграція фізичних властивостей (масса, сила тяжіння, колізії, матеріали тощо);
- Реалізація взаємодії об’єктів між собою у реальному часі (наприклад, зіткнення або рух);

- Забезпечення візуалізації сцени з базовим освітленням, камерами та UI-компонентами;
- Розробка користувацького інтерфейсу для взаємодії з об'єктами (переміщення, обертання);
- Оптимізація продуктивності сцени для стабільної роботи на середньостатистичному ПК;
- Тестування функціоналу в локальному середовищі запуску Unity;

2 Проектування інформаційного та програмного забезпечення

2.1Логічна модель даних у вигляді ER-діаграми

ER-діаграма (модель «сутність–зв’язок» або ER-діаграма) — це концептуальна модель даних, яка слугує основою для зручного та наочного проектування структури бази даних. Вона є універсальним підходом, на основі якого можуть бути побудовані різні типи баз даних: реляційні, ієрархічні, мережеві, об’єктно-орієнтовані тощо. Основними складовими цієї моделі є поняття сутності, атрибута та зв’язку.

Використання ER-діаграми під час проєктування масштабних інформаційних систем дозволяє уникнути критичних помилок на ранніх етапах, що в подальшому могло б призвести до складних і затратних змін у структурі БД або до переробки пов’язаного програмного забезпечення. Такі помилки часто стають причиною додаткових витрат часу, ресурсів і коштів.

ER-діаграма подає базу даних у вигляді графічної схеми, яка чітко відображає логічну структуру предметної області. За допомогою ER-діаграми можна легко візуалізувати сутності, їх властивості (атрибути), а також взаємозв’язки між ними, що значно спрощує процес аналізу та розробки системи.

Побудова ER-діаграми дає змогу наочно уявити, які об’єкти беруть участь у логіці програми, які атрибути вони мають, а також як між собою пов’язані. Це полегшує подальше проєктування бази даних або внутрішніх структур зберігання стану в самій грі чи додатку.

ER-діаграма виконує функцію формального опису інформаційної структури програми. Вона дозволяє визначити основні сутності, такі як моделі об’єктів, сцени, користувачі та збереження проєктів, а також описати способи їх взаємодії. Створення ER-діаграми є базовим кроком у підготовці до розробки бази даних.

Логічна модель даних у вигляді ER-діаграми забезпечує цілісне розуміння структури майбутньої інформаційної бази, що сприяє ефективній

реалізації програмного продукту, оптимізації запитів до бази даних та підвищенню загальної надійності й керованості системи.

Основні елементи ER-діаграми на рис 6.

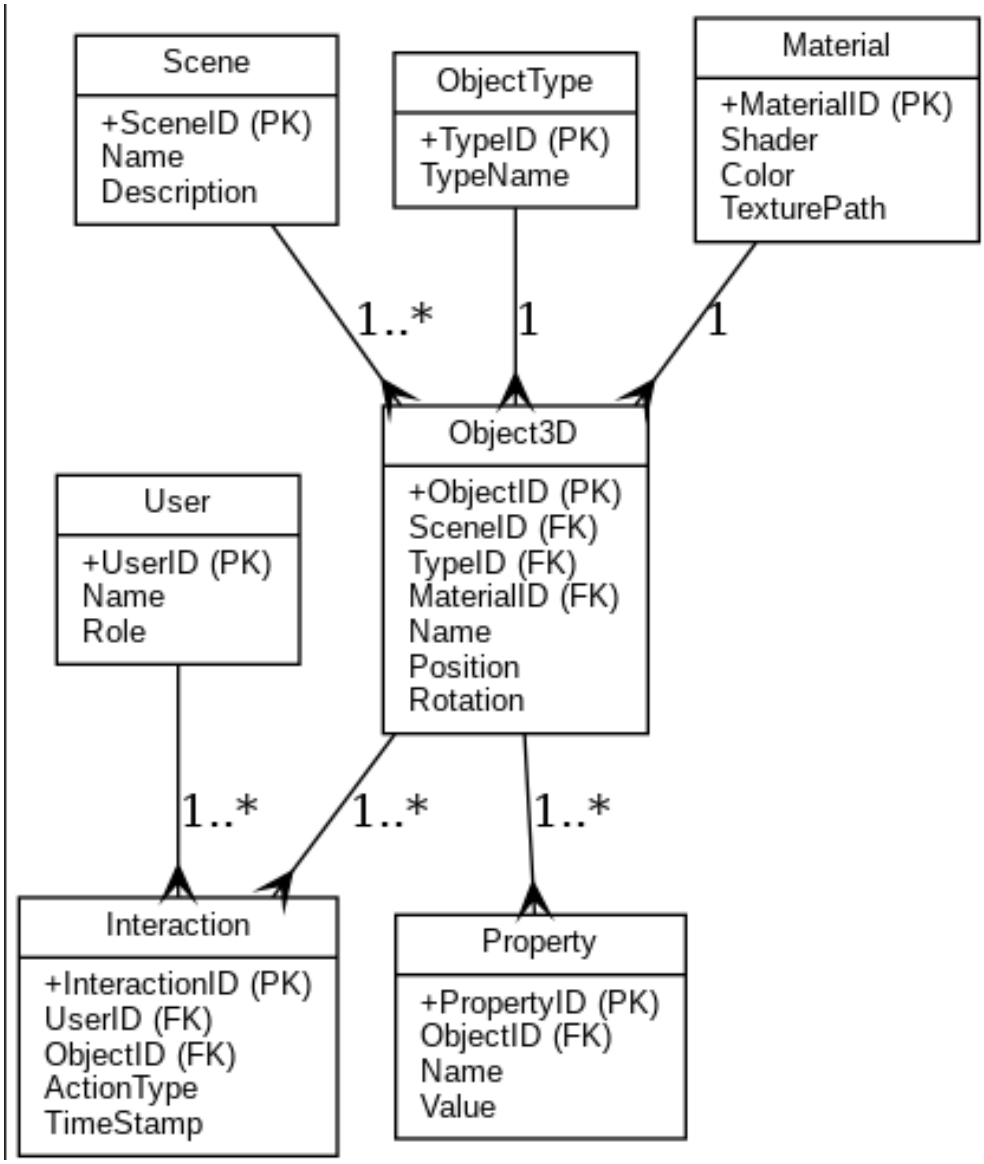

Рис 6 ER-діаграма

Сутність представляє собою об’єкт або концепцію, яку потрібно зберігати у базі даних. Кожна сутність характеризується набором атрибутів, які визначають її властивості. На ER-діаграмі відображені такі сутності:

1) Сутності:

• User (Користувач): користувач системи, який взаємодіє з віртуальними об’єктами та виконує дії у сцені (наприклад, гравець, розробник).

