Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ФІЗИКИ РУХУ ОБ'ЄКТІВ У 2D ПРОСТОРІ

Виконав: студент 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Максим ТАРАСОВСЬКИЙ
(прізвище та ініціали)

Керівник: д.т.н., доц. каф. АІІТ
Володимир ГАРМАШ
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., доц. кафедри АІІТ
Ярослав КУЛИК
(прізвище та ініціали)

«05» червня 2025 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Андрій ЯРОВИЙ
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» ТРАВНЯ 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Тарасовському Максиму Анатолійовичу

1. Тема роботи: Розробка програмного забезпечення для реалізації фізики руху об'єктів у 2D просторі
керівник роботи: Гармаш Володимир Володимирович, доц. каф. АІІТ
затверджені наказом вищого навчального закладу "20" березня 2025 року №97
2. Строк подання студентом роботи 30 ТРАВНЯ 2025 Р.
3. Вихідні дані до роботи :
Мова програмування: C#;
Середовище розробки: Unity;
4. Зміст текстової частини (перелік питань, які потрібно розробити):
Вступ; Обґрунтування доцільності розробки програмного забезпечення; Аналіз предметної області; Обґрунтування вибору інструментів технічної реалізації; Розробка алгоритму для програмного забезпечення; Формулювання мети роботи, об'єкта та задач подальшого дослідження; Висновки; Список використаних джерел; додатки.
5. Перелік графічного матеріалу:
Блок-схеми основних алгоритмів;

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Володимир ГАРМАШ, доц. каф. АІІТ		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Дослідження предметної області	05.05.25	07.05.25
2	Визначення мети, об'єкту, предмету і задач подальшого дослідження	08.05.25	12.05.25
3	Обгрунтування вибору інструментів технічної реалізації	13.05.25	17.05.25
4	Розробка алгоритму	18.05.25	25.05.25
5	Реалізація програмного модулю	26.05.25	29.05.25
6	Тестування програмного модулю	30.05.25	01.06.25
7	Оформлення пояснювальної записки	02.06.25	04.06.25

Студент _____ Тарасовський М.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гармаш В. В.
(підпис) (прізвище та ініціал

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 67 сторінок формату А4, містить 13 рисунків, список використаних джерел налічує 21 найменування.

У бакалаврській роботі розроблено алгоритм та програмна архітектура симуляційного забезпечення для для реалізації фізики руху об'єктів у 2D просторі. Реалізована фізична взаємодія між об'єктами а також досліджено та проаналізовано можливі застосування програмного забезпечення у різних галузях. Програмне забезпечення реалізовано на сучасному рушії Unity та мові програмування C#. Результатом роботи є симуляційне програмне забезпечення яке може використовуватись для навчання або у галузі відеоігор.

Ключові слова: Симуляція, 2D фізика, програмне забезпечення, гейміфікація.

ABSTRACT

The bachelor's thesis consists of 67 pages of A4 format, contains 13 figures, and the list of references includes 21 titles.

In the bachelor's thesis, an algorithm and software architecture of simulation software for the realization of the physics of motion of objects in 2D space was developed. The physical interaction between objects is realized, and possible applications of the software in various fields are investigated and analyzed. The software is implemented on the modern Unity engine and C# programming language. The result of the work is a simulation software that can be used for training or in the field of video games.

Keywords: Simulation, 2D physics, software, gamification.

ЗМІСТ

ВСТУП……4

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ……...6

1.1 Фізичні симуляції у програмуванні…….6

1.2 Основи фізики руху в 2D-просторі……..8

1.3 Застосування симуляцій у ігровій та науковій індустрії…….11

1.4 Висновок до розділу 1…….24

2 ОПИС ТЕХНІЧНОЇ РЕАЛІЗАЦІЇ ТА РОЗРОБКА АЛГОРИТМУ……..……26

2.1 Обгрунтування вибору інструментів технічної реалізації……..26

2.2 Опис використаної технології……27

2.3 Опис алгоритму у вигляді схеми……29

2.4 Висновок до розділу 2…….31

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ФІЗИКИ РУХУ ОБ’ЄКТІВ У 2D ПРОСТОРІ……33

3.1 Обґрунтування вибору мови програмування……33

3.2 Програмна реалізація програмного забезпечення……35

3.3 Тестування роботи програмного забезпечення для реалізації фізики руху об'єктів у 2D просторі……..39

3.4 Висновок до розділу 3……40

ВИСНОВКИ…….42

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……44

ДОДАТОК А. (Обов'язковий) Протокол перевірки кваліфікаційної роботи….46

ДОДАТОК Б. (Обов'язковий) Лістинг програмного коду……..….47

ДОДАТОК В. (Обов'язковий) Ілюстративна частина..……...52

ДОДАТОК В. (Довідниковий) Інструкція користувача..…….63

ВСТУП

Актуальність дослідження.

У сучасному цифровому середовищі 2D-графіка та фізичні симуляції залишаються одним із ключових напрямів розвитку в галузі програмування, геймдизайну, освіти та візуалізаційних технологій. Незважаючи на домінування 3D-проєктів у високобюджетних продуктах, двовимірні ігри, симулятори та навчальні додатки зберігають високу популярність завдяки своїй доступності, простоті сприйняття, ефективності реалізації та можливості точного моделювання фізичних процесів.

Особливої актуальності набувають проєкти, орієнтовані на реалістичну поведінку об’єктів, що рухаються та взаємодіють відповідно до законів класичної механіки. Завдяки розвитку середовищ на кшталт Unity, незалежні розробники отримали у своє розпорядження інструменти, які дозволяють створювати гнучкі та візуально привабливі фізичні симуляції навіть без наявності масштабних ресурсів. Це сприяє появі численних інді-проєктів та навчальних рішень, у яких моделювання фізики є основою геймплею, навчального експерименту або демонстрації.

Водночас багато існуючих реалізацій мають істотні обмеження: недостатню точність обчислень, шаблонну фізику без урахування складної динаміки, відсутність інтерактивної взаємодії користувача з симульованими об’єктами, або слабку гнучкість у налаштуванні фізичних параметрів. Такі недоліки знижують інформативність і достовірність симуляції, обмежують застосування подібного програмного забезпечення в освітніх чи наукових цілях, і негативно впливають на загальний досвід користувача.

Актуальність теми підсилюється потребою в інтерактивних навчальних засобах, які дозволяють не лише спостерігати за фізичними процесами, а й експериментувати з параметрами — масою, швидкістю, тертям, кутом удару, тощо. Це відкриває широкі можливості для розвитку логічного мислення,

інтуїтивного розуміння законів фізики та креативного застосування програмного моделювання у неігрових контекстах.

Запропоноване в межах цієї роботи програмне забезпечення дозволяє реалізувати реалістичну симуляцію руху тіл у 2D-просторі з дотриманням основ класичної механіки, обробкою зіткнень, застосуванням зовнішніх сил, а також з інтерактивним керуванням об'єктами з боку користувача. Розробка орієнтована як на подальше вдосконалення навичок у галузі фізичного моделювання, так і на створення бази для проєктів у сфері ігор, освіти, демонстрацій або навіть наукових візуалізацій.

Метою дослідження є підвищення якості та достовірності моделювання фізичних процесів у двовимірному середовищі через розробку та впровадження інтерактивного програмного продукту на базі рушія Unity. Досягти цього планується шляхом створення гнучкої системи, яка дозволяє точно і наочно відтворювати основні фізичні явища — зокрема рух тіл під впливом сили тяжіння, тертя, імпульсу та обробку зіткнень у реальному часі.

Об’єктом дослідження є процес розробки інтерактивного програмного забезпечення, що імітує фізику руху об’єктів у двовимірному просторі.

Предметом дослідження є методи, алгоритми та технічні рішення, що забезпечують побудову функціонального, стабільного та інтуїтивно зрозумілого середовища для моделювання фізики у 2D-просторі. Зокрема — способи реалізації гравітації, пружних зіткнень, взаємодії тіл, їх параметризації, а також інтеграції користувацького керування у фізичну модель.

Задачі дослідження:

1) Обґрунтувати доцільність розробки програмного забезпечення.
2) Аналізувати предметну область.
3) Обґрунтувати вибір інструментів технічної реалізації.
4) Розробити алгоритм для програмного забезпечення.
5) Розробити інструкцію користувача програмного модулю.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Фізичні симуляції у програмуванні

Фізичне моделювання у комп'ютерному програмуванні – це процес створення математичних моделей фізичних явищ та їх відтворення за допомогою програмного коду. Це дозволяє імітувати реальні або гіпотетичні процеси у середовищі, яке повністю підконтрольне та піддається змінам.

На практиці фізичне моделювання активно застосовується у наступних сферах:

Ігрова індустрія — для реалістичного відтворення рухів персонажів, транспорту, оточуючих об'єктів (каміння, вода, повітря тощо).

Навчальні програми — демонстрація законів механіки, гравітації, енергії, зіткнень та інших явищ у доступній формі.

Інженерія та CAD — візуалізація механічної поведінки конструкцій, аналіз навантажень та моделювання руху частин механізмів.

Фізика та хімія — симуляція мікроскопічних частинок, молекулярної динаміки, термодинаміки та інше.

Кінематограф і візуальні ефекти — створення спецефектів, що імітують падіння, вибухи, дим, рідини, зіткнення об'єктів.

Основна перевага фізичного моделювання — можливість експериментувати у віртуальному середовищі, не витрачаючи ресурси на реальні фізичні експерименти. Це особливо актуально при роботі з системами, які можуть бути дорогими, складними або небезпечними у реальному житті.

Фізичне моделювання складається з кількох ключових етапів:

Побудова фізичної моделі — формалізація об'єкта або явища у вигляді системи рівнянь [1].

Вибір числового методу — визначення способу обчислення змін параметрів системи з плином часу.

Реалізація у програмному коді — імплементація логіки моделі у вигляді програмних алгоритмів.

Візуалізація результату — відображення зміни стану об'єктів у вигляді графіки або чисел.

Для двовимірного простору (2D) моделювання, зазвичай включає в себе рух тіл по площині під впливом сили тяжіння, сили тертя, реакції при зіткненнях, пружності та інерції. Об’єкти описуються як точки, кола, прямокутники або інші прості геометричні фігури, кожна з яких має власні фізичні властивості — масу, швидкість, прискорення тощо.

Також важливо враховувати:

Крок симуляції (delta time) — впливає на точність моделювання. Менший крок забезпечує більшу точність, але вимагає більше ресурсів.

Стабільність алгоритмів — неякісно реалізовані симуляції можуть накопичувати похибки й призводити до некоректної поведінки.

Оптимізацію — особливо критично при моделюванні великої кількості об'єктів.

Серед популярних бібліотек та рушіїв для реалізації фізики в 2D-програмуванні варто виділити:

Box2D — фізичний рушій з відкритим кодом, який реалізує гравітацію, зіткнення, фіксацію тіл, пружність тощо.

Pymunk — обгортка для Python, яка дозволяє реалізувати фізику на базі рушія Chipmunk.

Matter.js — фізичний рушій для JavaScript.

Godot Physics 2D / Unity Physics 2D — вбудовані інструменти популярних ігрових рушіїв [2].

Отже, фізичне моделювання в програмуванні — потужний інструмент, що відкриває широкі можливості для створення інтерактивних, динамічних та реалістичних систем. Його важливість постійно зростає разом із розвитком обчислювальної техніки, яка дає змогу реалізовувати все складніші моделі в реальному часі[2].

1.2 Основи фізики руху в 2D-просторі

Рух предметів у двовимірному просторі є ключовою темою класичної механіки. У програмному забезпеченні, призначеному для моделювання фізичних явищ, цей рух визначається математичними моделями, що враховують координати, швидкість, прискорення та зовнішні впливи. Моделювання такого руху забезпечує реалістичну поведінку предметів у двовимірному середовищі, що є вкрай важливим для створення інтерактивних додатків, ігор та навчальних симуляцій.

Кінематика – це розділ механіки, що описує рух об'єктів, не враховуючи причин цього руху. Основні параметри, які використовуються:

Положення (*x, y*) – координати об'єкта у просторі.

Швидкість (*vx, vy*) – вектор, що визначає напрямок та швидкість зміни положення.

Прискорення (*ax, ay*) – вектор, який описує зміну швидкості з часом.

Основні рівняння кінематики для рівномірно прискореного руху в 2D виглядають так:

$$x(t) = x_0 + u_x \cdot t\frac{1}{2}a_x \cdot t^2 \quad (1)$$

$$y(t) = y_0 + u_y \cdot t\frac{1}{2}a_y \cdot t^2 \quad (2)$$

Динаміка розглядає причини зміни руху, тобто сили, що діють на об'єкт [3].

В основі лежить другий закон Ньютона:

Для точного відтворення поведінки об’єктів у двовимірному середовищі програмне забезпечення має враховувати низку фундаментальних сил:

Гравітація – модель постійного прискорення вниз, наприклад,

$a_y = -9.8$ м/c^2.

У візуальних симуляціях цю силу часто зменшують для уповільнення руху.

Сила тертя – виникає під час переміщення об’єкта по поверхні[3]. Вона враховується за формулою:

$$F_{\text{тертя}} = \mu \cdot N \tag{3}$$

де

μ — коефіцієнт тертя, а N — сила нормальної реакції опори.

Сила опору середовища – гальмівна сила, що залежить від швидкості руху:

$$F_{\text{опір}} = -k \cdot v \tag{4}$$

де

k — коефіцієнт опору.

Сила пружності – реалізується для тіл, які можуть деформуватися, або при використанні віртуальних пружин:

$$F_{\text{опір}} = -k \cdot x \tag{5}$$

де x — зміщення від положення рівноваги.

Імпульс при зіткненні – у симуляціях зазвичай реалізується згідно із законами збереження імпульсу та енергії:

Одним із ключових аспектів симуляції руху є виявлення зіткнень (collision detection) та реакція на них (collision response). Це особливо важливо для анімацій, ігор та систем, де об’єкти взаємодіють між собою.

Типи зіткнень у 2D:

$$m_1 v_1 + m_2 v_2 = m_1 v_1' + m_2 v_2'$$

Одним із ключових аспектів моделювання руху є виявлення зіткнень (розпізнавання колізій) та відповідна реакція на них (реагування на колізії). Це

має вирішальне значення в анімаційних проектах, відеоіграх та будь-яких системах, де об'єкти повинні взаємодіяти між собою.

- Прямокутник з прямокутником (AABB — axis-aligned bounding box)
- Коло з колом
- Прямокутник з колом
- Полігон з полігоном

Методи обробки:

- Просте відштовхування (restitution)
- Зміна вектора швидкості
- Врахування коефіцієнта пружності
- Для більшої реалістичності обробка зіткнень може враховувати масу, втрати енергії, тертя та зсув.

У програмній реалізації рух об'єкта обчислюється покроково, тобто в кожному кадрі симуляції перераховується нова позиція об'єкта на основі швидкості та прискорення [4].

Найбільш поширені методи:

Метод Ейлера:

$$v = v + a \cdot \Delta t \tag{6}$$

$$x = x + v \cdot \Delta t \tag{7}$$

Простий та швидкий, але не дуже точний.

Метод Верле (Verlet Integration) – зберігає стабільність і добре підходить для фізичних симуляцій.

Метод Рунге-Кутта – складніший, але дає високу точність.

1.3 Застосування симуляцій у ігровій та науковій індустрії

Фізичне моделювання - ключовий інструмент у сучасному програмуванні, що дозволяє відтворити комплексні реальні явища у віртуальному світі. У двовимірному просторі фізика руху об'єктів знайшла широке застосування в різних областях, таких як комп'ютерні ігри, наукове моделювання, навчальні платформи та тренажери.

У розробці ігор фізичні симуляції відіграють визначальну роль у формуванні реалістичного та динамічного ігрового світу, де гравець взаємодіє з об'єктами так, ніби вони існують насправді. Фізика в іграх не лише поліпшує зовнішній вигляд і поведінку об'єктів, а й суттєво впливає на ігровий процес, роблячи його більш захопливим [5].

