МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНИЙ ПРОЄКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Інтерактивна модель Сонячної системи у середовищі Unity 3D"

Виконавець: студент групи УС-411 Щербина Костянтин Юрійович

Керівник: к.т.н., доцент Райчев Ігор Едуардович

Нормоконтролер: Олександр ШЕВЧЕНКО

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2023р.

ЗАВДАННЯ

на виконання дипломного проєкту студента

Щербини Костянтина Юрійовича

1. **Тема проєкту:** «Інтерактивна модель Сонячної системи у середовищі Unity 3D» затверджена наказом ректора № 623/ст. від 01.05.2023р.
2. **Термін виконання проєкту:** з 15.05.2023р. по 25.06.2023р.
3. **Вихідні дані до проєкту:** середовище розробки Unity 3D версії 2021.3.16f1 Personal, мова програмування C#, список та текстури астрономічних об'єктів, характеристики астрономічних об'єктів, методи моделювання руху, математичні формули для обчислення параметрів, перелік параметрів для виведення, вимоги до інтерактивності.
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** методи та інструменти розробки інтерактивної моделі Сонячної системи у середовищі Unity 3D, проєктування архітектури системи, реалізація у середовищі Unity 3D.
5. **Перелік обов'язкового графічного матеріалу:** Слайди презентації MS PowerPoint: діаграма варіантів використання, діаграми класів, скріншоти інтерфейсу користувача, результати моделювання.

6. Календарний план-графік

№ з/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Визначити вимоги та цілі проєкту, створити візуальний концепт моделі	15.05.2023 – 17.05.2023	
2.	Створити базову сцену в Unity, імпортувати необхідні матеріали в проєкт, задати базові характеристики об’єктам і реалізувати гравітаційну взаємодію	18.05.2023 – 23.05.2023	
3.	Задати алгоритми обчислень параметрів об’єктів у реальному часі, реалізувати інтерактивні елементи	24.05.2023 – 29.05.2023	
4.	Розробити інтерфейс користувача, систему введення і виведення параметрів	30.05.2023 – 04.06.2023	
5.	Виконати тестування і налагодження	05.06.2023 – 07.06.2023	
6.	Написати пояснювальну записку дипломного проєкту	08.06.2023 – 12.06.2023	
7.	Підготувати демонстраційний матеріал та доповідь	13.06.2023 – 16.06.2023	

7. Дата видачі завдання: «15» травня 2023 р.

Керівник кваліфікаційної роботи _____ Ігор РАЙЧЕВ

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Костянтин ЩЕРБИНА

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Інтерактивна модель Сонячної системи у середовищі Unity 3D»: 99 с., 27 рис., 17 літературних джерел, 4 додатки.

Об’єкт дослідження: програмний продукт у середовищі Unity 3D.

Предмет дослідження: програмне забезпечення, яке буде точно моделювати Сонячну систему, включаючи реалістичну візуалізацію об’єктів, правильні орбіти та рух, а також можливість взаємодії користувача з моделлю.

Мета дипломного проєкту: розробка інтерактивної моделі Сонячної системи у середовищі Unity 3D, що дозволяє користувачам досліджувати та візуалізувати астрономічні об’єкти, їх орбіти, рух і поведінку у реальному часі.

Методи дослідження: аналіз джерел з астрономічними даними, моделювання руху об’єктів, розробка інтерфейсу користувача, тестування і налагодження.

Результатом дипломного проєкту є інтерактивна модель Сонячної системи з фізичною взаємодією між об’єктами, можливістю додавання та видалення об’єктів, керування камерою, введення та виведення характеристик об’єктів.

ІНТЕРАКТИВНА МОДЕЛЬ, СОНЯЧНА СИСТЕМА, UNITY 3D, АСТРОНОМІЯ, АСТРОНОМІЧНИЙ ОБ’ЄКТ, НЕБЕСНА МЕХАНІКА, ОРБІТА, ІНТЕРФЕЙС КОРИСТУВАЧА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Unity 3D	інтегроване середовище розробки для створення візуальних ефектів та інтерактивних додатків.
Скрипт	сценарій, який написаний мовою програмування для взаємодії об’єктів у додатку.
Сцена	містить об’єкти та їх розміщення у віртуальному середовищі Unity.
Rigidbody	компонент Unity, що відповідає за фізичну поведінку об’єктів.
Awake	функція Unity, яка викликається автоматично під час завантаження сцени або активації об’єкта, до початку виконання інших методів
Start	функція Unity, яка виконується один раз при старті сцени і використовується для початкового налаштування об’єктів та ініціалізації змінних перед початком виконання додатку.
Update	функція Unity, яка виконується кожен кадр і використовується для оновлення логіки додатку.
FixedUpdate	функція Unity, яка виконується з фіксованою частотою і використовується для фізичних обчислень.
Prefab	префаб Unity, який представляє готовий до використання шаблон об’єкта.
Кеплерові елементи орбіти	характеристики орбіти, такі як велика піввісь, ексцентриситет, нахил орбіти, довгота висхідного вузла, аргумент перицентру і середня аномалія.
Skybox	спосіб відображення фону за допомогою текстури, яка охоплює всю сцену.
Quaternion	тип даних, який використовується для представлення обертання або орієнтації об’єктів у тривимірному просторі.