• Scene (Сцена): віртуальне середовище, у якому розміщуються 3D-об’єкти та відбуваються взаємодії.

• Object3D (Об’єкт): тривимірний елемент сцени, який має позицію, вигляд, тип та набір властивостей.

• ObjectType (Тип об’єкта): класифікація об’єктів за функціональністю або природою (напр. кнопка, персонаж, стіна).

• Material (Матеріал): візуальні властивості об’єкта, включаючи колір, текстуру, шейдер тощо.

• Interaction (Взаємодія): дія користувача, спрямована на об’єкт (наприклад, клік, переміщення, активування події).

• Property (Властивість): додаткові параметри об’єкта, які не входять до основного набору (напр. швидкість, вага, стан).

2) Атрибути (Attributes)

User (Користувач):

- UserID (PK): унікальний ідентифікатор користувача;
- Name: ім’я або псевдонім;
- Role: роль у системі (гравець, адміністратор тощо).

Scene (Сцена):

- SceneID (PK): унікальний ідентифікатор сцени;
- Name: назва сцени;
- Description: короткий опис призначення сцени.

Object3D (Об’єкт):

- ObjectID (PK): унікальний ідентифікатор 3D-об’єкта;
- SceneID (FK): зв’язок із відповідною сценою;
- TypeID (FK): зв’язок із типом об’єкта;
- MaterialID (FK): зв’язок із матеріалом;
- Name: назва об’єкта;
- Position: положення у 3D-просторі;
- Rotation: орієнтація в просторі.

ObjectType (Тип об’єкта):

- TypeID (PK): унікальний ідентифікатор типу;
- TypeName: назва типу (кнопка, стіна тощо).

Material (Матеріал):

- MaterialID (PK): унікальний ідентифікатор матеріалу;
- Shader: тип візуального шейдера;
- Color: основний колір;
- TexturePath: шлях до текстури.

Interaction (Взаємодія):

- InteractionID (PK): унікальний ідентифікатор взаємодії;
- UserID (FK): користувач, що здійснив взаємодію;
- ObjectID (FK): об’єкт, до якого застосовано дію;
- ActionType: тип дії (клік, переміщення, включення);
- TimeStamp: дата і час взаємодії.

Property (Властивість):

- PropertyID (PK): унікальний ідентифікатор властивості;
- ObjectID (FK): об’єкт, до якого належить властивість;
- Name: назва властивості;
- Value: значення властивості.

3) Зв’язки (Relationships)

User та Interaction: Один користувач може здійснювати багато взаємодій. Взаємозв’язок реалізовано через UserID у сутності Interaction.

Scene та Object3D: Кожна сцена може містити багато об’єктів.

Зв’язок реалізовано через SceneID у сутності Object3D.

Object3D та ObjectType: Кожен об’єкт має один тип.

Зв’язок — TypeID у Object3D вказує на ObjectType.

Object3D та Material: Кожен об’єкт має один матеріал.

Зв’язок — MaterialID у Object3D вказує на Material.

Object3D та Property: Один об’єкт може мати багато властивостей.

Зв’язок — ObjectID у Property вказує на Object3D.

Object3D та Interaction: Один об’єкт може бути ціллю багатьох взаємодій.

Зв’язок — ObjectID у Interaction.

2.2Діаграма класів та кооперацій

Діаграма кооперацій ілюструє взаємодію основних об'єктів системи під час моделювання поведінки у сцені Unity (рис 7). Вона описує, як користувач ініціює дії, які передаються об'єктам сцени, а ті у свою чергу активують відповідну логіку або фізичну реакцію. Усі об’єкти розміщуються в межах сцени, яка виступає в ролі контейнера.

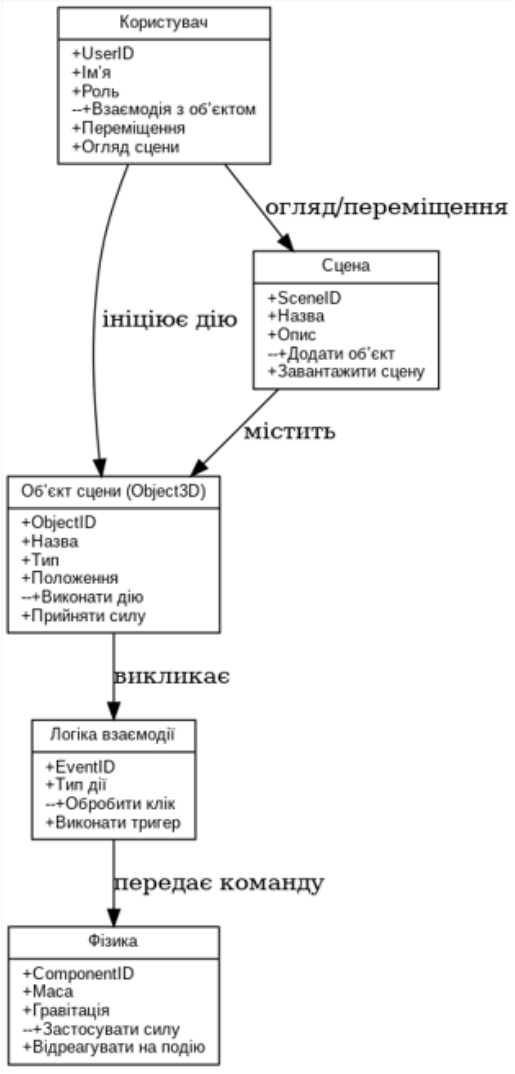

Рис 7 діаграма кооперацій

Об’єкти діаграми:

Користувач (User)

Роль: Ініціатор взаємодій.