Основні приклади використання фізики в іграх включають:

- Реалістичний рух персонажів та об'єктів - врахування маси, інерції, прискорення та опору середовища забезпечує природну динаміку рухів.
- Гравітація - імітація падіння, стрибків, польоту та інших відповідних явищ.
- Зіткнення та взаємодія - об'єкти можуть відскакувати, перекидатися або руйнуватися в залежності від сили удару.
- Взаємодія з оточенням - наприклад, гравець може штовхати ящики, стріляти у стіни або переміщувати предмети.
- Деструктивна фізика - руйнування об'єктів згідно з фізичними законами, прикладом є платформери чи аркади.
- Ефекти - падіння частинок, розбризкування води, рух листя на вітрі, вогонь, дим та інші візуальні ефекти.

Фізичні рушії, що використовуються в іграх:

- Box2D - один з найпоширеніших рушіїв для 2D-фізики, використовується у багатьох інді-іграх.
- Chipmunk2D / Pymunk - рушій на C з оболонкою для Python.

- Unity Physics 2D - фізичний рушій, інтегрований у Unity, заснований на Box2D.
- Godot Physics 2D - нативний рушій, що працює у середовищі Godot Engine.

Приклади ігор з 2D-фізикою:

Angry Birds - реалістичне моделювання польоту об’єктів, їх зіткнень та руйнувань.

У грі реалізовано реалістичний рух птахів та елементів конструкцій (блоків, платформ, свиней) відповідно до законів класичної механіки [6]. При запуску птаха з рогатки враховується:

- початковий кут вильоту та сила натягу (як приклад — вектор сили);
- гравітація, що діє постійно вниз;
- ефекти прискорення, сповільнення, обертання (при зіткненнях);
- залежність траєкторії від маси, швидкості та типу об’єкта.

Це дозволяє кожному рівню відчуватись унікальним і стимулює гравця експериментувати з траєкторією та силою пострілу.

Взаємодія об’єктів і зіткнення

Окрему увагу в грі приділено системі зіткнень. При ударі птахом у споруду:

- відбувається розрахунок сили імпульсу (за масою й швидкістю);
- визначається, які об’єкти руйнуються або зміщуються;
- деякі блоки можуть падати один на одного або знищувати ворогів.

Використано комбіновану фізику жорстких тіл з урахуванням матеріалів (дерево, камінь, скло), кожен із яких має власні властивості: масу, міцність, рівень пружності тощо. Внаслідок цього поведінка об’єктів стає передбачуваною, але ніколи не абсолютно однаковою.

Рисунок 1.1 – Зображення гри Angry Birds

Terraria - часткове застосування фізики для гравітації та рідин.

Гравітація в грі працює як стала сила вниз, що впливає на швидкість падіння об’єкта. Ця сила враховується під час стрибків, падіння з висоти, а також під час застосування інструментів (наприклад, крил чи гаків).
Кожен об’єкт має власну масу та габарити, що впливають на те, як він реагує на гравітацію — важчі предмети швидше падають, легші можуть зависати або плавно опускатися.

Всі твердотільні блоки у грі мають сітчасту структуру (кожен блок — окрема клітинка), що спрощує перевірку зіткнень і дозволяє об’єктам взаємодіяти із середовищем на основі координат.

Коли гравець або предмет стикається з блоками, спрацьовує система колізій, яка зупиняє об’єкт або змінює його траєкторію.

Обробка зіткнень реалізована простою, але стабільною логікою, яка дозволяє уникати "залипання" в стінах і некоректного поводження тіл навіть у складних побудовах.

Terraria також підтримує ефекти відскоку, коли, наприклад, персонаж підстрибує від пружних об’єктів або вороги відлітають після удару. Усе це моделюється за допомогою фізичних параметрів типу "штовхання", "відкату" чи "імпульсу".

Рисунок 1.2 – Зображення гри Terraria

Limbo - фізика для взаємодії з об'єктами, переміщення платформ, важільних механізмів тощо.

Гра Limbo є знаковим прикладом використання фізики у 2D-просторі як інструменту для створення атмосферного геймплею, емоційного занурення та глибокої інтерактивності.

Проєкт поєднує мінімалістичний візуальний стиль із надзвичайно точною фізичною моделлю, що дозволяє гравцю відчувати рух, вагу та небезпеку світу навколо.

Рух персонажа в Limbo моделюється з урахуванням основних принципів класичної механіки.

Герой має:

- реалістичну інерцію — він не починає і не зупиняє рух миттєво;
- гравітацію, що змушує його падати з висоти з прискоренням;
- можливість стрибути, що реалізується через вертикальну силу з імпульсом;
- відчутне тертя, що запобігає ковзанню на поверхнях.

Рух виглядає плавним, природним і водночас вразливим. Це створює особливу атмосферу напруження та крихкості, підсилюючи психологічний ефект від ігрового процесу.

У Limbo активно використовується механіка маніпулювання об’єктами. Гравець може:

- штовхати та тягнути ящики, колоди, платформи;
- пересувати вагові об’єкти для розв’язання головоломок;
- використовувати механізми, що реагують на зміну маси або положення.

Ці елементи мають власні фізичні характеристики — масу, об’єм, момент інерції — і вступають у взаємодію з іншими тілами. Наприклад, неправильно розташований ящик може впасти або перекинутись, знищивши гравця, або заблокувати шлях. Поведінка об'єктів повністю залежить від умов середовища та впливу сили з боку гравця або інших тіл.

Рисунок 1.3 – Зображення гри Limbo

World of Goo – гра ігрова механіка якої базується на створенні конструкцій з урахуванням фізичних властивостей.

Головна особливість гри — можливість створювати динамічні конструкції з вузлів і з’єднань, які реагують на вагу, напругу, нахил і зовнішні сили. Кожна кулька приєднується до інших через пружні лінії, що працюють як фізичні з’єднання. Конструкції можуть згинатися, розтягуватися, коливатись або навіть обвалюватись — усе це моделюється в реальному часі.

Кожне з’єднання має:

- власну довжину, жорсткість та еластичність;

- реакцію на навантаження — занадто довга чи асиметрична структура може зруйнуватись;
- фізичну нестабільність, що дозволяє реалізувати ефекти типу маятників, тремтіння, розгойдування.

Вага кожної кульки впливає на загальну стійкість конструкції. При розміщенні нових елементів потрібно враховувати:

- центр маси всієї структури;
- точку опори і те, чи рівномірно розподілено навантаження;
- гравітацію, яка постійно тягне конструкцію вниз.

Гравець вчиться розуміти, що таке перевантаження, момент сили та навіть принцип важеля — гра фактично навчає основам механіки без жодних формул. Це досягається через пряму симуляцію фізичних явищ.

Усі об’єкти в грі мають точно задані Collider-и, що дозволяє:

- точно моделювати зіткнення між частинами конструкції та перешкодами;
- реалізувати відштовхування, якщо об’єкт рухається з надто великою швидкістю;
- враховувати тертя по поверхнях або взаємне проковзування.

Деякі рівні включають вітряки, рідини, вогонь або змінні гравітаційні поля, що додає складності фізичній симуляції й вимагає глибшого аналізу сил у системі.

Отже, фізичне моделювання суттєво збагачує ігровий досвід та є невід’ємною частиною багатьох сучасних ігор.

Фізичні симуляції широко застосовуються в науці та техніці для аналізу та передбачення поведінки складних систем, що дозволяє уникнути необхідності проводити дорогі або ризиковані експерименти.

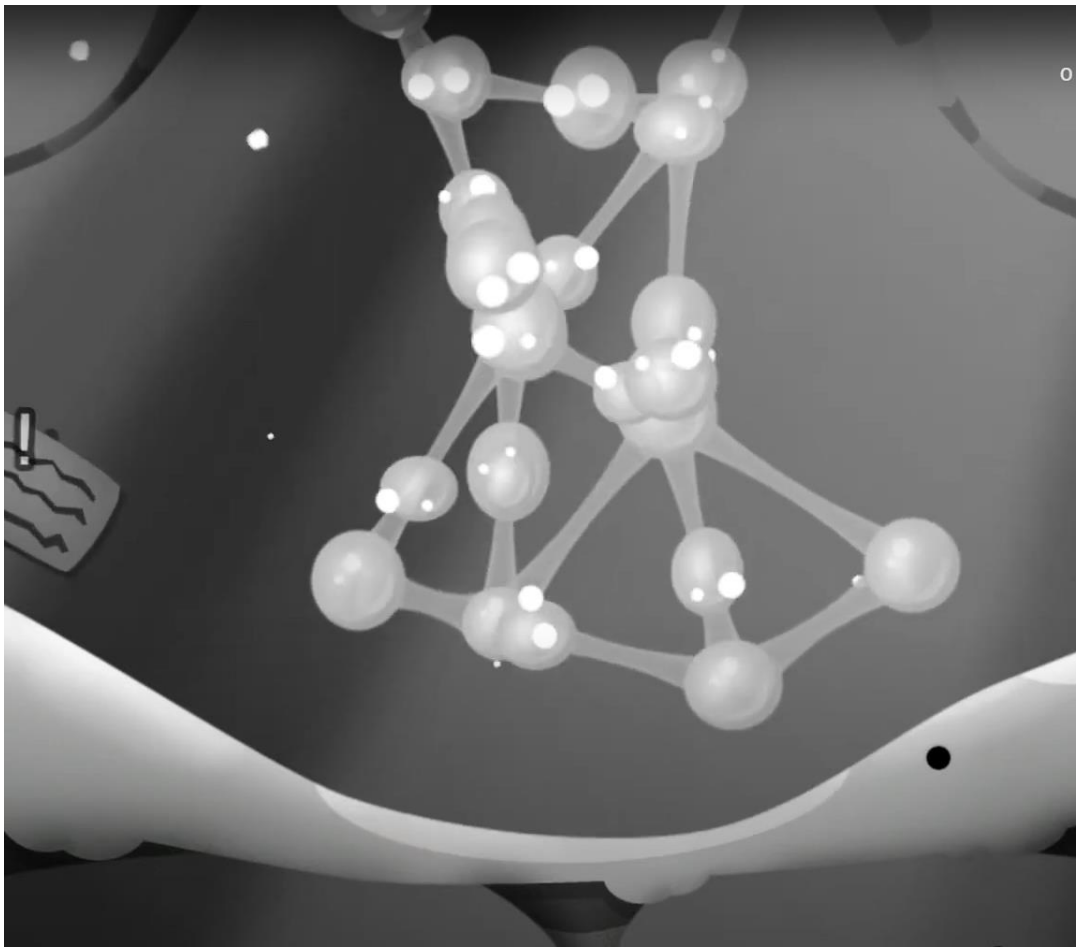

Рисунок 1.4 – Зображення гри World of Goo

Приклади застосування:

- Механіка та динаміка тіл - розрахунок траєкторій руху об'єктів, коливання систем, розрахунок навантажень та інших фізичних параметрів.
- Аеродинаміка та гідродинаміка - симуляція руху рідин або газів навколо об'єктів, наприклад, для проектування літаків чи автомобілів.
- Космічна галузь - розрахунок руху супутників, планетних систем, імітація посадок на поверхню інших об'єктів.
- Робототехніка - перевірка поведінки механізмів, маніпуляторів, навігація у робочому середовищі.
- Медична візуалізація - симуляція кровотоку, рухів м'язів або кісток під дією навантаження та інших фізичних факторів.

Симуляції дають змогу змінювати параметри моделей та миттєво отримувати результати, підвищуючи ефективність наукової роботи.

Інструменти для наукових симуляцій:

- MATLAB, Simulink - середовища для наукового моделювання та аналізу фізичних процесів.
- COMSOL Multiphysics, ANSYS - для вирішення складних фізичних та інженерних задач.
- Processing, Python + matplotlib / numpy - для візуального моделювання та обчислень у науці та освіті.

Симуляції фізики також широко використовуються в освітньому процесі, де вони сприяють візуалізації складних концепцій та кращому засвоєнню навчального матеріалу.

Застосування в освіті:

- Демонстрація законів Ньютона (сила, маса, прискорення).
- Вивчення гравітації, тертя та пружності.
- Аналіз руху маятника, снаряда, систем тіл.
- Моделювання електромагнітних процесів [8].

Приклади освітніх платформ:

PhET Interactive Simulations (University of Colorado Boulder) - інтерактивні візуальні симуляції з фізики та хімії.

Більшість симуляцій PhET реалізовані саме у двовимірному середовищі з використанням базових фізичних моделей та візуальних елементів, що робить їх актуальними прикладами для аналізу

Усі симуляції PhET мають спільні риси:

- побудовані у 2D-просторі з простою графікою;

- моделюють фізичні процеси у реальному часі (рух тіл, сили, енергія, зіткнення, електричні кола тощо);
- містять візуальні індикатори (вектори сил, швидкість, траєкторії, значення величин);
- підтримують інтерактивність — користувач може змінювати масу, швидкість, напрям сили, коефіцієнти тертя, пружності та інше.

При цьому всі взаємодії базуються на реалістичних фізичних моделях, але адаптованих до навчального формату — зручних, стабільних і візуально зрозумілих [9].

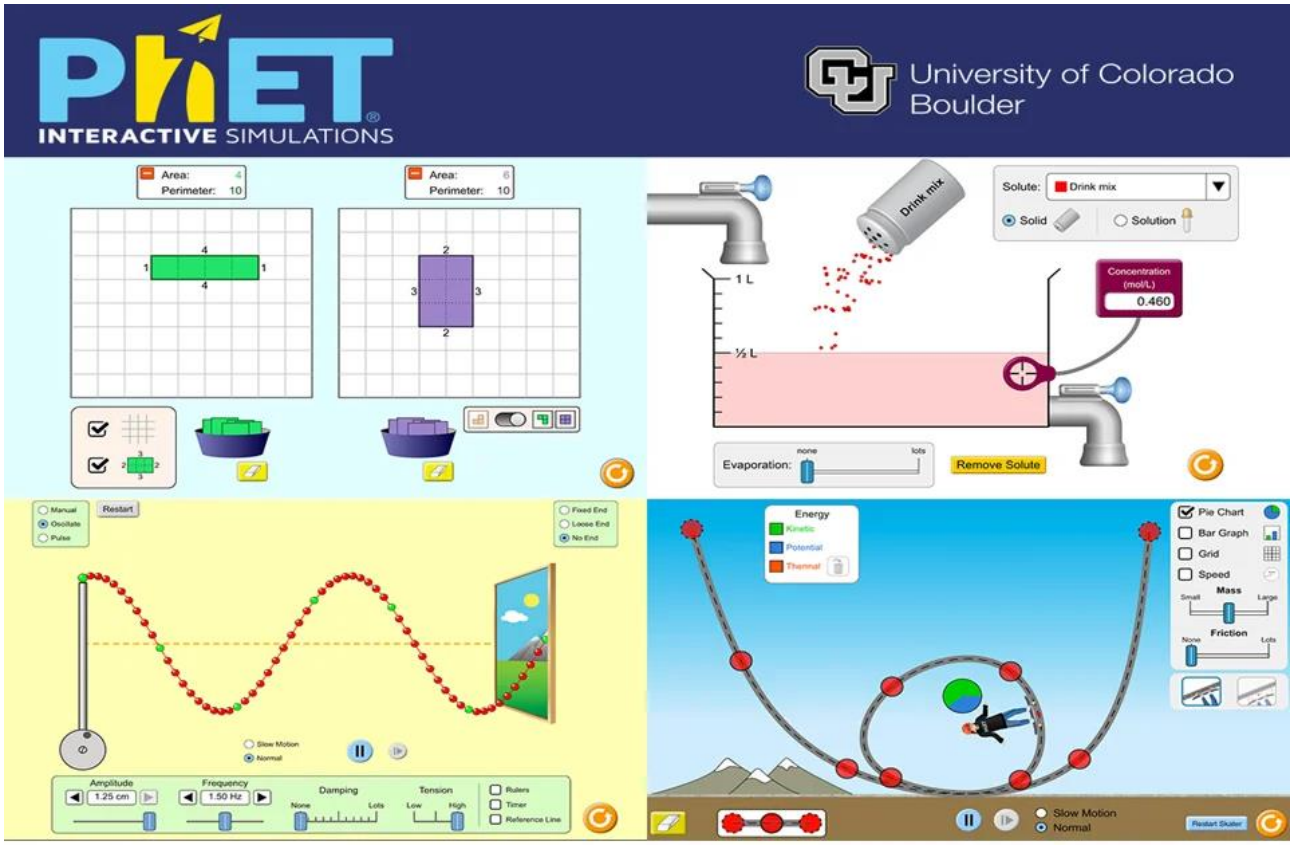

Рисунок 1.5 – Приклад симуляції у PhET Interactive Simulations

Algodoo - це інтерактивне програмне середовище для моделювання фізичних процесів у 2D-просторі, орієнтоване на навчання, творчість і

експерименти. Програма дозволяє створювати, редагувати й запускати фізичні сцени, де всі об’єкти поводяться згідно з законами класичної механіки. Вона є чудовим прикладом прикладного програмного забезпечення, де фізика руху є не другорядною, а центральною функцією, що повністю визначає суть продукту.

Algodoo базується на реалістичній двовимірній фізичній моделі: усі об’єкти сцени можуть мати масу, швидкість, прискорення, тертя, пружність, кутову швидкість, момент інерції, щільність тощо. Обчислення виконується в реальному часі, з урахуванням:

- гравітації — задається глобально;
- зіткнень — з повною підтримкою багатокутників, кіл, складних форм;
- сил — можна застосовувати як вручну, так і автоматизовано;
- з’єднань — підтримуються шестерні, поршні, пружини, мотори, шарніри тощо.

Гравці або учні можуть створювати як прості об’єкти (наприклад, м’яч, блок або похилу площину), так і складні механізми (ліфти, автомобілі, пушки, реверсивні передачі, годинникові механізми).

Особливістю Algodoo є візуальне моделювання. Кожен елемент можна змінювати без написання коду — всі параметри доступні через інтерфейс:

- масу, колір, текстуру, матеріал, коефіцієнт пружності, тертя, рух, тощо.
- Об’єкти можна:
- рухати вручну мишею;
- пускати в дію (катити, кидати, штовхати);
- прив’язувати до інших тіл (наприклад, створити підвісний міст із пружин);
- створювати датчики і тригери (які реагують на дотик, швидкість, кут).

Це дозволяє створювати повноцінні експерименти або демонстрації.

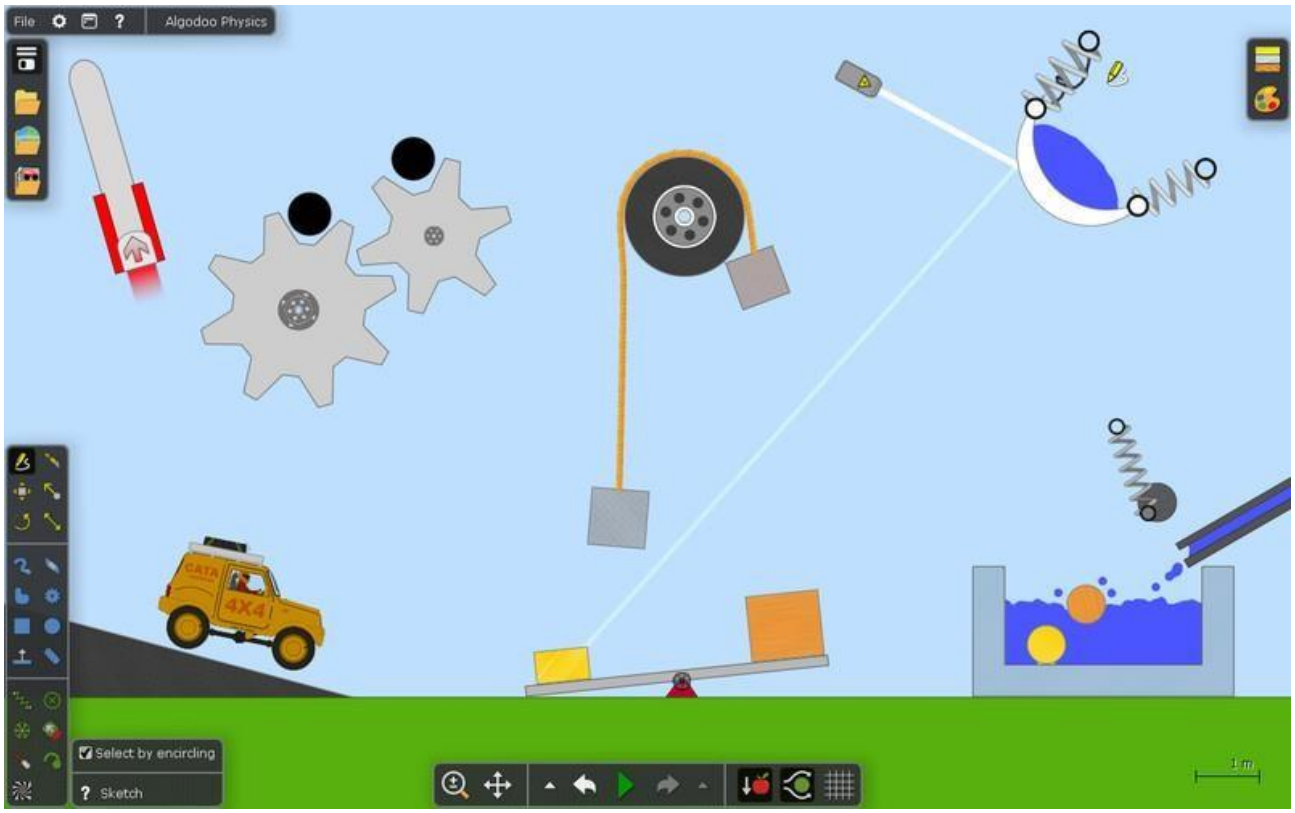

Рисунок 1.6 – Приклад симуляції у Algodoo

Scratch + Physics extensions - можливість для школярів створювати власні симуляції з простими фізичними ефектами.

Хоча базова версія Scratch не включає реалістичну фізику, можливість підключення розширень (extensions), зокрема Physics Engine (Box2D-based), дає змогу реалізувати прості фізичні моделі у 2D-просторі. Це відкриває доступ до створення симуляцій із підтримкою гравітації, маси, тертя, зіткнень, пружності тощо навіть для користувачів без досвіду програмування.

Після підключення фізичного розширення , у користувача з’являються нові блоки, які дозволяють:

- призначити об’єкту фізичне тіло (static, dynamic, sensor);
- змінювати масу, гравітацію, тертя, bounciness;
- застосовувати силу чи імпульс до об’єкта;
- виявляти зіткнення між фізичними тілами;

- використовувати з’єднання (joints) для створення підвісних конструкцій.

Користувачі можуть створювати сцени, в яких об’єкти:

- падають під впливом гравітації;
- зіскакують із платформ із урахуванням пружності;
- ковзають або сповільнюються залежно від тертя;
- штовхаються або обертаються внаслідок прикладеної сили;
- реагують на зіткнення зі стінами, підлогою, іншими тілами.

Також можна будувати просту симуляцію: маятники, кулі, механізми, транспорт.

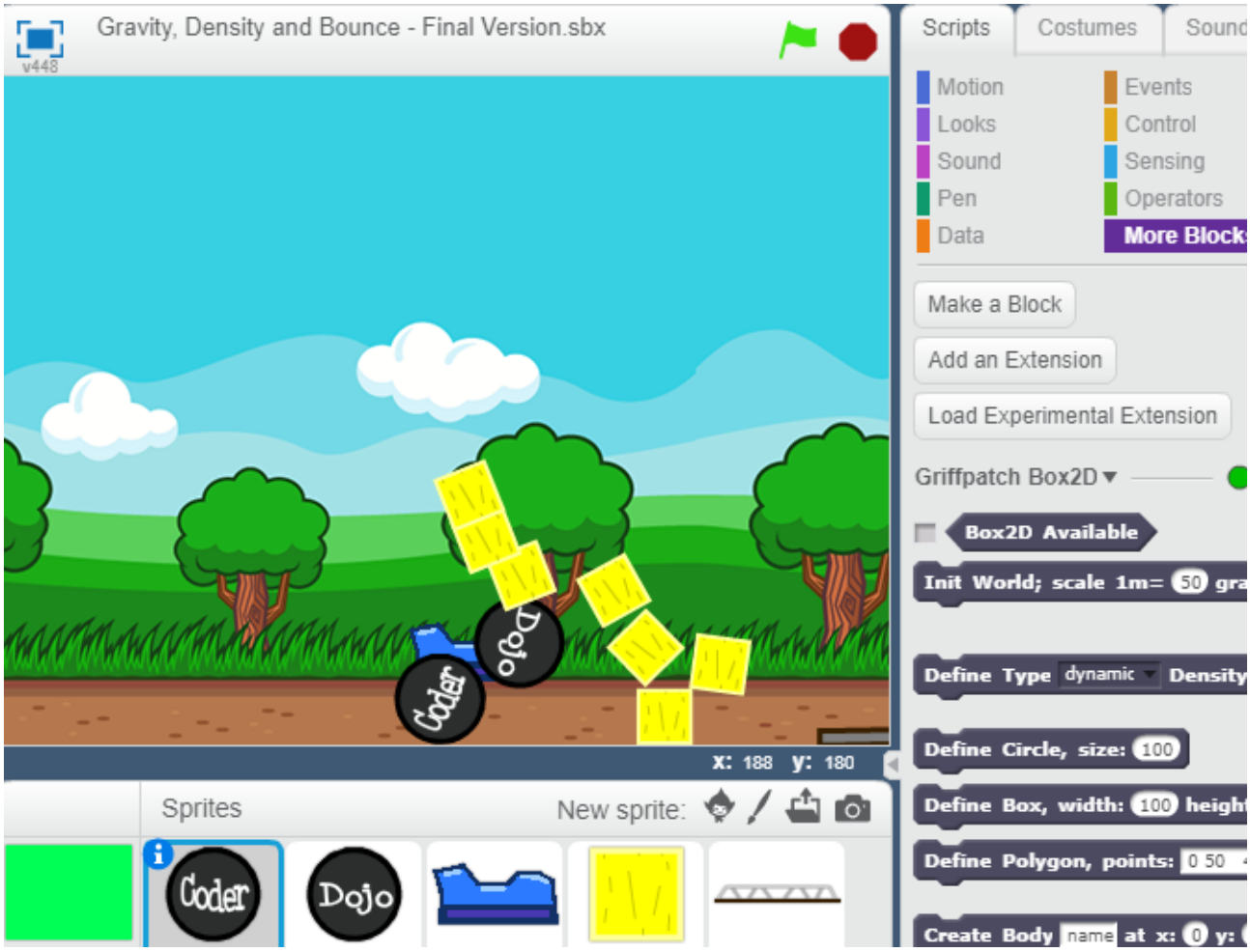

Рисунок 1.7 – Приклад симуляції у Scratch

Такі системи сприяють розвитку інженерного мислення, умінню працювати з моделями та розумінню причинно-наслідкових зв’язків.

1.4 Висновок до розділу 1

У першому розділі було проведено ретельне дослідження предметної сфери, яка стосується моделювання фізики руху в двовимірному просторі, що дозволило обґрунтувати актуальність та важливість розробки відповідного програмного забезпечення.

Як показує практика фізичні симуляції є потужним інструментом, який широко застосовується у різноманітних сферах: від ігрової індустрії до наукових досліджень та освіти. Завдяки фізичному моделюванню можна створювати інтуїтивно зрозумілі, реалістичні системи, що демонструють поведінку тіл згідно з законами класичної механіки. Особливо важливою є можливість відтворення таких явищ, як гравітація, сила тертя, зіткнення, інерція, відскок і траєкторія руху.

У підрозділі, присвяченому основам фізики у 2D-просторі, було розглянуто ключові поняття, які складають фундамент будь-якої симуляції: положення, швидкість, прискорення, маса, сила. Були розглянуті базові рівняння руху, типи сил, що найчастіше використовуються в моделюванні, а також принципи виявлення та обробки зіткнень між об'єктами. Окрему увагу було приділено числовим методам інтегрування, які дозволяють обчислювати положення об'єктів у дискретні моменти часу.

Розділ також висвітлив широке використання фізичних симуляцій на практиці: зокрема, в комп'ютерних іграх, де фізика дозволяє значно підвищити рівень реалізму та залучення користувача, а також у наукових та освітніх програмних продуктах, що надають можливість вивчати складні фізичні процеси без необхідності проведення дорогих або небезпечних експериментів.

Проведений аналіз дає змогу сформувати чітке розуміння задач, які має вирішувати система, що розробляється, а також визначити підходи, що будуть застосовані при її реалізації.

2 ОПИС ТЕХНІЧНОЇ РЕАЛІЗАЦІЇ ТА РОЗРОБКА АЛГОРИТМУ

2.1 Обгрунтування вибору інструментів технічної реалізації

Для втілення програмного забезпечення, яке імітує фізику руху в 2D просторі, було обрано актуальне середовище розробки — Unity. Основною підставою для такого рішення є те, що Unity пропонує зручні інструменти для створення двовимірних додатків з інтегрованим фізичним рушієм. Рушій базується на перевіреній часом технології Box2D, яка дає змогу автоматично обробляти сили, зіткнення, тертя, відскоки та інші фізичні взаємодії між об’єктами. Це значно полегшує процес розробки, адже більшість рутинних фізичних розрахунків виконується рушієм автоматично, а розробнику достатньо лише налаштувати параметри об'єктів і визначити сценарії їхньої поведінки [10].

Unity також має зручний візуальний редактор сцени, який дозволяє створювати, переміщати й змінювати об’єкти безпосередньо в середовищі розробки. Завдяки компонентній архітектурі легко комбінувати графічні, фізичні та логічні елементи: кожний об’єкт складається з набору компонентів, таких як Transform, SpriteRenderer, Rigidbody2D та Collider2D, які визначають зовнішній вигляд, розташування в просторі, фізичні властивості та здатність взаємодіяти з іншими тілами.

Для написання логіки в Unity використовується мова програмування C#, яка є сучасною, строго типізованою та гнучкою. Вона підтримує об'єктно-орієнтований підхід, делегати, події, систему класів і має високу сумісність із редактором Visual Studio. Це дає змогу ефективно реалізувати обробку фізичних подій, взаємодію користувача з об'єктами, а також програмне змінювання сили, швидкості, напрямку руху та інших параметрів у реальному часі [11].

Ще однією суттєвою перевагою Unity є її кросплатформеність. Проєкт, створений у середовищі Unity, за необхідності легко експортується під різні операційні системи та пристрої — Windows, Android, iOS, WebGL, Linux тощо.

Це відкриває широкі можливості для подальшого розвитку програмного продукту, його масштабування або розгортання у вигляді веб-застосунку чи мобільної програми.

Крім того, велика спільнота розробників Unity, розлога офіційна документація, доступні туторіали й приклади коду суттєво прискорюють процес навчання і розробки. Існує безліч готових рішень, плагінів і бібліотек, які можуть бути інтегровані в проєкт за потреби [13].

Всі ці фактори разом роблять Unity оптимальним середовищем для реалізації програмного забезпечення, що імітує фізику руху тіл у 2D-просторі. Вибір цієї платформи забезпечує гнучкість, продуктивність та надійність розробки, даючи можливість зосередитись безпосередньо на втіленні алгоритмів фізичних взаємодій.

2.2 Опис використаної технології

Розроблене програмне забезпечення призначене для моделювання фізичних явищ руху об'єктів у двовимірному просторі, ґрунтуючись на базових законах механіки. Основна ідея полягає у створенні простору, де розташовані фізичні тіла, які взаємодіють одне з одним під впливом сил, таких як тяжіння, тертя та пружність, з урахуванням їх маси та інших властивостей. Unity забезпечує можливість автоматично виконувати ці розрахунки завдяки вбудованому 2D-фізичному рушію.

Функціонування системи починається з формування основної сцени, до якої додаються статичні елементи оточення (наприклад, платформи, стіни, обмеження) та динамічні об'єкти, які будуть піддаватися впливу фізичних сил. Кожен динамічний об'єкт отримує компонент Rigidbody2D, що робить його чутливим до фізики, та відповідний Collider2D, що дозволяє йому взаємодіяти з іншими об'єктами при зіткненнях.

Наступний крок – налаштування параметрів фізичного середовища. Зокрема, через Project Settings → Physics 2D задається загальний вектор

гравітації, зазвичай спрямований вниз по осі Y. Окремі об'єкти можуть мати свої унікальні властивості: масу, лінійне та кутове тертя, коефіцієнт пружності (bounciness), обмеження на обертання або швидкість падіння.