Атрибути:

- UserID: ідентифікатор користувача;
- Ім’я, Роль: інформація про особу.

Методи:

- Взаємодія з об'єктом: клацання, наближення, активація;
- Переміщення: переміщення в сцені;
- Огляд сцени: контроль камери або навігація.

Об’єкт сцени (Object3D)

Роль: Основний елемент віртуального середовища.

Атрибути:

- ObjectID, Назва, Тип, Положення.

Методи:

- Виконати дію: відповісти на подію користувача;
- Прийняти силу: змінити положення під впливом фізики.

Логіка взаємодії (Interact Logic)

Роль: Приймає події від об’єкта та обробляє логічну поведінку.

Атрибути:

- EventID, Тип дії.

Методи:

- Обробити клік: запуск скрипту чи анімації;
- Виконати тригер: виклик події у фізичному або логічному середовищі.

Фізика (Physics)

Роль: Реалізує зміну положення об’єкта або реакцію на взаємодію.

Атрибути:

- ComponentID, Маса, Гравітація.

Методи:

- Застосувати силу: перемістити або обернути об'єкт;
- Відреагувати на подію: симулювати падіння, зіткнення тощо.

Сцена (Scene)

Роль: Контейнер для всіх об'єктів, в якому розгортається взаємодія.

Атрибути:

- SceneID, Назва, Опис.

Методи:

- Додати об’єкт: динамічне оновлення сцени;
- Завантажити сцену: ініціалізація об'єктів при запуску.

Зв’язки

Користувач → Об’єкт сцени: ініціює взаємодію;

Об’єкт сцени → Логіка взаємодії: активує обробку події;

Логіка → Фізика: передає команду на зміну стану;

Користувач → Сцена: взаємодіє з середовищем, перемикає або навігує;

Сцена → Об’єкти: містить набір 3D-елементів.

2.3 Діаграма пакетів

Діаграма пакетів відображає логічну структуру програмної системи та розподіл функціональних компонентів у вигляді окремих модулів (пакетів), які взаємодіють між собою. Такий підхід дозволяє забезпечити модульність, гнучкість і масштабованість розроблюваного програмного додатку рис 8.

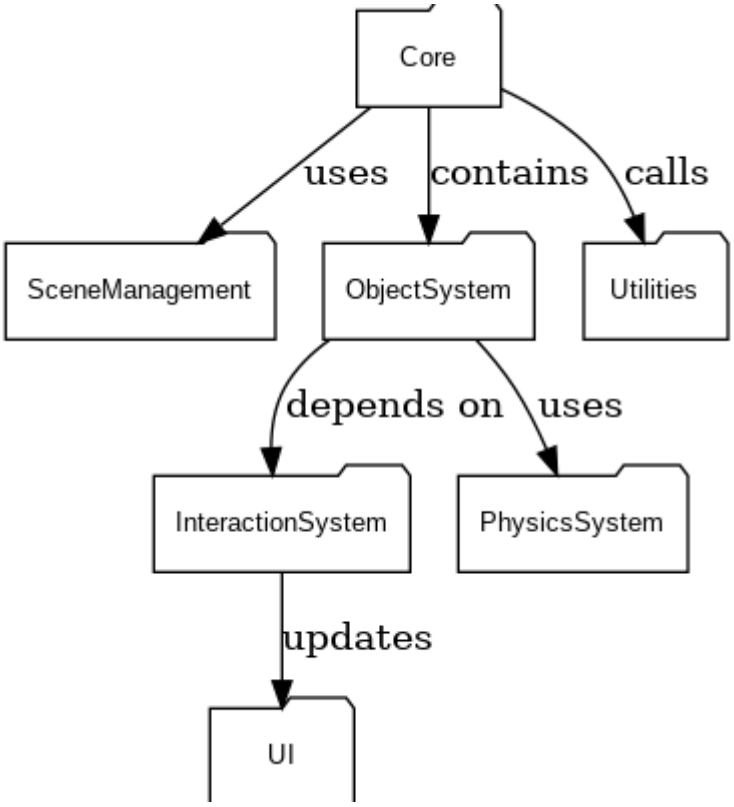

рис 8 діаграма пакетів

Core (Ядро системи): Центральний модуль, який координує роботу всієї системи.

Він викликає інші пакети, контролює основний потік виконання програми та забезпечує інтеграцію між частинами системи.

Використовує:

- SceneManagement — для керування сценами;
- ObjectSystem — для створення та керування об’єктами;
- Utilities — для використання допоміжних функцій.

SceneManagement (Керування сценами): Відповідає за створення, завантаження, оновлення та перемикання сцен. Містить засоби для організації просторового розміщення об’єктів.

ObjectSystem (Система об’єктів): Містить логіку створення, зберігання та оновлення 3D-об’єктів, які розміщуються у сцені. Цей пакет є ключовим для візуального моделювання.

Залежить від:

- InteractionSystem — для реалізації реакцій об’єктів на дії користувача;

- PhysicsSystem — для застосування фізичних впливів.

InteractionSystem (Система взаємодії): Обробляє події взаємодії користувача з об’єктами (натискання, наведення, активація). Реалізує логіку відповіді об’єктів на дії гравця.

Оновлює:

- UI — інтерфейс користувача у відповідь на події.

PhysicsSystem (Фізичний модуль): Реалізує симуляцію фізичних властивостей: гравітацію, застосування сили, обробку зіткнень. Використовується системою об’єктів для моделювання реалістичної поведінки.

UI (Інтерфейс користувача): Відповідає за графічне відображення подій, станів об’єктів та взаємодій. Оновлюється системою взаємодії у відповідь на дії користувача.

Utilities (Допоміжні засоби): Містить загальні функції, інструменти або сервіси, які не належать до конкретного модуля, але використовуються у Core (наприклад, логування, серіалізація, математичні обчислення тощо).