Після ініціалізації відбувається основний фізичний цикл, що запускається на кожному кадрі за допомогою функції FixedUpdate() — це спеціальний метод Unity, що викликається з фіксованим часовим інтервалом, незалежно від частоти оновлення графіки, що гарантує стабільну роботу фізичних систем.

В рамках цього циклу кожне фізичне тіло оновлює свої координати відповідно до прикладених до нього сил. Наприклад, гравітація автоматично прискорює об'єкт вниз, а при контакті з поверхнею виникають сили реакції, тертя та відскоку. У випадку, якщо об’єкт отримує додаткову силу — наприклад, в результаті взаємодії з гравцем — її можна програмно застосувати через метод AddForce(), що дає змогу створювати ефекти штовхання, вибуху або керованого руху.

Зіткнення між тілами обробляються внутрішнім механізмом Unity. Компоненти Collider2D взаємодіють між собою автоматично, а розробник може відстежувати події зіткнення за допомогою методів OnCollisionEnter2D(), OnCollisionStay2D() та OnCollisionExit2D(), які дозволяють реалізувати додаткову логіку у відповідь на дотик — наприклад, зупинку, знищення об'єкта, зміну кольору або звуковий ефект.

У випадку, якщо об’єкти необхідно генерувати під час роботи програми (наприклад, при натисканні кнопки), використовуються функції Instantiate(), які динамічно додають нові елементи до сцени, одразу наділяючи їх фізичними властивостями.

Особливу роль відіграє система візуалізації. Незважаючи на те, що координати об’єкта обчислюються фізичним рушієм, переміщення та обертання візуального елементу (спрайта) автоматично синхронізуються через компонент Transform. Таким чином, користувач бачить плавний, реалістичний рух без потреби ручного оновлення координат.[9]

Також можливе використання візуальних індикаторів, які показують напрямок сили, траєкторію або швидкість об’єкта. Для цього застосовуються Gizmos, LineRenderer або UI-елементи.

Загальна послідовність роботи програми:

після запуску сцени відбувається ініціалізація фізичних параметрів об’єктів, далі в циклі FixedUpdate обчислюється вплив сил, обробляються зіткнення й оновлюються позиції, що завершується відображенням нового стану сцени. Всі взаємодії з користувачем — створення нових тіл, застосування сил або зміна параметрів — обробляються асинхронно у відповідь на події.

Таким чином, Unity дозволяє розробити ефективну, гнучку та масштабовану систему симуляції фізики у 2D-просторі з високою точністю та інтерактивністю.

2.3 Опис алгоритму у вигляді схеми

Для зручності розуміння процесу роботи симуляції фізики у 2D-просторі на базі Unity, нижче представлено спрощену блок-схему алгоритму, що відображає основні етапи роботи застосунку, а саме:

1. Ініціалізація сцени.
2. Призначення фізичних параметрів.
3. Основний цикл FixedUpdate.
4. Візуалізація сцени.
5. Обробка дій користувача.

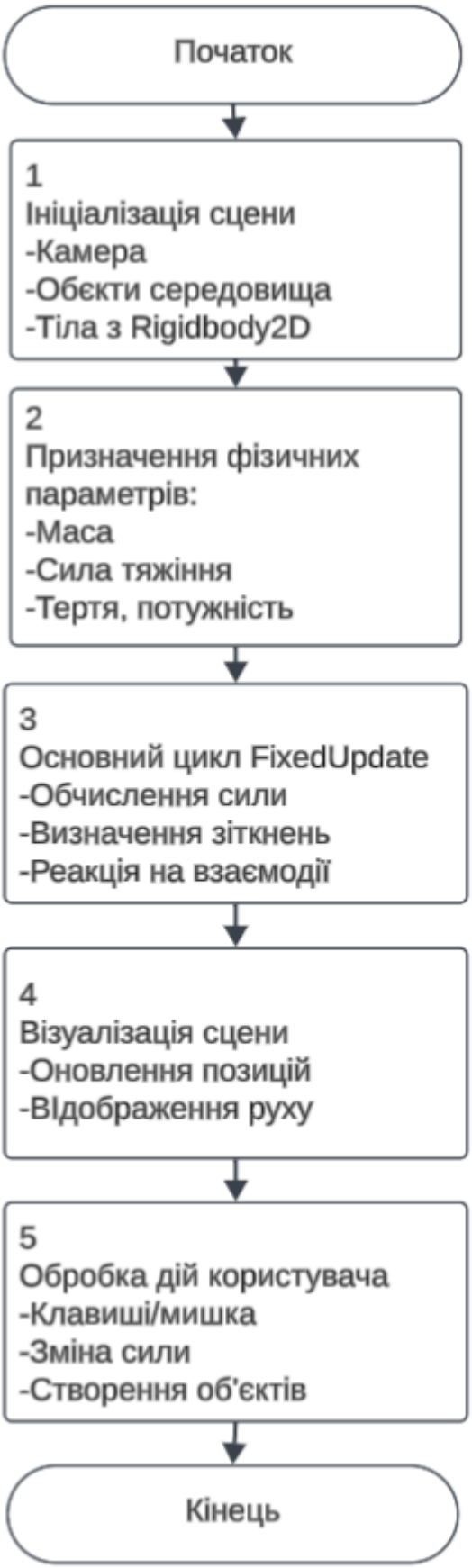

Рисунок 2.1 – Блок-схема логіки фізичної симуляції в Unity

Основні компоненти, використані в Unity:

- Rigidbody2D — головний компонент для фізики. Визначає масу, гравітацію, швидкість, прискорення.
- Collider2D (BoxCollider2D, CircleCollider2D) — дозволяє виявляти зіткнення між об’єктами.
- Physics2D — внутрішній рушій, який обчислює сили, зіткнення, тертя, відскоки тощо.
- FixedUpdate() — спеціальний метод Unity, який виконується з фіксованим інтервалом часу для стабільної фізики.
- AddForce(Vector2 force) — застосовує силу до тіла в певному напрямку.
- OnCollisionEnter2D() — метод для обробки подій зіткнення.

2.4 Висновок до розділу 2

У другому розділі було зосереджено увагу на технологічній базі для розробки програмного продукту, призначеного для імітації фізики руху об'єктів у двовимірному просторі. Було аргументовано вибір рушія Unity як ключового інструменту розробки, враховуючи його значний функціонал, дружній інтерфейс, інтегровану фізичну систему на основі Box2D, сумісність із мовою програмування C#, а також розширену екосистему, що забезпечує прискорене створення, тестування та масштабування програмних рішень.

Окреслено загальний підхід до втілення симуляції фізичних процесів у Unity. Надано текстовий опис функціонування основного фізичного алгоритму, який включає етапи: початкове налаштування сцени, встановлення фізичних параметрів, обробка фізичних взаємодій в циклі FixedUpdate та відображення результатів на екрані. Акцентовано увагу на можливості програмної взаємодії з об’єктами – застосування сили, додавання нових тіл, обробка зіткнень і створення візуальних ефектів.

Розглянута структура забезпечує гнучкість у реалізації основних законів механіки в двовимірному середовищі, моделюючи поведінку тіл з урахуванням гравітації, тертя, пружності, маси, а також дозволяє природну взаємодію об'єктів між собою та з користувачем. Unity автоматизує більшість низькорівневих обчислень, що дозволяє зосередитися на проєктуванні фізичних сцен і створенні потрібної поведінки.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ФІЗИКИ РУХУ ОБ'ЄКТІВ У 2D ПРОСТОРІ

3.1 Обґрунтування вибору мови програмування

Для розробки програмного забезпечення, що симулює рух об'єктів у двовимірному просторі, було обрано мову програмування C# у поєднанні з ігровим рушієм Unity. Це рішення базується на ретельному аналізі технічних специфікацій, функціональних можливостей та врахуванні переваг і недоліків C# та Unity у порівнянні з іншими мовами, зокрема, з Python.

При виборі враховувалися конкретні вимоги предметної області: необхідність реалізації точної фізичної симуляції руху, інтеграція з візуальним середовищем розробки, а також забезпечення взаємодії об'єктів у двовимірному просторі. Крім того, було важливо забезпечити швидке тестування, налагодження та масштабованість системи [10-12].

Переваги C# та Unity.

1. Глибока інтеграція фізичного рушія — Unity надає потужний та оптимізований 2D-фізичний двигун із підтримкою компонентів Rigidbody2D, Collider2D та сил, що значно полегшує розробку реалістичного руху об'єктів без потреби писати фізичні алгоритми з нуля.

2. Об'єктно-орієнтоване програмування (ООП) — C# підтримує чітку парадигму ООП, що дає змогу структурувати код за класами, створювати компоненти, що повторно використовуються, та підтримувати чистоту архітектури, що вкрай важливо при масштабуванні проекту.

3. Потужне середовище розробки — Unity пропонує інтегрований візуальний редактор, де можна в режимі реального часу змінювати параметри фізики, анімації та налагоджувати код, що значно прискорює цикл розробки.

4. Широка екосистема та підтримка — Unity має великий обсяг документації, готових бібліотек та прикладів, що полегшує швидкий пошук рішень для типових задач та інтеграцію додаткових функцій[10-12].

5. Кросплатформність — програми, створені в Unity, можна запускати на різних операційних системах та пристроях без серйозних змін у коді.

6. Продуктивність — C# компілюється у байт-код, який виконується швидко, а оптимізації Unity дають змогу ефективно працювати навіть зі складними фізичними симуляціями в реальному часі.

Недоліки C# та Unity.

1. Складність на початковому етапі — для новачків робота з Unity та C# може здатися складною через велику кількість функціональності, потребу розуміти компоненти рушія та принципи ООП.

2. Великі розміри проекту та ресурсоємність — Unity має великий розмір інсталяційних файлів та створюваних проектів, що може бути зайвим для надзвичайно простих 2D-ігор або симуляцій.

3. Залежність від рушія — розробка тісно пов’язана з платформою Unity, що ускладнює перенесення на інші технології[10-12].

Переваги Python.

1. Простота та читабельність коду — Python має зрозумілий та лаконічний синтаксис, що значно спрощує розробку та підтримку коду, особливо на ранніх етапах проекту.

2. Швидке прототипування — завдяки простоті синтаксису та великій кількості готових бібліотек (наприклад, Pygame для 2D графіки), розробка базових функцій відбувається дуже швидко.

3. Розвинені бібліотеки для наукових розрахунків — такі пакети як NumPy, SciPy, Matplotlib та інші надають потужні інструменти для математичного моделювання, що є корисним для створення фізичних моделей.

4. Велика спільнота та ресурси — наявність великої кількості навчальних матеріалів та підтримки дає змогу легко знаходити рішення для різних задач.

Недоліки Python у контексті розробки 2D фізики.

1. Відсутність вбудованого фізичного рушія — на відміну від Unity, Python не має готових компонентів для обробки фізики та зіткнень в іграх, що потребує розробки цих механізмів з нуля або використання сторонніх бібліотек, які не завжди є достатньо оптимізованими.

2. Низька продуктивність — будучи інтерпретованою мовою, Python значно повільніший за компільовані мови, що критично при обчисленні фізики в реальному часі, особливо для інтерактивних додатків.

3. Обмежені можливості для інтерактивної 2D графіки — бібліотеки на кшталт Pygame мають менший функціонал та гнучкість, ніж Unity, та потребують більше зусиль для налаштування візуальної частини та фізики.

4. Відсутність потужного візуального редактора — немає можливості швидко налаштовувати сцени, фізичні властивості об’єктів та взаємодії через зручний графічний інтерфейс[10-12].

Вибір C# та Unity для реалізації фізики руху об’єктів у 2D просторі базується на їх потужному інструментарії, оптимізаціях та інтеграції фізичного рушія, що забезпечує швидке, надійне та гнучке створення програмного забезпечення. Python, незважаючи на свої переваги у простоті та швидкості прототипування, має суттєві обмеження у продуктивності та відсутність спеціалізованих інструментів для фізичної симуляції в ігровому контексті. Отже, для поставленої задачі C# у поєднанні з Unity є оптимальним вибором.

3.2 Програмна реалізація програмного забезпечення

Програмна реалізація модуля управління пересуванням героя була здійснена за допомогою мови C# в середовищі Unity, що забезпечує зручну інтеграцію фізичного моделювання та обробки команд від користувача. Логіка управління рухом персонажа організована в одному класі Hero, який успадковується від MonoBehaviour — стандартного базового класу для компонентів в Unity. Це дозволяє безпосередньо взаємодіяти з фізичним тілом об'єкта (Rigidbody2D) та його візуальним представленням (SpriteRenderer).

Клас Hero містить базові налаштування швидкості руху та сили стрибка, які доступні для зміни в редакторі Unity завдяки атрибуту [SerializeField]:

speed — швидкість горизонтального пересування героя;

jumpForce — сила, з якою герой стрибає.

Для управління стрибками застосовується логічна змінна canJump, що вказує, чи дозволено герою виконати стрибок в конкретний момент часу (наприклад, коли герой торкається поверхні).

Основні складові класу:

Rigidbody2D rb — посилання на фізичне тіло героя, що відповідає за рухи та взаємодію з фізичним середовищем;

SpriteRenderer sprite — компонент для відображення спрайту героя, використовується для візуального розвороту персонажа при зміні напрямку руху.

Метод Awake() відповідає за ініціалізацію посилань на компоненти Rigidbody2D та SpriteRenderer до початку роботи гри.

Метод Update() виконується щокадру і відповідає за обробку команд від користувача. У ньому викликається метод Run() для пересування героя, а також перевіряється можливість стрибка при натисканні клавіш Space або стрілки вгору, якщо герой наразі може стрибати.

Метод Run() обробляє горизонтальний рух героя відповідно до натиснутих клавіш:

Клавіша A та стрілка вліво змінюють напрямок руху вліво;

Клавіша D та стрілка вправо — рух вправо.

Напрямок руху задає горизонтальну швидкість Rigidbody2D, а також керує розворотом спрайту героя, щоб він дивився в бік пересування.

Метод Jump() застосовує вертикальну силу до Rigidbody2D, реалізуючи стрибок героя, і скидає прапорець canJump, щоб запобігти багатократним стрибкам у повітрі.

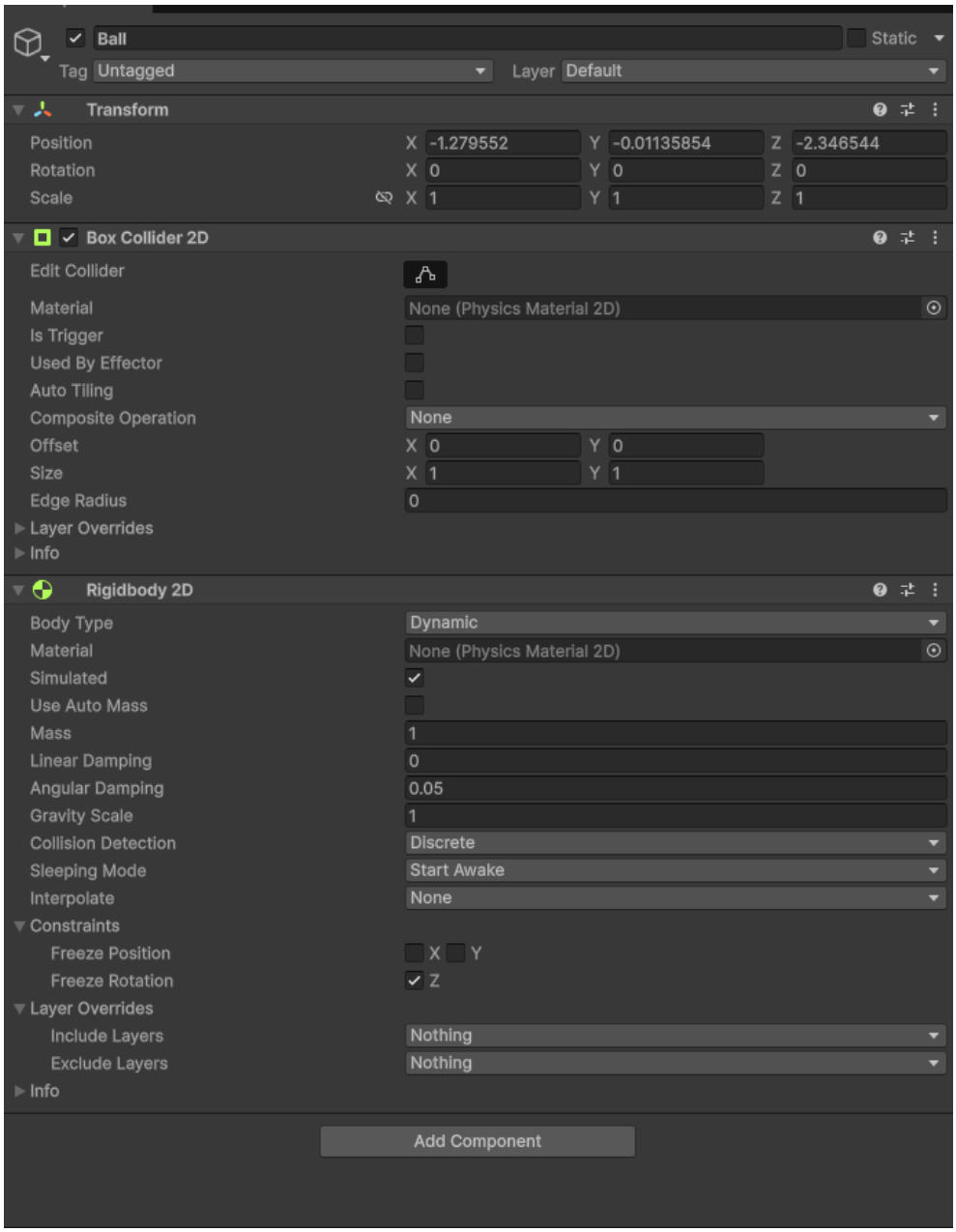

Рисунок 3.1 – Вигляд інспектора у якому прописані налаштування за допомогою методу Rigidbody 2D.

Колізії з іншими об’єктами відслідковуються через методи OnCollisionEnter2D та OnCollisionExit2D, які визначають, коли герой торкається платформи (або землі) — тоді змінна canJump встановлюється в true, дозволяючи стрибок, або false, коли герой відривається від поверхні.

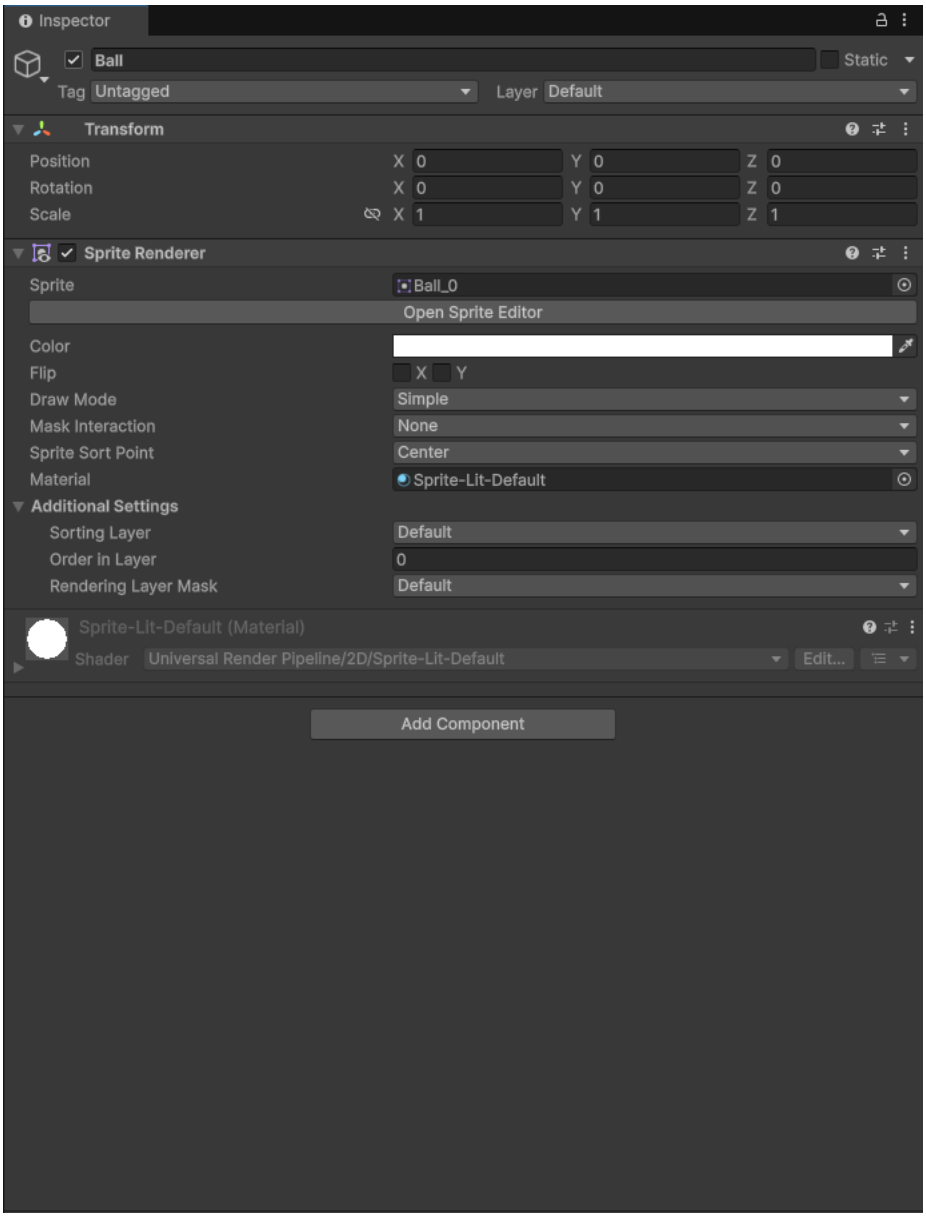

Рисунок 3.2 – Вигляд інспектора у якому вказані спрайти з текстурою об’єкта.

Даний модуль забезпечує базовий, але ефективний механізм управління рухом персонажа у 2D просторі, з урахуванням як горизонтального руху, так і можливості стрибка, а також управління напрямком обличчя героя за допомогою візуального перевороту спрайту. Архітектура коду відповідає принципам інкапсуляції та модульності, що полегшує подальше розширення функціональності (наприклад, додавання анімацій, різних типів пересування, складніших фізичних взаємодій).

3.3 Тестування роботи програмного забезпечення для реалізації фізики руху об'єктів у 2D просторі

Метою тестування роботи програмного забезпечення для реалізації фізики руху об’єктів у 2D просторі було перевірити коректність роботи налаштувань фізичних взаємодій між об’єктами. Тестування проводилось з перевіркою усіх функціональних можливостей.

Для запуску програмного забезпечення потрібно встановити та запустити Assets програмного забезпечення та рушій Unity. Запуск програми здійснюється безпосередньо за допомогою рушію Unity. На рисунку 3.1 показано головне вікно Unity яке з’являється після запуску рушія

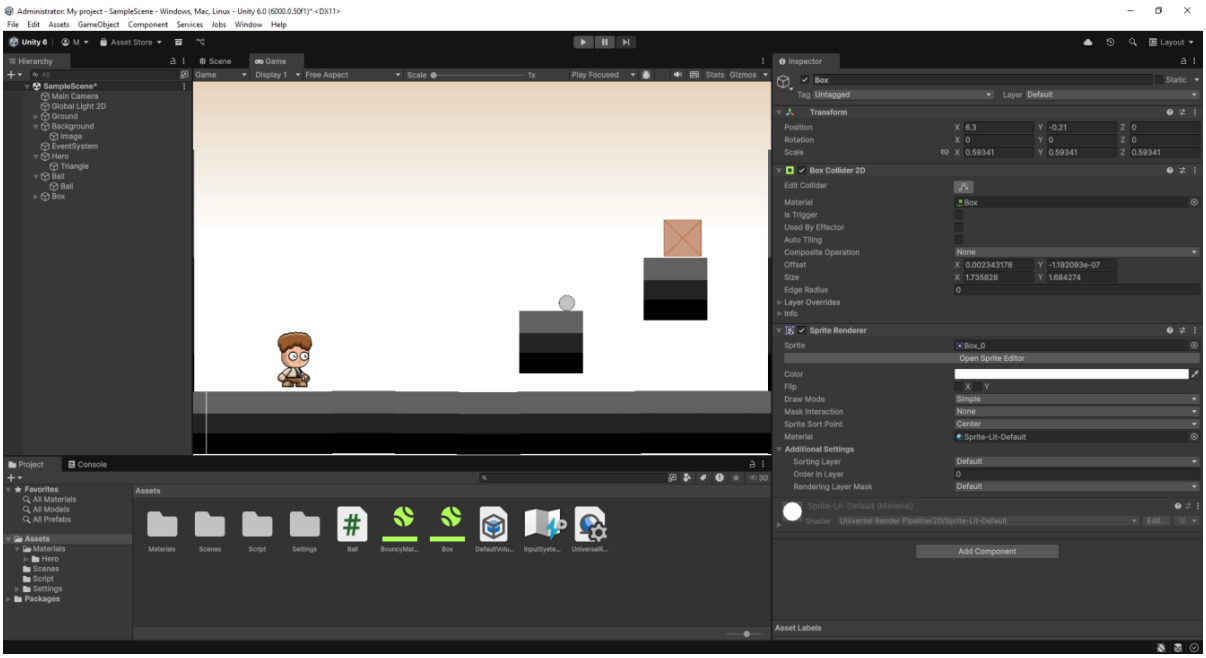

Рисунок 3.3 – Загальний вигляд рушія Unity.

Для початку тестування нам потрібно натиснути на кнопку “Play” у верхній центральній частині екрана. Запуск програмного забезпечення надає змогу рухатись “Герою” щоб можна було взаємодіяти з різними об’єктами розташованими на екрані. Для більш реалістичної взаємодії герой отримав обмеження у кількості стрибків. Також різноманітність об’єктів на екрані носить не лише естетичний характер у кожного об’єкта налаштовані свої індивідуальні параметри.

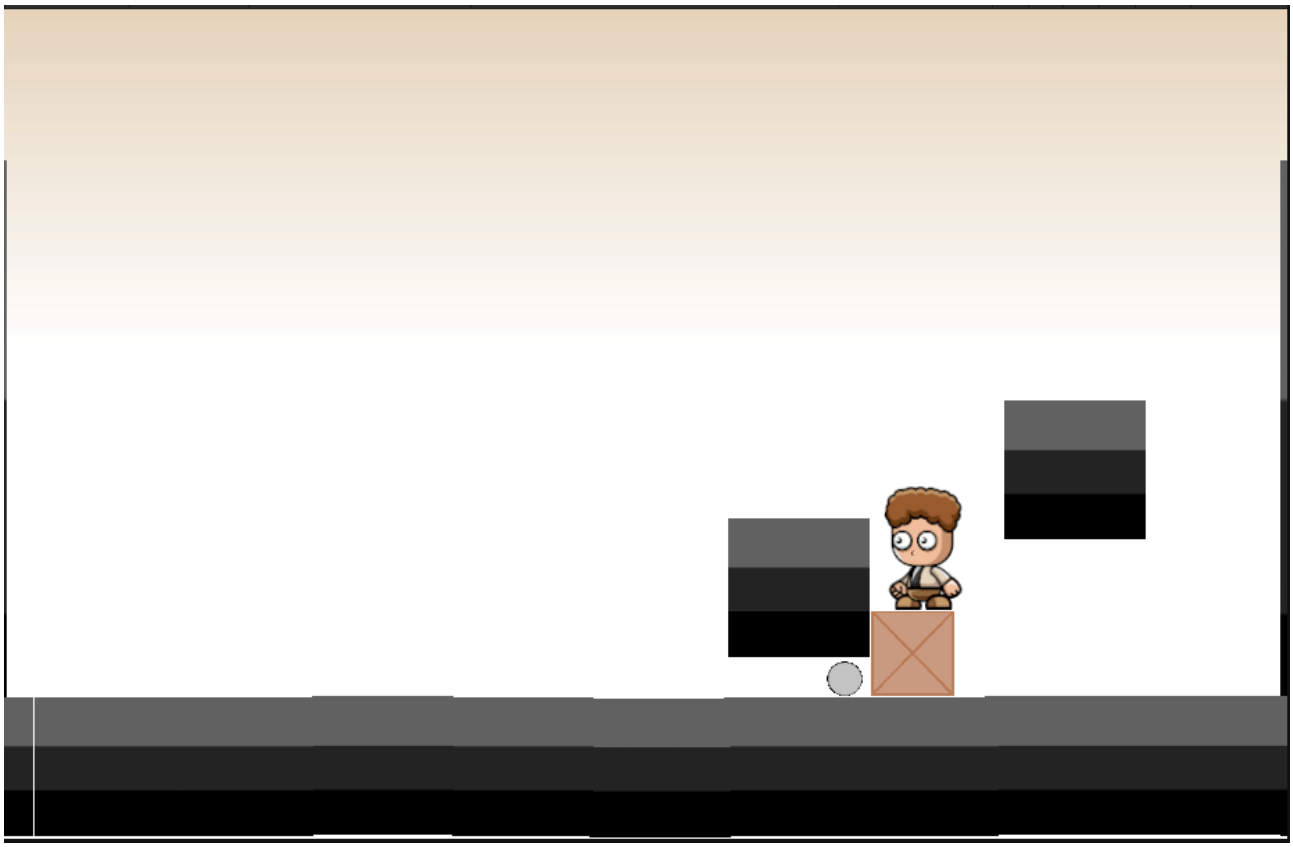

Рисунок 3.4 – Взаємодія між об’єктами

Тестування програмного забезпечення підтвердило коректність налаштувань усіх частин програмного забезпечення. Система демонструє ситуації приближені до реалістичних взаємодії з об’єктами. Тестування підтвердило готовність програмного модуля та його можливості демонстрування фізики руху у 2D середовищі.

3.4 Висновок до розділу 3

У третьому розділі було розроблено та програмно втілено модуль управління переміщенням персонажа у 2D-просторі, після чого було проведено всебічне тестування його основних робочих складових.

На стадії визначення технологій було обрано платформу Unity та мову програмування C#. Це забезпечило легку інтеграцію фізичного моделювання, обробки введення даних від користувача, а також можливість оперативного створення прототипів ігрової логіки. Вибір був обумовлений широким застосуванням Unity у сфері розробки 2D ігор, значним співтовариством розробників та розгалуженою документацією.

Програмна реалізація модуля була виконана відповідно до заздалегідь спроектованої архітектури, де логіка руху персонажа зосереджена у класі Hero. Цей клас відповідає за горизонтальний рух, стрибки, а також управління напрямком обличчя персонажа через візуальне обертання спрайту. Застосування компонентів Rigidbody2D і SpriteRenderer дозволило ефективно інтегрувати фізику руху та візуалізацію. Розподіл функціональності на окремі методи Run(), Jump(), а також обробка колізій сприяли зрозумілості та модульності коду.

Комплексне тестування модуля включало перевірку коректності руху персонажа за різних комбінацій натискання клавіш, валідацію механізму стрибка з урахуванням фізичних колізій із платформами та тестування коректності відображення напрямку персонажа. Результати тестування підтвердили стабільну роботу системи, адекватну реакцію на дії користувача та відсутність помилок у логіці пересування.

Отже, розроблений програмний модуль успішно виконує завдання з управління переміщенням персонажа в 2D-оточенні, забезпечуючи надійну основу для подальшого розширення функціональності та покращення ігрової механіки.

ВИСНОВКИ

В межах представленої роботи було розроблено програмний комплекс, призначений для моделювання фізичних процесів руху об'єктів у двовимірному просторі, використовуючи рушій Unity. Робота охоплює повний спектр дослідження, починаючи з аналізу предметної галузі та формулювання наукової задачі, і завершуючи практичною реалізацією функціонального продукту [21].

В першому розділі виконано детальний аналіз фізичних законів, які визначають рух тіл у 2D-просторі. Розглянуто кінематичні та динамічні властивості, типи сил, таких як гравітація, тертя, пружність, а також методи чисельного інтегрування руху та принципи виявлення зіткнень. Окрім того, проаналізовано ключові сфери застосування фізичних симуляцій – в індустрії розваг, науці, інженерії та освітньому процесі.

У другому розділі обґрунтовано вибір технологічної платформи. Unity було обрано як основу для розробки, враховуючи інтегрований фізичний рушій Box2D, компонентну архітектуру та широкі можливості для графіки та взаємодії з користувачем. Описано логіку функціонування системи, зокрема, етапи ініціалізації сцени, обробки фізичних взаємодій в циклі FixedUpdate, застосування сил та візуалізацію руху об'єктів.