2.4 Діаграма компонентів

Діаграма компонентів (на рис 9) відображає структурну організацію програмного застосунку *Unity Modeling Application* як набору взаємопов’язаних функціональних блоків. Вона демонструє, як основна система та її графічний інтерфейс користувача взаємодіють з окремими функціональними модулями, що відповідають за різні аспекти моделювання об’єктів у середовищі Unity.

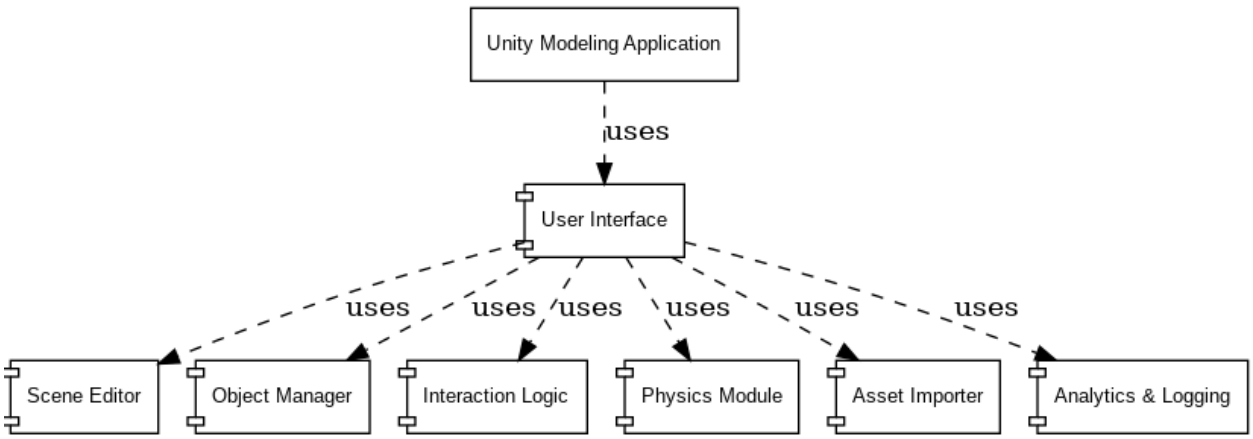

рис 9 діаграма компонентів

Unity Modeling Application - це головна система, яка інкапсулює всю функціональність додатку. Вона взаємодіє з користувачем через спеціалізований User Interface, який у свою чергу опосередковує доступ до решти компонентів.

User Interface (UI): Центральний компонент для взаємодії з користувачем. Через UI здійснюється доступ до основних підсистем, зокрема:

- Редагування сцен;
- Управління об’єктами;
- Обробка взаємодій;
- Застосування фізичних ефектів;
- Імпорт активів;
- Аналіз дій та логування.

Scene Editor: Відповідає за редагування віртуальної сцени: додавання, переміщення та видалення об’єктів, налаштування параметрів сцени, збереження стану.

Object Manager: Керує всіма 3D-об’єктами, які існують у сцені: їх атрибутами, типами, унікальними ідентифікаторами. Надає функції для створення, клонування, оновлення або видалення об’єктів.

Interaction Logic: Реалізує логіку взаємодії користувача з об'єктами: події натискання, наведення, активації, тригерні сценарії. Цей модуль відповідає за обробку реакцій на дії у режимі реального часу.

Physics Module: Симулює фізичну поведінку об’єктів: застосування сили, гравітації, колізій, відштовхування. Співпрацює з Object Manager і забезпечує реалістичну поведінку в сцені.

Asset Importer: Надає функції для імпорту зовнішніх 3D-моделей та ресурсів з таких форматів як .fbx, .obj, .gltf. Забезпечує правильну інтеграцію моделей у сцену з усіма необхідними параметрами.

Analytics & Logging: Компонент для збору статистики, логів користувацьких дій та системних подій. Забезпечує аудит, налагодження та аналітику продуктивності або поведінки користувача.

3 Розробка інформаційно та програмного забезпечення

3.1 Система управління інформаційною базою

Інформаційна база, що використовується в системі моделювання об’єктів у Unity, є сукупністю структурованих даних, необхідних для створення, зберігання, обробки та візуалізації тривимірних об’єктів, а також для збереження їх властивостей, взаємодій і сценаріїв поведінки у сцені.

На відміну від класичних реляційних баз даних, дана інформаційна база реалізована переважно у вигляді серіалізованих об’єктів, структурованих файлів ресурсів та внутрішніх механізмів збереження стану Unity. Усі дані зберігаються у вигляді асоційованих сутностей, які логічно поділені на такі основні категорії:

1. Основні складові інформаційної бази:

- 3D-об’єкти (Object3D)

Містять атрибути: унікальний ідентифікатор, назва, тип, положення, ротація, масштаб, стан, взаємодії. Ці дані потрібні для відображення та взаємодії з об'єктами в сцені.

- Сцени (Scene)

Містять набір об'єктів та їхній просторовий розподіл, а також метадані про сцену (назва, опис, параметри освітлення, активність об'єктів).

- Типи об’єктів (ObjectType)

Класифікують об’єкти за роллю: статичні, інтерфейсні, взаємодіючі тощо.

- Властивості об’єктів (Property)

Індивідуальні налаштування об'єкта (наприклад, швидкість, вага, твердість, колір), які можуть бути змінені у реальному часі.

- Взаємодії (Interaction)

Дані про події, які ініціює користувач (тип взаємодії, час, цільовий об’єкт).

- Користувачі (User)

У системі зберігаються базові дані про користувача або гравця, який здійснює взаємодію (роль, дії, історія взаємодій).

2. Форми зберігання даних:

- ScriptableObjects — для конфігурацій типів об'єктів, матеріалів;
- JSON/XML — для експорту/імпорту сцени або властивостей;
- Prefab-об’єкти — для збереження стану 3D-моделей з усіма компонентами;

3. Призначення інформаційної бази:

Інформаційна база забезпечує централізоване управління даними, що використовуються при:

- генерації сцени,
- ініціалізації об’єктів,
- симуляції фізичних процесів,
- збереженні та відновленні стану моделі.