У третьому розділі сформульовано науково-практичну мету дослідження, визначено об'єкт та предмет роботи, а також окреслено конкретні завдання, які успішно реалізовано в процесі розробки.

У результаті проведеної роботи створено інтерактивну систему, яка демонструє рух фізичних тіл під впливом зовнішніх та внутрішніх сил. Об'єкти на сцені реалізують поведінку згідно з законами механіки: падають під дією гравітації, зупиняються через тертя, відскакують під час зіткнень, реагують на вплив користувача. Користувачу надається можливість динамічно змінювати параметри симуляції та додавати нові об'єкти, що робить програму придатною як для навчальних цілей, так і для демонстраційних показів.

Отримані результати підтверджують коректність вибору технічних рішень, ефективність запропонованих алгоритмів та доцільність практичного впровадження програмного продукту. В майбутньому система може бути розширена шляхом додавання додаткових функцій: реалізацією складних тіл, інтеграцією з 3D-середовищем, розробкою систем частинок, або використанням для створення повноцінного фізичного симулятора або навчальної гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity Manual: Physics 2D — https://docs.unity3d.com/Manual/Physics2D.html
2. Венгер В. І. Основи комп’ютерного моделювання фізичних процесів. – Київ: Либідь, 2017. – 232 с.
3. Кириченко О.О. Теоретична механіка. – Харків: ХНУРЕ, 2019. – 348 с.
4. Чернов Ю.І. Основи прикладної механіки. – Вінниця: ВНТУ, 2021. – 215 с.
5. Колесник О.С., Бойчук І.Є. Основи класичної механіки. – Львів: ЛНУ, 2020. – 282 с.
6. Коваленко О. О. Моделювання руху тіл у двовимірному просторі: методи та алгоритми. Вісник Київського національного університету імені Т. Шевченка. Серія «Прикладна математика і механіка», 2020, Вип. 45, с. 56-67.
7. Петренко С. Моделювання руху об’єктів у 2D-просторі з урахуванням силових взаємодій. Науковий вісник Черкаського університету. Серія: Технічні науки, 2019, Вип. 72, с. 89-96.
8. Millington I. Physics for Game Developers. – Morgan Kaufmann, 2020. – 350 p.
9. Бондаренко І. М. Моделювання динаміки твердих тіл у 2D-системах. Науковий вісник Національного технічного університету України "Київський політехнічний інститут", 2018, Вип. 85, с. 102-110.
10. Журавель О. С. Методи чисельного інтегрування в задачах моделювання руху об'єктів у двовимірному просторі. Вісник Харківського національного університету, Серія: Прикладна математика, 2019, Вип. 35, с. 45-53.
11. Eberly D. H. 2D Physics for Game Programmers. – Morgan Kaufmann, 2014. – 290 p.
12. Thomas R., Millington I. Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for your Game. – CRC Press, 2018. – 400 p.

13. Smith R., Jones A. Real-Time 2D Physics Simulation for Interactive Applications. IEEE Transactions on Visualization and Computer Graphics, 2019, Vol. 25(6), pp. 2104-2115.
14. Гамільтон Дж. Unity у дії. Розробка ігор на C#. – Харків: Видавництво Фабула, 2021. – 416 с.
15. Могильний С. А., Слободянюк О. О. Методи чисельного моделювання фізичних процесів. – Київ: КНЕУ, 2018. – 240 с.
16. Савченко І. В., Костенко А. П. Основи об'єктно-орієнтованого програмування мовою C#. – Вінниця: ВНТУ, 2019. – 176 с.
17. Box2D Manual. — https://box2d.org/documentation/
18. Netguru. "Python vs C#: What's the difference?" *Netguru*, — https://www.netguru.com/blog/python-vs-c-sharp
19. GeeksforGeeks. "Difference between Python and C#" *GeeksforGeeks*, — https://www.geeksforgeeks.org/difference-between-python-and-c-sharp/
20. Indeed Career Guide. "C# vs Python: Which Language Is Better?" *Indeed*, — https://www.indeed.com/career-advice/finding-a-job/c-sharp-vs-python
21. А. А. Яровий, О. В. Сілагін, І. В. Богач. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» Вінниця : ВНТУ, 2023. 54 с.

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення для реалізації фізики руху об'єктів у 2D просторі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,14 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Гармаш В.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Тарасовський М.А.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (ОБОВ'ЯЗКОВИЙ)

Фрагмент лістингу програмного коду

Вміст файлу Hero.cs

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

public class Hero : MonoBehaviour
{
   [SerializeField] private float speed = 3f;
   [SerializeField] private float jumpForce = 1f;
   private bool canJump = false;

   private Rigidbody2D rb;
   private SpriteRenderer sprite;

   private void Awake()
   {
      rb = GetComponent<Rigidbody2D>();
      sprite = GetComponentInChildren<SpriteRenderer>();
   }

   private void Update()
   {
      Run();

      if (canJump && (Input.GetKeyDown(KeyCode.Space) ||
Input.GetKeyDown(KeyCode.UpArrow)))
         Jump();
   }

   private void Run()
   {
      float direction = 0f;

      if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
         direction = -1f;
      else if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
         direction = 1f;

      Vector3 dir = transform.right * direction;
      transform.position = Vector3.MoveTowards(transform.position,
transform.position + dir, speed * Time.deltaTime);
```

```
        if (direction != 0f)
            sprite.flipX = direction < 0.0f;
    }

    private void Jump()
    {
        rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Ground"))
        {
            canJump = true;
        }
    }

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

using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class BallController : MonoBehaviour
{
    private Rigidbody2D rb;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }
```

Вміст файлу Ball.cs

```
using UnityEngine;
[RequireComponent(typeof(Rigidbody2D))]
public class BallController : MonoBehaviour
{
    public float jumpForce = 5f;          // Сила підстрибування
    private Rigidbody2D rb;
    private bool isGrounded = false;      // Чи торкається м'яч землі

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            rb.velocity = new Vector2(rb.velocity.x, 0f); // Скидання вертикальної
швидкості
            rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
        }
    }

    // Перевірка зіткнення з "землею"
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Якщо м'яч торкнувся чогось із тегом "Ground" — вважаємо, що він на
землі
        if (collision.collider.CompareTag("Ground"))
        {
            isGrounded = true;
        }
    }

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

```
[RequireComponent(typeof(Rigidbody2D), typeof(Collider2D))]
public class BallBounciness : MonoBehaviour
{
   public float bounciness = 0.8f;
   public float friction = 0.3f;

   void Start()
   {
      // Створення нового PhysicsMaterial2D у пам’яті
      PhysicsMaterial2D mat = new PhysicsMaterial2D("AutoBouncy");
      mat.bounciness = bounciness;
      mat.friction = friction;
      mat.bounceCombine = PhysicsMaterialCombine.Multiply;

      // Призначення цього матеріалу колайдеру
      Collider2D col = GetComponent<Collider2D>();
      col.sharedMaterial = mat;
   }
}
void Update()
{
   if (Input.GetKeyDown(KeyCode.UpArrow))
   {
      Collider2D col = GetComponent<Collider2D>();
      col.sharedMaterial.bounciness += 0.1f;
   }
}
```

Вміст файлу Box.cs

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class BoxController : MonoBehaviour
{
   public float jumpForce = 0f;          // Сила підстрибування
   private Rigidbody2D rb;
   private bool isGrounded = false;      // Чи торкається коробка землі

   void Start()
   {
      rb = GetComponent<Rigidbody2D>();
   }

   void Update()
```

```
    {
        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            rb.velocity = new Vector2(rb.velocity.x, 0f); // Обнулення вертикальної
швидкості
            rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Якщо коробка торкнулась об’єкта з тегом Ground
        if (collision.collider.CompareTag("Ground"))
        {
            isGrounded = true;
        }
    }

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

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)

ІЛЮСТРАТИВНА ЧАСТИНА

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ФІЗИКИ РУХУ ОБ'ЄКТІВ У 2D ПРОСТОРІ»

Виконав: студент 4 курсу, групи 2КН-21б

Спеціальності 122 – Комп'ютерні науки

(шифр і назва напрямку підготовки, спеціальності)

Максим ТАРАСОВСЬКИЙ

(прізвище та ініціали)

Керівник: Д.т.н доц. каф. АІІТ

Володимир ГАРМАШ

(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Рисунок В.1 – Зображення гри Angry Birds

Рисунок В.2 – Зображення гри Terraria

Рисунок В.3 – Зображення гри Limbo

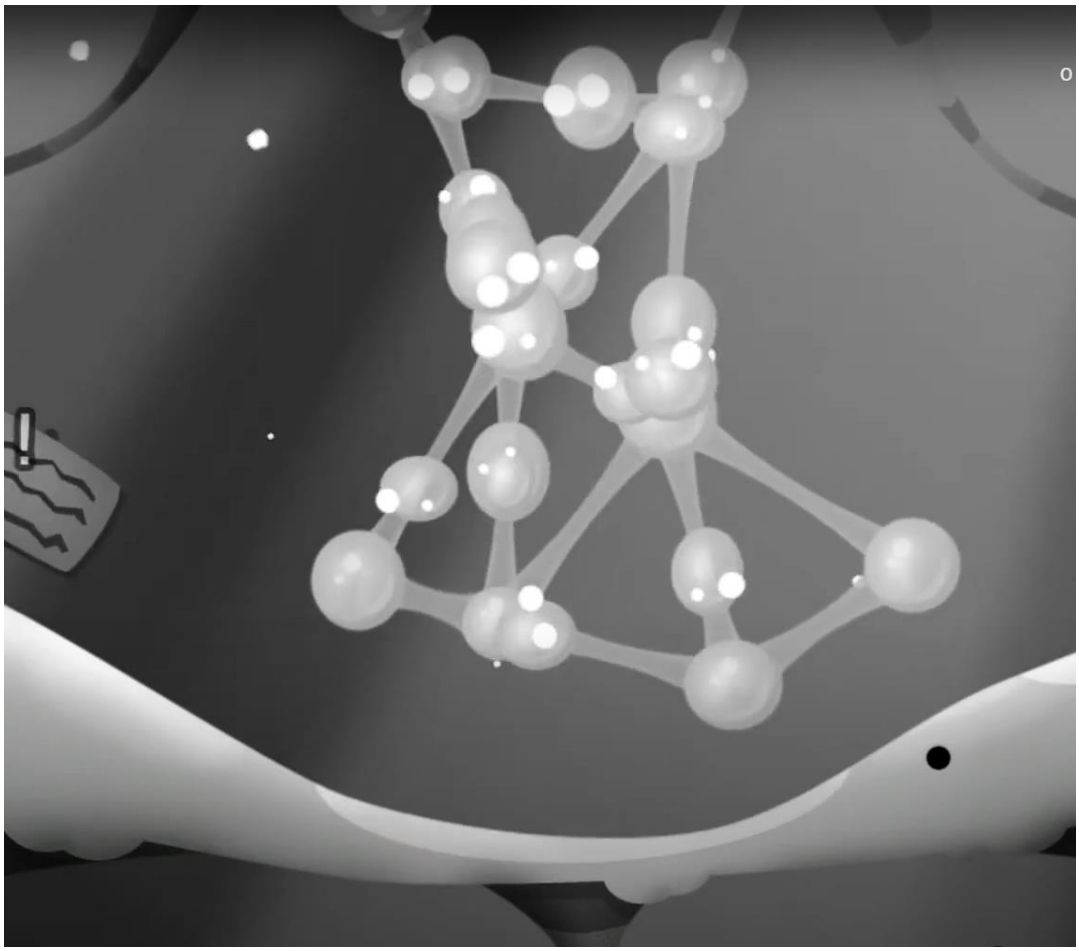

Рисунок В.4 – Зображення гри World of Goo

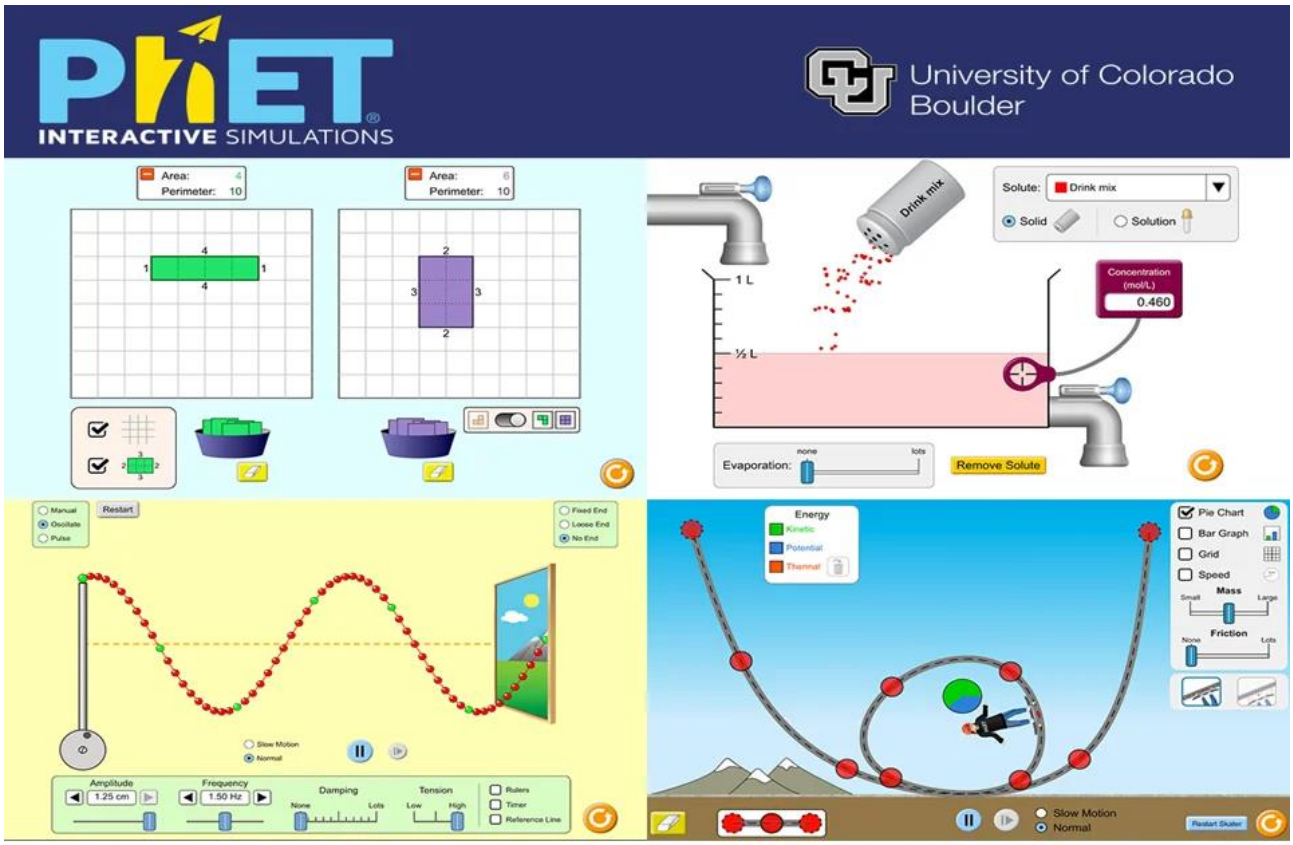

Рисунок В.5 – Приклад симуляції у PhET Interactive Simulations

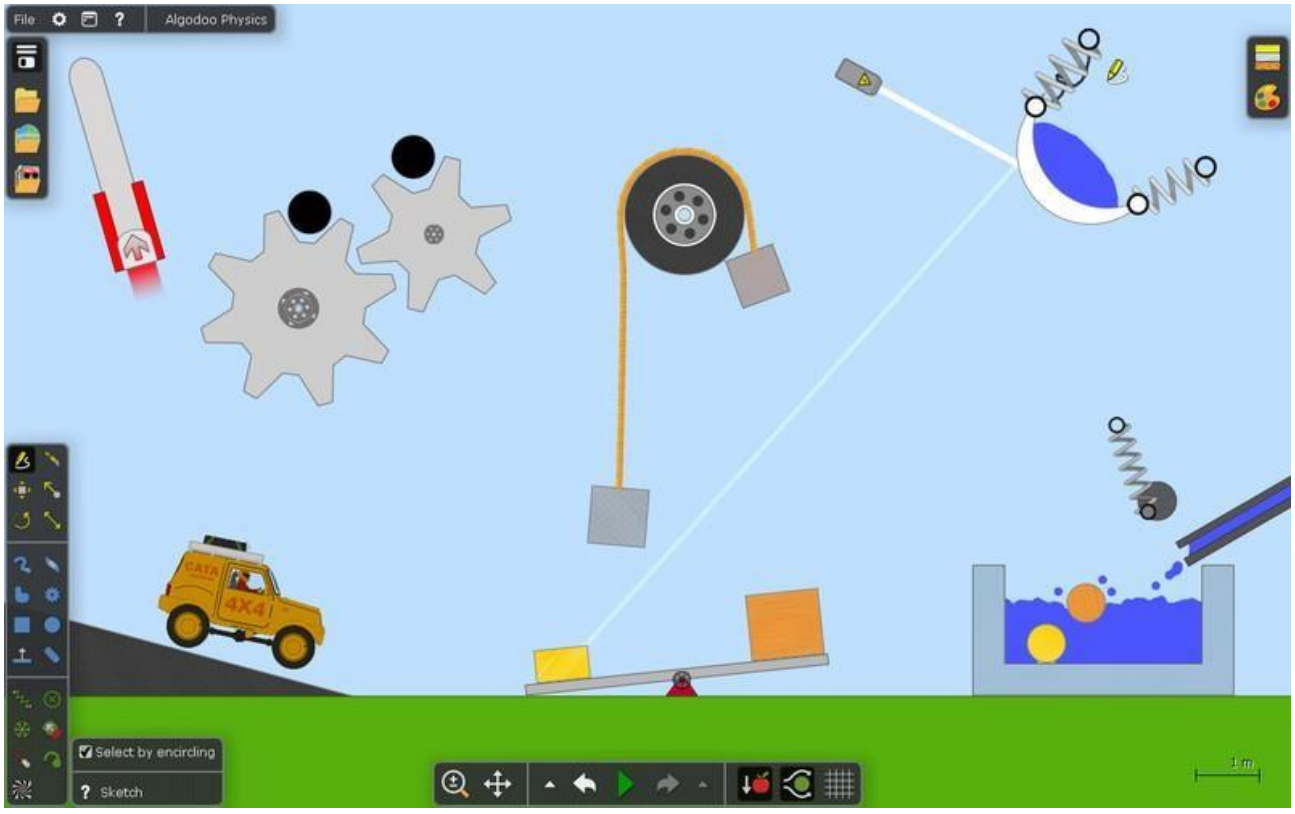

Рисунок В.6 – Приклад симуляції у Algodoo

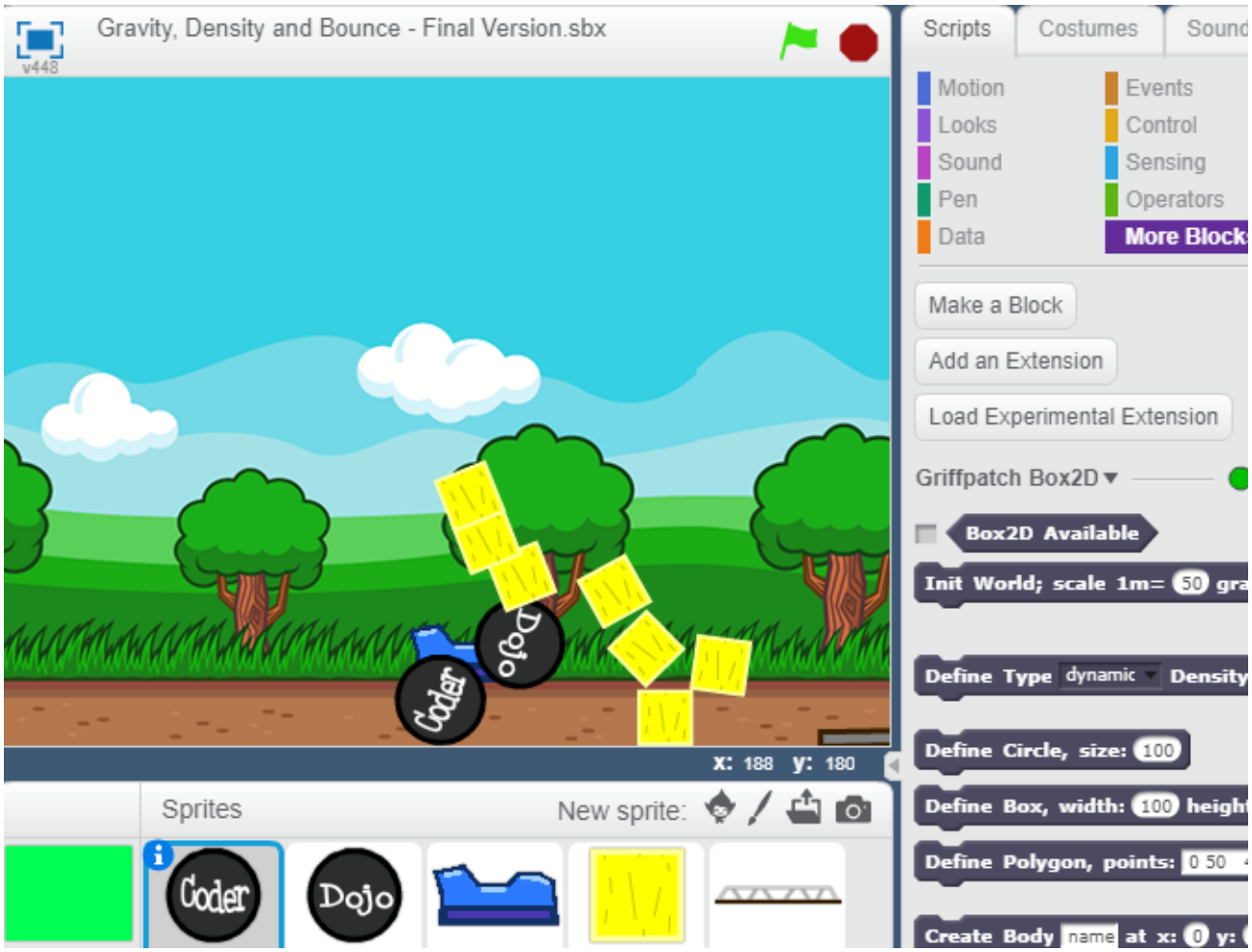

Рисунок В.7 – Приклад симуляції у Scratch

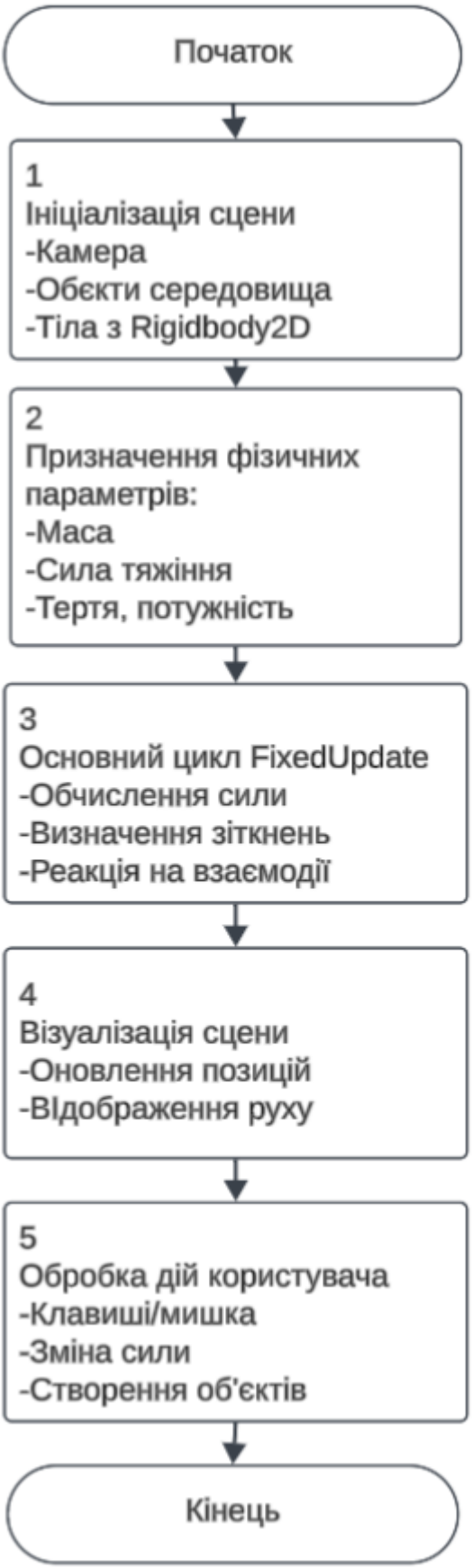

Рисунок В.8 – Блок-схема логіки фізичної симуляції в Unity

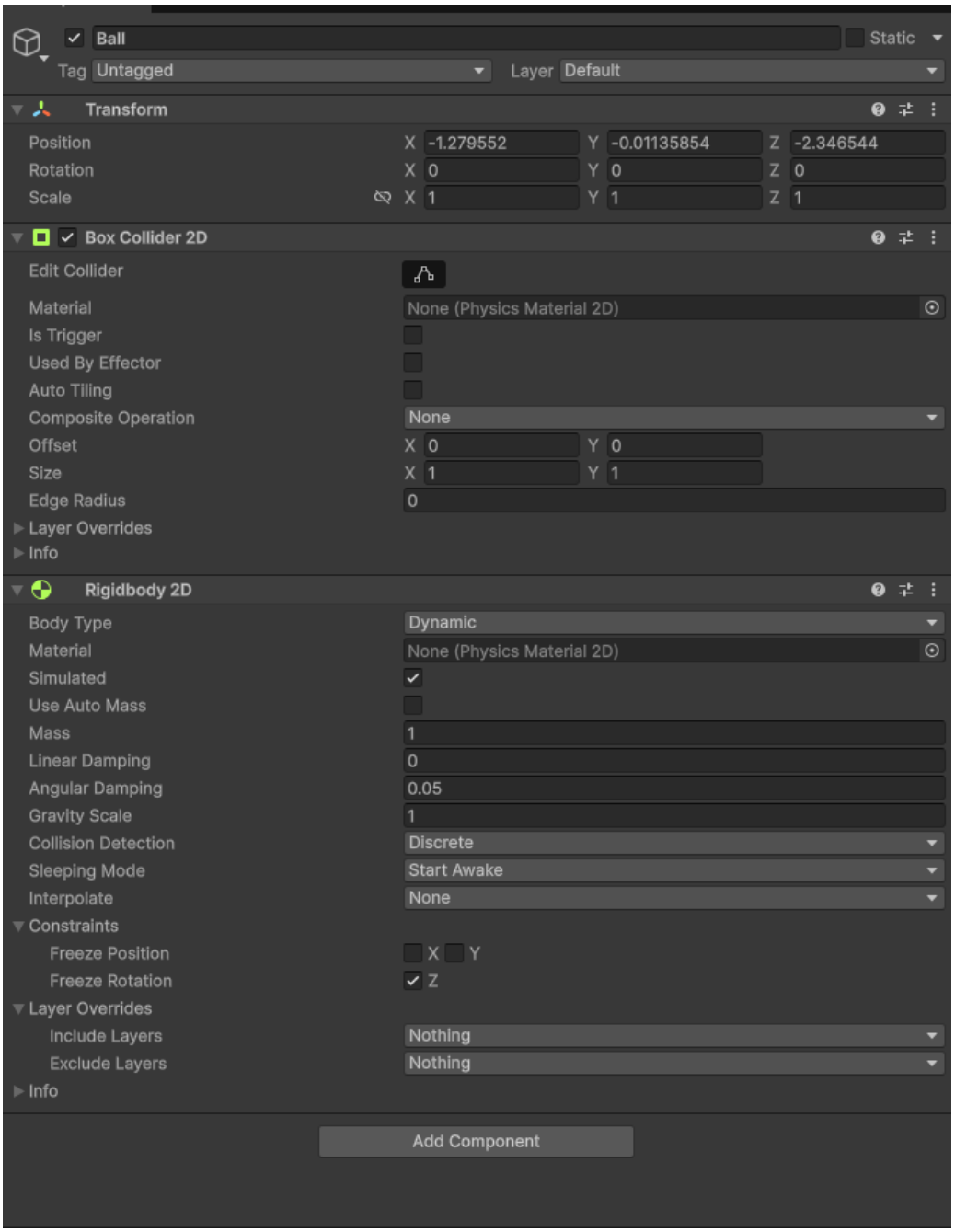

Рисунок В.9 – Вигляд інспектора у якому прописані налаштування за допомогою методу Rigidbody 2D.

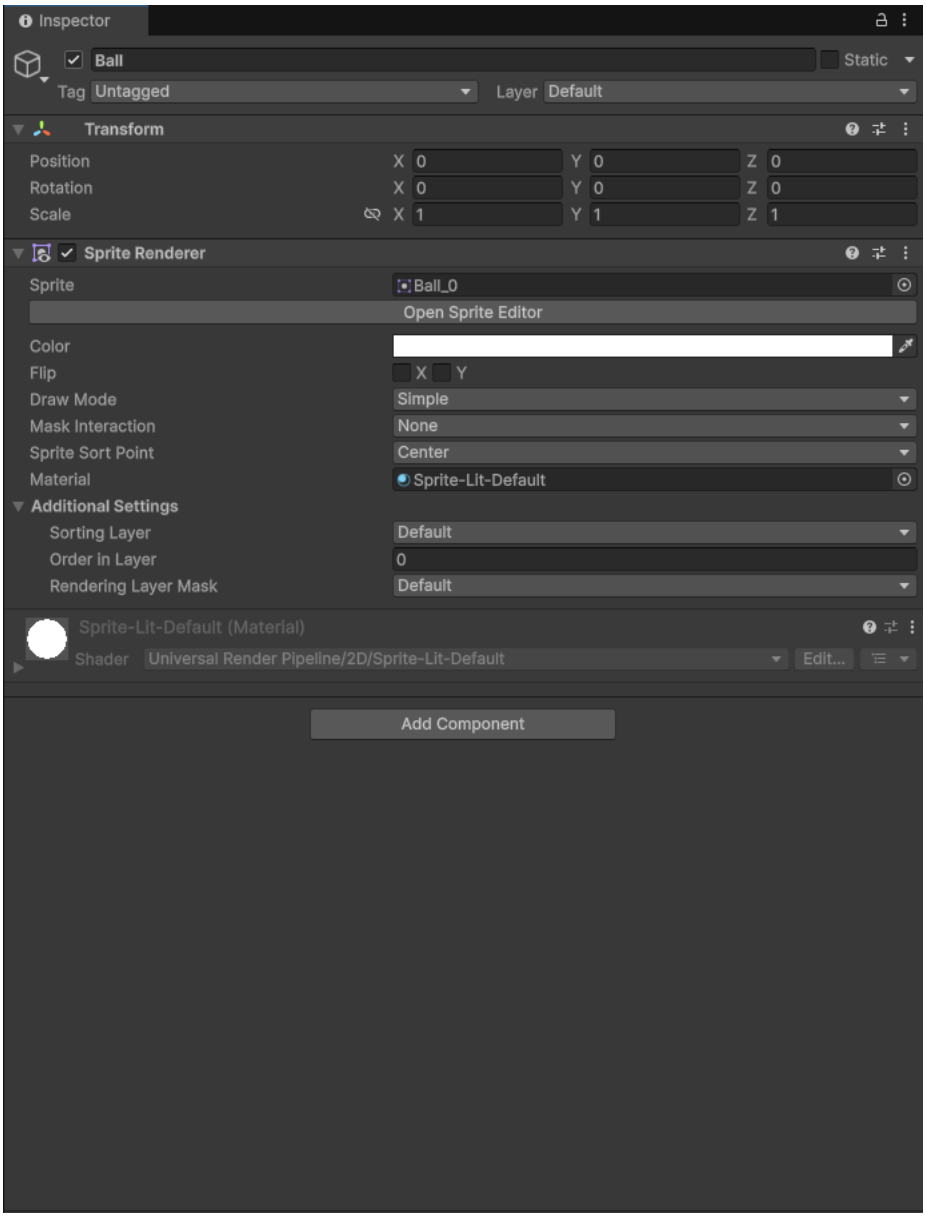

Рисунок В.10 – Вигляд інспектора у якому вказані спрайти з текстурою об’єкта.