3.1.1 Причини вибору інформаційної бази

У процесі розробки системи моделювання об’єктів у середовищі Unity було прийнято рішення використовувати інформаційну базу, реалізовану у вигляді серіалізованих об’єктів (ScriptableObjects), JSON-структур та вбудованих механізмів управління ресурсами Unity. Такий підхід обумовлений низкою технічних, функціональних та архітектурних переваг, які відповідають специфіці проєкту.

Основні причини вибору:

1. Природна інтеграція з Unity

- ScriptableObject та Prefab є стандартними засобами збереження даних у середовищі Unity.
- Це дозволяє безпосередньо працювати з даними в редакторі, не вдаючись до складних налаштувань сторонніх баз даних.

2. Гнучкість у структурі зберігання

- JSON-формат дає змогу легко експортувати, редагувати або імпортувати сцени та властивості об’єктів.
- Такий підхід дозволяє зберігати як прості типи даних (string, float), так і складні ієрархії (списки об’єктів, вкладені властивості).

3. Мінімальна потреба у зовнішніх СУБД

- Оскільки система орієнтована на локальне моделювання об’єктів, використання класичної реляційної бази (наприклад, MySQL, PostgreSQL) є надлишковим.
- Всі дані ефективно зберігаються у вигляді активів проєкту (Assets), що забезпечує швидкий доступ без надмірної інфраструктури.

4. Підтримка серіалізації та десеріалізації

- Unity автоматично підтримує збереження стану об’єктів, що спрощує управління інформаційною базою.
- За допомогою JsonUtility або Newtonsoft.Json реалізується простий імпорт/експорт конфігурацій.

5. Масштабованість для подальшого розвитку

- Обрана структура дозволяє в майбутньому інтегрувати сторонні бази (SQLite, Firebase) без значних змін у логіці системи.
- Дані зберігаються у форматі, придатному до розширення: наприклад, можна додавати нові властивості об’єктів без порушення цілісності даних.

3.2 Розробка інформаційної бази

Інформаційна база є основою будь-якої програмної системи, що працює з обробкою, зберіганням і доступом до даних. У контексті дипломної роботи, яка присвячена моделюванню об'єктів у середовищі розробки Unity, інформаційна база відіграє ключову роль у збереженні структури даних, взаємозв'язків між елементами сцени, властивостей об'єктів та результатів взаємодії користувача з програмою.

Розробка інформаційної бази передбачає створення логічної та фізичної моделі даних, визначення основних таблиць, зв'язків між ними, а також реалізацію механізмів доступу до даних. Важливою вимогою є дотримання принципів нормалізації, що забезпечує цілісність, узгодженість та ефективність обробки даних.

У процесі створення бази даних враховуються специфіка програмного продукту, функціональні потреби користувача та архітектура майбутньої системи. Обирається оптимальна система керування базами даних (СКБД), яка дозволяє ефективно інтегрувати інформаційну базу з Unity, забезпечити надійне зберігання даних та легкий доступ до них через програмну логіку.

3.2.1 Функціональні ролі таблиць

Фізична модель даних для мого проекту зображена на (рис 10).

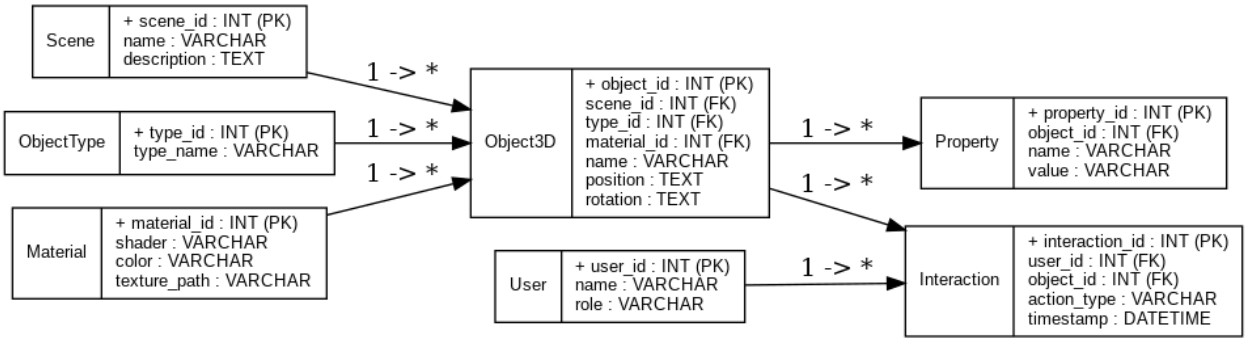

рис 10 Фізична модель даних

1. User (Користувач)

• Роль у системі: Таблиця User зберігає інформацію про осіб, що взаємодіють із 3D-системою: розробників, тестувальників або кінцевих користувачів. Вона використовується для ідентифікації користувачів, розподілу ролей (наприклад, "гравець" або "редактор сцени") та відстеження історії взаємодій з об’єктами.

• Приклади використання: Авторизація користувача, запис його дій, контроль доступу до об'єктів або сцен.

2. Scene (Сцена)

• Роль у системі: Scene — це основна віртуальна площина, де розміщуються 3D-об’єкти. Таблиця використовується для зберігання метаданих про кожну сцену: її унікальний ідентифікатор, назву, опис.

• Приклади використання: Завантаження сцени при старті проєкту, збереження конфігурації сцени, навігація між сценами.

3. Object3D (3D-об’єкт)

• Роль у системі: Центральна таблиця, що містить інформацію про кожен об’єкт у сцені. Зберігає такі дані як позиція, тип, матеріал та належність до конкретної сцени.

• Приклади використання: Завантаження 3D-моделей у сцену, оновлення атрибутів об’єкта (позиція, текстура), виведення об'єктів у редакторі.

4. ObjectType (Тип об’єкта)

• Роль у системі: Забезпечує класифікацію об’єктів: статичний, динамічний, колізійний, декорація тощо. Дозволяє відокремлювати логіку обробки для різних категорій.

• Приклади використання: Визначення поведінки об’єкта в сцені, фільтрація об’єктів у редакторі.

5. Material (Матеріал)

• Роль у системі: Зберігає дані про візуальне оформлення об’єктів, включаючи колір, шейдер, текстуру.

• Приклади використання: Призначення матеріалу об'єкту, оновлення текстури в сцені, налаштування PBR-матеріалів.