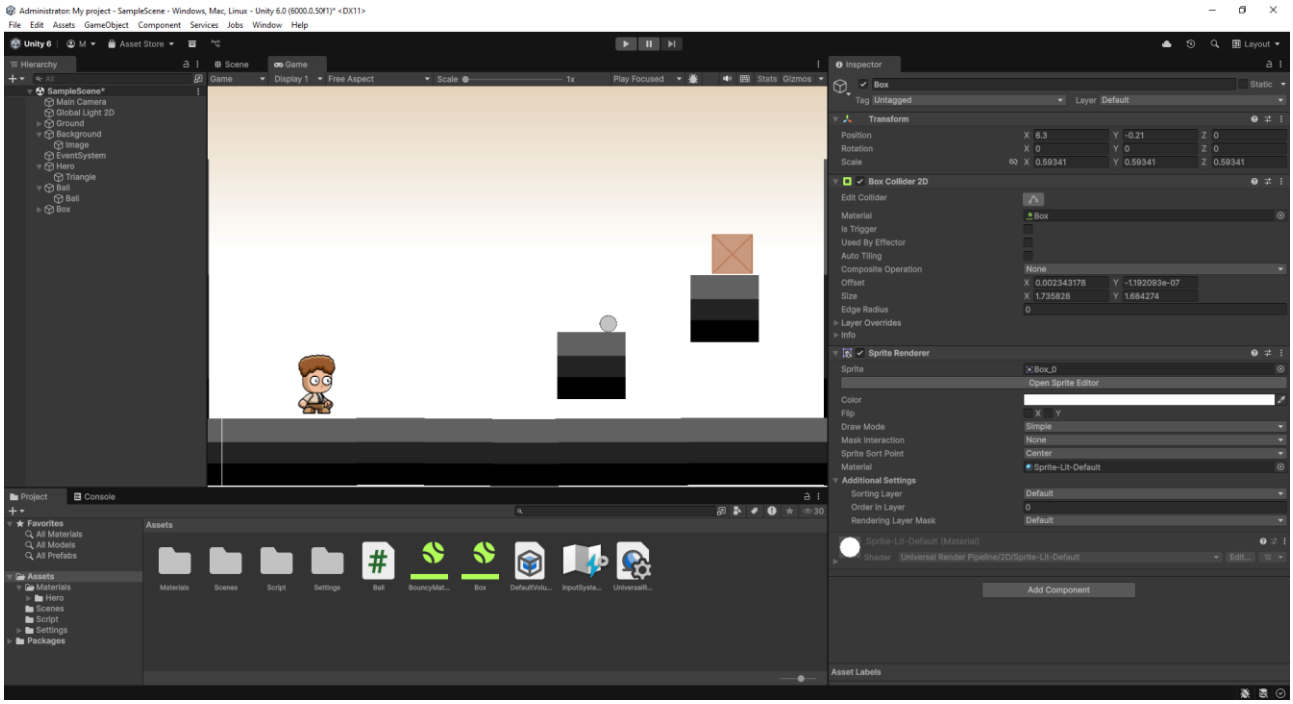

Рисунок В.11 – Загальний вигляд рушія Unity.

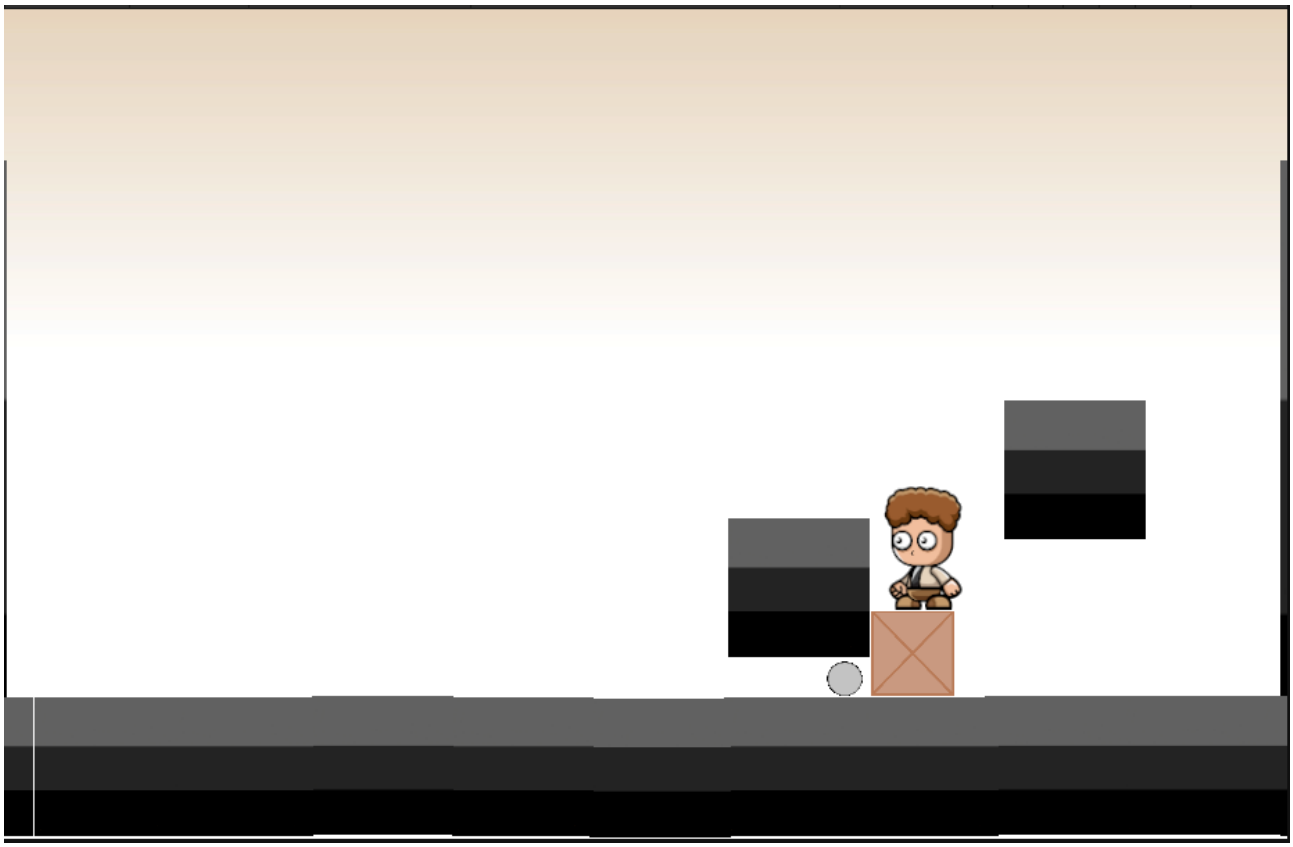

Рисунок В.12 – Взаємодія між об’єктами

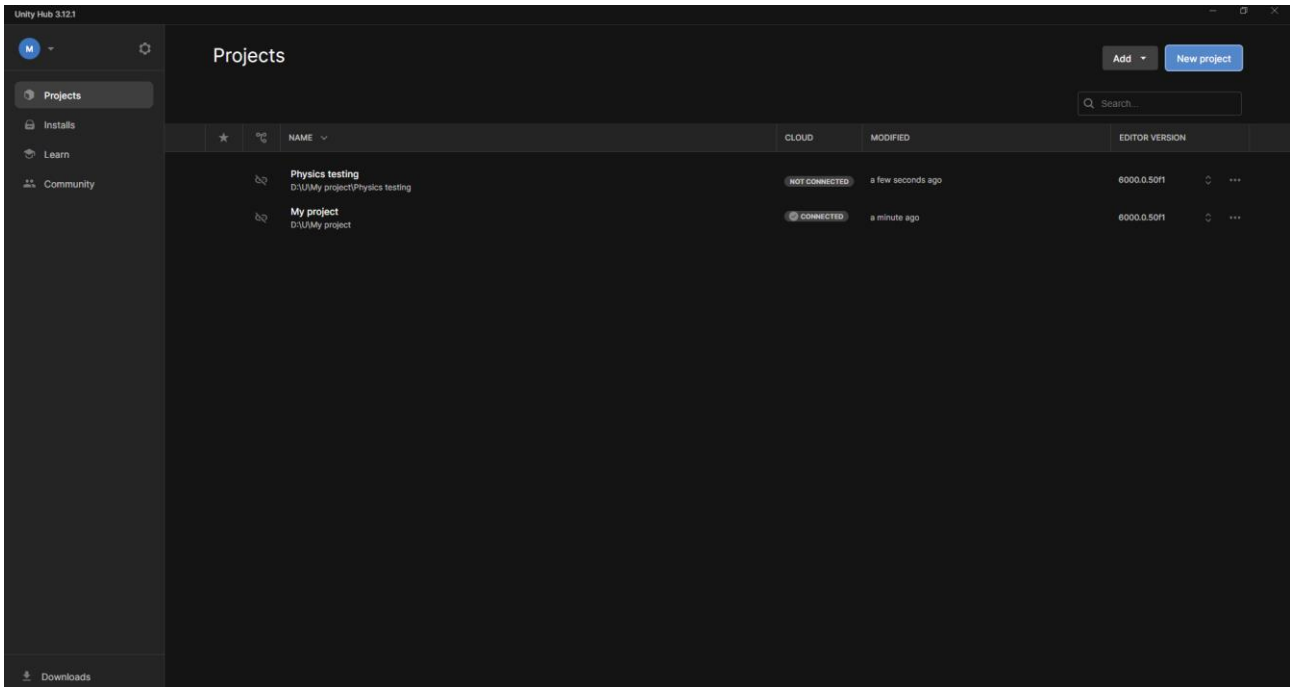

Рисунок В.13 – Unity Hub версії 3.12.1

ДОДАТОК Г (ДОВІДНИКОВИЙ)

Інструкція користувача

1) Запускаємо рушій Unity відповідної версії.

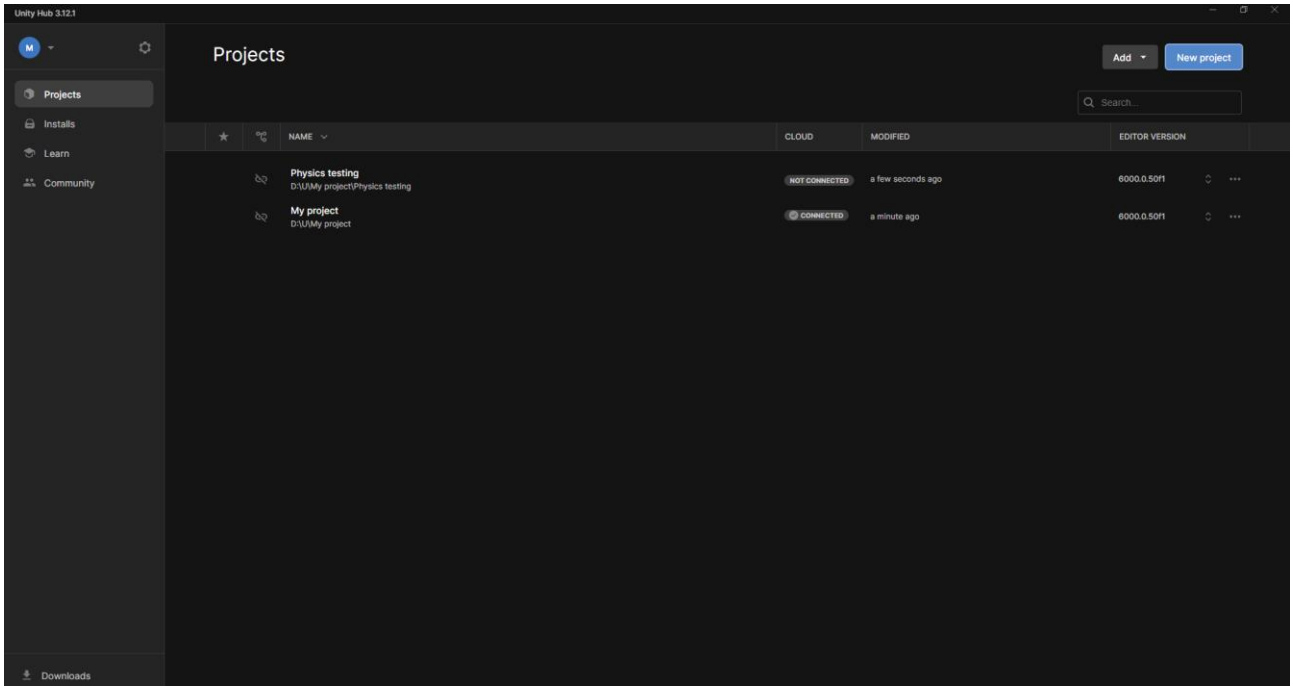

Рисунок Г.1 – Unity Hub версії 3.12.1

2) Вибираємо та натискаємо на проект “Physics testing”

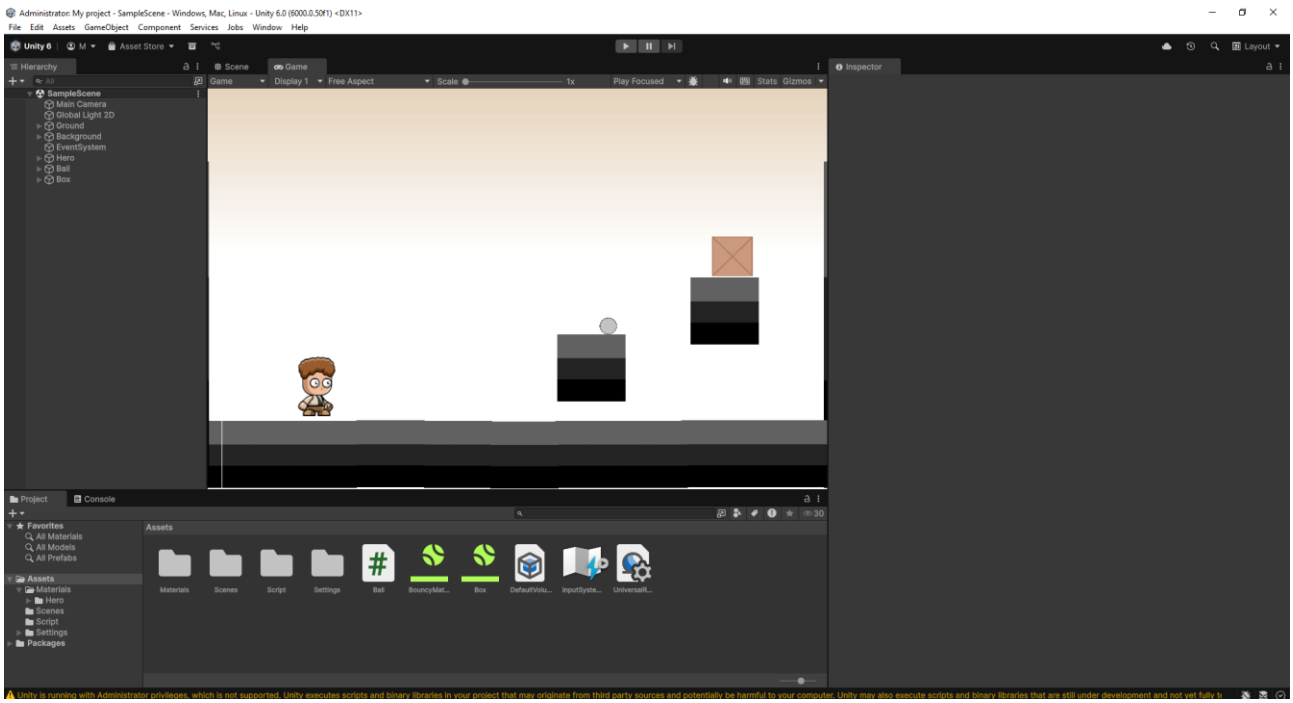

Рисунок Г.2 – Вигляд вікна редагування Unity

3) Натискаємо кнопку “Play” у верхній центральній частині екрану, користувач отримує можливість тестування та спостерігання фізичних взаємодій у 2D просторі
4) Щоб вийти із програми достатньо закрити програму натиснувши кнопку завершення у правій верхній частині екрану.