6. Property (Властивість)

• Роль у системі: Таблиця Property дозволяє зберігати динамічні або додаткові параметри об’єктів, наприклад: швидкість, стійкість, статус, стан.

• Приклади використання: Додавання властивості "руйнується" до об’єкта, зчитування рівня міцності об’єкта при симуляції.

7. Interaction (Взаємодія)

• Роль у системі: Фіксує події, ініційовані користувачем щодо конкретного об’єкта. Містить тип взаємодії (наприклад, “клік”, “удар”), час події та цільовий об’єкт.

• Приклади використання: Відображення історії дій, запуск анімацій, журнал подій для налагодження.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментарію є одним із ключових етапів при розробці прикладного програмного забезпечення, оскільки саме набір технологій, мов програмування, бібліотек і середовищ розробки визначає ефективність реалізації поставлених задач, продуктивність системи та зручність її супроводу.

У межах даної дипломної роботи, спрямованої на створення інтерактивної системи моделювання об’єктів у реальному часі, особливу увагу приділено інструментам, що підтримують 3D-графіку, фізичні симуляції, обробку подій користувача та збереження інформації про об’єкти сцени.

Під час вибору програмного інструментарію враховувалися такі критерії:

- Сумісність із вимогами проекту (підтримка 3D-моделювання, фізики, UI);
- Продуктивність і гнучкість середовища розробки;
- Можливість масштабування та повторного використання компонентів.

У цьому розділі наведено обґрунтування вибору основних технологічних засобів, зокрема рушія Unity, мови C#, що використовуються для реалізації функціональних компонентів проєкту. Отже розглянемо їх.

3.3.1 Мова програмування C#

C# — це об’єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона є однією з основних мов, що підтримується ігровим рушієм Unity, і широко застосовується для розробки як

клієнтських, так і серверних додатків. У контексті дипломної роботи C# використовується як основна мова для реалізації логіки моделювання об'єктів, обробки взаємодій користувача, фізичних обчислень та серіалізації даних.

Причини вибору C#:

1. Повна підтримка в Unity

- C# є стандартною мовою програмування у середовищі Unity. Усі скрипти, компоненти та взаємодія з об’єктами реалізуються саме за допомогою C#.

2. Об’єктно-орієнтований підхід

- Підтримка класів, спадкування, інтерфейсів, подій та делегатів дозволяє будувати чітку та масштабовану архітектуру програмного забезпечення.

3. Безпечне управління пам’яттю

- Автоматичне збирання сміття у C# знижує ризик витоків пам’яті, що особливо важливо в реальному часі під час моделювання сцен.

4. Простота та наочність синтаксису

- Мова має зручний та зрозумілий синтаксис, що спрощує розробку, тестування та обслуговування коду.

5. Широка екосистема

- Велика кількість бібліотек, документації, навчальних матеріалів і плагінів спрощує реалізацію складної функціональності (фізика, UI, серіалізація, мережа тощо).

3.3.2 Unity

Unity це потужний кросплатформенний рушій розробки ігор та інтерактивних 3D-додатків, який широко використовується в ігровій індустрії, віртуальній та доповненій реальності, симуляціях, архітектурній візуалізації й освітніх проєктах. У контексті даної дипломної роботи Unity використовується як основне середовище для реалізації системи моделювання об’єктів у реальному часі.

Причини вибору Unity:

1. Підтримка 3D-графіки та фізики в реальному часі

- Unity має вбудовані модулі рендерингу, освітлення, а також фізичний рушій (Unity Physics / NVIDIA PhysX), що дозволяє легко реалізовувати складні взаємодії об’єктів та симуляції.

2. Зручний редактор сцен

- Інтуїтивно зрозумілий інтерфейс дозволяє швидко створювати, переміщувати та налаштовувати об’єкти без потреби написання коду на перших етапах моделювання.

3. Інтеграція з мовою C#

- Unity повністю підтримує C# як основну мову програмування для написання скриптів і взаємодії з компонентами.

3.3.3 Visual Studio

Visual Studio — це інтегроване середовище розробки, яке забезпечує повний цикл створення програмного забезпечення: від написання коду до його перевірки, компіляції та підготовки до розповсюдження. Це потужний багатофункціональний інструмент із багатовіконним інтерфейсом, який охоплює всі аспекти програмування.

Visual Studio від Microsoft є одним із найпопулярніших середовищ серед розробників, і компанія регулярно оновлює його — нові версії випускаються щороку або раз на два роки.

Розробка інтерфейсів користувача в Visual Studio відбувається за допомогою вбудованого Конструктора форм, де можна легко додавати візуальні елементи із панелі інструментів. Налаштування властивостей компонентів здійснюється через спеціальне вікно властивостей. Хоча інтерфейс англомовний, він достатньо зручний і зрозумілий навіть для початківців.

Середовище надає велику кількість довідкових матеріалів і навчальних ресурсів онлайн, а також постійно оновлюється, що робить його актуальним та зручним для сучасних розробників. Visual Studio також містить набір

бібліотек, які можна підключати за потреби, що дозволяє створювати масштабні, повнофункціональні програмні продукти.

Таким чином, Visual Studio є професійним інструментом, який підходить як для початківців, так і для досвідчених розробників, і дозволяє реалізовувати найрізноманітніші проєкти.

Причини вибору Visual Studio:

1. Офіційна інтеграція з Unity

Visual Studio є рекомендованим для Unity і підтримується безпосередньо через Unity Hub. Інтеграція дозволяє відкривати проєкти Unity та автоматично пов’язувати скрипти зі сценами.

2. Розширені можливості автодоповнення (IntelliSense)

Завдяки IntelliSense, Visual Studio підказує синтаксис, типи змінних, методи та документацію, що суттєво пришвидшує написання коду та знижує ймовірність помилок.

3. Потужна система налагодження (debugging)

Visual studio дозволяє запускати налагодження Unity-проєкту прямо з редактора, ставити точки зупинки, переглядати значення змінних у реальному часі та аналізувати стек викликів.

4. Підтримка плагінів та розширень

Visual Studio має широку екосистему розширень, які дозволяють підключати додаткові засоби аналізу коду, шаблони, генератори коду тощо.

3.3.4 Принцип роботи у середовищі розробки програм

Принцип роботи Visual Studio полягає в об'єднанні редактора коду, компілятора, засобів управління проектами, системи контролю версій, а також інструментів налагодження в єдиному середовищі. Після створення проєкту в Unity, середовище автоматично інтегрується з Visual Studio, дозволяючи відкривати й редагувати C#-скрипти, зберігати зміни та миттєво повертатися до Unity для запуску гри чи симуляції.

4 Рекомендації щодо впровадження та експлуатації системи

4.1 Тестування системи

Тестування програмного забезпечення — це процес перевірки, оцінки та аналізу роботи програмного продукту з метою виявлення помилок, дефектів та відхилень від очікуваної поведінки. Тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення і забезпечує контроль якості на всіх етапах проєкту.

Основна суть тестування полягає у створенні умов, що дозволяють зімітувати дії користувача, та перевірці того, чи працює система відповідно до заданих функціональних і нефункціональних вимог.

Метою тестування є перевірка працездатності, стабільності та відповідності програмного продукту вимогам, що були визначені на етапі проєктування. Тестування дозволяє виявити помилки, недоліки або відхилення в роботі системи ще до її впровадження, що знижує ризики, пов'язані з використанням ненадійного або некоректного програмного забезпечення.

Основні завдання тестування:

- Перевірка функціональних можливостей системи відповідно до технічного завдання;
- Виявлення помилок логіки або некоректної взаємодії між компонентами;
- Оцінка продуктивності системи в умовах навантаження;
- Перевірка стійкості до некоректного введення даних або помилкових дій користувача;
- Тестування інтерфейсу користувача на зручність і відповідність очікуванням.

Що входить до тестування в проєкті:

- Перевірка коректного створення, переміщення та збереження об'єктів сцени;
- Тестування взаємодії користувача з UI (кнопки, меню, налаштування);
- Перевірка фізичної поведінки об'єктів при моделюванні;
- Перевірка коректності серіалізації даних у JSON та повторного завантаження;
- Перевірка відповідності результатів моделювання очікуваним даним.

Одним із ключових підходів тестування є модульне тестування, яке передбачає перевірку окремих функцій, класів або компонентів у ізоляції. Цей метод дозволяє швидко виявляти помилки на рівні логіки реалізації та забезпечити коректність роботи кожного елемента системи.

Наступним важливим методом є функціональне тестування, яке зосереджується на перевірці відповідності програмного продукту визначеним вимогам користувача. У цьому контексті перевіряється, чи система виконує всі необхідні дії: створення об’єктів, їх переміщення, збереження сцени, імпорт моделей тощо.

Інтеграційне тестування застосовується для перевірки взаємодії між різними модулями системи, зокрема між редактором сцени, логікою взаємодії та фізичним модулем. Такий підхід дозволяє виявити помилки, що виникають не на рівні окремих класів, а при спільному використанні кількох компонентів.

У процесі тестування застосовується також метод «чорного ящика», який передбачає перевірку роботи системи без знання її внутрішньої структури. З боку кінцевого користувача проводяться дії в інтерфейсі: створення нових сцен, взаємодія з об’єктами — з подальшим аналізом відповідності результатів очікуваним.

Особливу увагу було приділють UI/UX тестуванню, тобто перевірці зручності, логічності та інтуїтивності інтерфейсу користувача. Завдяки цьому

вдається оптимізувати розміщення елементів управління, покращити сценарії взаємодії та усунути неочевидні помилки інтерфейсу.

Також виконується обмежене навантажувальне тестування, яке дозволяє оцінити стабільність системи при значному числі одночасних об’єктів або при повторному збереженні великих сцен. Це дозволить гарантувати стабільну роботу системи навіть у складних умовах.

4.2 Вимоги до апаратного та програмного забезпечення

Розробка та запуск програмного забезпечення для моделювання об’єктів у середовищі Unity потребують дотримання певних вимог до обчислювальних ресурсів і програмних інструментів. Оскільки система реалізується на основі ігрового рушія Unity з використанням мови програмування C#, важливо забезпечити сумісність усіх компонентів середовища розробки, стабільну роботу в процесі моделювання та рендерингу, а також підтримку графічного інтерфейсу користувача.

У цьому підпункті визначено перелік мінімальних та рекомендованих апаратних характеристик, а також необхідне програмне забезпечення, яке повинне бути встановлене для ефективної роботи з програмною системою. Це дозволяє гарантувати, що користувачі зможуть коректно встановити, налаштувати та використовувати застосунок, а також забезпечити стабільне виконання основних функцій на різних конфігураціях ПК.

Під час розробки враховувалася підтримка сучасних операційних систем, наявність середовища розробки, сумісність з бібліотеками Unity, а також наявність драйверів для роботи з графічним прискоренням, що критично важливо для комфортної роботи з тривимірними об’єктами в реальному часі.

Діаграма розгортання зображена на (Рис 11), вона допоможе нам зрозуміти як буде розгортатися наша система.

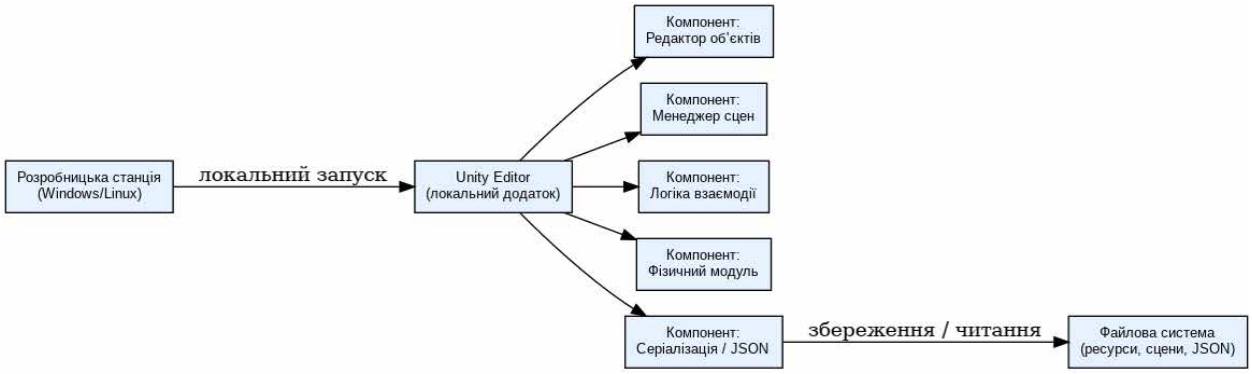

Рис 11 Діаграма розгортання

4.2.1 Апаратні вимоги. вимоги стосуються кінцевих користувачів платформи, які будуть використовувати додаток.

Мінімальні вимоги:

- Процесор : Intel Core i3 або AMD Ryzen 3
- Оперативна пам’ять : 4 ГБ
- Графічна карта : Intel HD Graphics 5000 або аналогічна вбудована графіка
- Жорсткий диск: 3–5 ГБ вільного простору
- Операційна система: Windows 10 (64-bit)

Рекомендовані вимоги:

- Процесор: Intel Core i5 або AMD Ryzen 5
- Оперативна пам’ять: 8–16 ГБ
- Графічна карта: NVIDIA GeForce GTX 1050 Ti або AMD RX 570
- Накопичувач: SSD з мінімум 5 ГБ вільного простору
- Операційна система: Windows 10/11

Обов’язкові компоненти:

- .NET Framework 4.8 або вище
- Visual C++ Redistributable 2015–2022

4.2.2 Додаткові вимоги для розробника

Для розробників, які будуть працювати над платформою, важливо мати належні інструменти та середовища для розробки.

Основне програмне забезпечення:

- Unity Hub – керування проєктами та встановлення редакторів Unity;
- Unity Editor (2021.3 LTS або новіший) – середовище створення 3D-сцен, скриптів, фізичних симуляцій;
- Visual Studio 2022 / Visual Studio Code – написання коду на C#, інтеграція з Unity;

Висновок

У процесі виконання дипломної роботи було здійснено повноцінне дослідження, проектування та реалізація програмного рішення з моделювання об’єктів у середовищі Unity, що дозволяє інтерактивно створювати, відображати та взаємодіяти з віртуальними 3D-об’єктами. Обрана тема є актуальною, оскільки моделювання у 3D-середовищах усе ширше використовується в таких галузях, як ігрова індустрія, освіта, архітектура, симуляційні системи та віртуальна реальність.

У рамках роботи було проведено системний аналіз предметної області, виявлено функціональні та нефункціональні вимоги до майбутнього програмного продукту, сформовано сценарії його використання, розроблено відповідну архітектуру та логіку роботи системи. Побудовано низку UML-діаграм — класів, послідовностей, акторів, кооперацій, компонентів і топології, які формують основу архітектурної моделі та демонструють структуру і взаємозв’язки компонентів системи.

Особливу увагу було приділено проєктуванню інформаційної бази, яка відображає логічну і фізичну модель даних, що забезпечує ефективне зберігання та обробку інформації в системі. Застосовані технології відповідають сучасним вимогам до розробки інтерактивних застосунків, зокрема: Unity Editor, C#, Visual Studio, Blender для 3D-моделювання та відповідні засоби тестування.

У результаті виконаної роботи:

- реалізовано прототип програмного продукту, що дозволяє моделювати об’єкти з використанням тривимірної графіки та елементів фізики;
- доведено працездатність та функціональність розробленої системи на основі тестування;
- сформульовано вимоги до апаратного та програмного забезпечення як для розробника, так і для кінцевого користувача;

- запропоновано оптимізовану топологію розміщення компонентів для роботи у локальному середовищі.

Практичне значення розробки полягає в тому, що отриманий програмний продукт може бути використаний як основа для створення навчальних симуляторів, ігор або систем візуалізації в інших сферах.

Таким чином, мета дипломної роботи досягнута, а її результати можуть бути надалі розвинені шляхом інтеграції додаткових функціональних модулів, розширення фізичних симуляцій або створення повноцінного продукту з графічним інтерфейсом користувача.

Використані джерела

1) Unity Technologies. Unity User Manual – офіційна документація до Unity. https://docs.unity3d.com/Manual/index.html
2) Unity Technologies. Unity Scripting API – офіційна довідка з програмування в Unity на C#. https://docs.unity3d.com/ScriptReference/
3) Microsoft. Документація по C# – офіційний ресурс з описом синтаксису, парадигм та прикладів. https://learn.microsoft.com/en-us/dotnet/csharp/
4) Wazura, M., & Siuta-Tokarska, B. (2021). The use of Unity in 3D modeling education. https://doi.org/10.1007/978-3-030-77877-7_16
5) Pluralsight. Unity Game Development Tutorials – навчальний портал з курсами для Unity. https://www.pluralsight.com/paths/unity-game-dev
6) Udemy. Complete C# Unity Game Developer 2D/3D – приклад курсу, що охоплює моделювання і програмування. https://www.udemy.com/course/unitycourse/
7) Blender Foundation. Blender Documentation – офіційна довідка для роботи з 3D-моделями. https://docs.blender.org/manual/en/latest/
8) Gamasutra (Game Developer). Статті про фізику та моделювання в іграх. https://www.gamedeveloper.com/
9) Visual Studio Code. "Visual Studio Code Documentation." Available at: https://code.visualstudio.com/docs
10) Joseph Hocking. Unity in Action: Multiplatform Game Development in C# – Manning Publications, 2022. https://www.manning.com/books/unity-in-action-third-edition
11) Microsoft Learn. C# Guide – Офіційний гід для розробників від Microsoft. https://learn.microsoft.com/en-us/dotnet/csharp/
12) DotNetPerls. Приклади оптимізованого коду на C# https://www.dotnetperls.com/

13) GitHub – C# Examples and Libraries. Репозиторії з прикладами C#-коду, які часто використовуються з Unity https://github.com/search?q=c%23+examples

14) .NET Blog (Microsoft). Офіційний блог по C#, .NET, компіляторам та екосистемі https://devblogs.microsoft.com/dotnet/