ВСТУП

Актуальність теми «Інтерактивна модель Сонячної системи у середовищі Unity 3D» полягає в тому, що сучасні досягнення в галузі комп'ютерної графіки дозволяють створювати переконливі та реалістичні візуалізації космічних об'єктів. Також зростаючий інтерес суспільства до астрономії та космосу вимагає доступних та зрозумілих інструментів для вивчення та візуалізації нашої Сонячної системи. Крім того, існуючі моделі Сонячної системи часто обмежені у функціональності та не забезпечують повноцінної взаємодії користувача з моделлю. Тому розробка інтерактивної моделі Сонячної системи у середовищі Unity 3D є актуальним та перспективним напрямом, що дозволяє надати інформацію про космос більш наочно та доступно для широкої аудиторії.

Мета дипломного проєкту «Інтерактивна модель Сонячної системи у середовищі Unity 3D» полягає у дослідженні, розробці та створенні інтерактивної моделі Сонячної системи з використанням середовища Unity 3D. Для досягнення цієї мети необхідно вирішити такі завдання:

- вивчити основи астрономії та фізики, пов'язані з рухом та взаємодією небесних тіл у Сонячні системі;
- вивчити можливості та інструменти, що надаються середовищем Unity 3D;
- розробити архітектуру моделі Сонячної системи, визначити об'єкти, їх параметри та взаємозв'язки;
- реалізувати візуалізацію об'єктів Сонячної системи;
- розробити алгоритми для розрахунку параметрів орбіт та руху небесних тіл у моделі;
- створити інтерактивні елементи керування та функції, що дозволяють користувачеві взаємодіяти з моделлю Сонячної системи;
- провести тестування та оптимізацію моделі для забезпечення її плавної роботи.

Об’єкт дослідження: програмний продукт у середовищі Unity 3D.

Предмет дослідження: програмне забезпечення, яке буде точно моделювати Сонячну систему, включаючи реалістичну візуалізацію об’єктів, правильні орбіти та рух, а також можливість взаємодії користувача з моделлю.

Для досягнення поставленої мети в рамках даного дипломного проєкту використовувалися такі методи дослідження:

- Вивчення літератури та аналіз джерел. Проведено аналіз наукових та технічних статей, книг, онлайн-ресурсів та інших матеріалів, пов’язаних із моделюванням Сонячної системи та середовищем Unity 3D. Це дозволило отримати необхідну теоретичну базу та огляд існуючих підходів до моделювання.
- Проєктування та розробка програмного продукту. Було проведено процес проєктування інтерактивної моделі Сонячної системи, визначено вимоги до функціональності та візуалізації. Потім була розроблена програма, використовуючи середовище Unity 3D, що включає моделювання орбітальних рухів, візуалізацію небесних тіл та взаємодію з користувачем.
- Тестування та оцінка. Проведено тестування розробленої інтерактивної моделі з метою виявлення помилок, перевірки коректності орбітальних рухів та загальної працездатності програмного продукту. Оцінка проводилася шляхом порівняння результатів моделювання з відомими фактами та спостереженнями.
- Аналіз результатів та висновки. Після проведення дослідження та аналізу результатів моделювання були зроблені висновки про якість та ефективність розробленої інтерактивної моделі Сонячної системи.

Використання вказаних методів дослідження дозволило досягти поставленої мети та розробити інтерактивну модель Сонячної системи у середовищі Unity 3D, що має реалістичну візуалізацію та правильні орбітальні рухи небесних тіл, а також можливість взаємодії користувача з моделлю.

У рамках дипломного проєкту «Інтерактивна модель Сонячної системи у середовищі Unity 3D» було отримано такі наукові новизни:

- розроблена інтерактивна модель Сонячної системи у середовищі Unity 3D є комплексною системою, що поєднує реалістичну візуалізацію небесних тіл, правильні орбітальні рухи та можливість взаємодії користувача з моделлю;
- розроблено та реалізовано нові алгоритми та методи моделювання орбітальних рухів, які забезпечують точність та достовірність відтворення реальних орбіт небесних тіл у Сонячній системі;
- реалізована інтерактивна модель дозволяє користувачам досліджувати та взаємодіяти із Сонячною системою, включаючи можливість зміни параметрів, спостереження за рухом планет, а також отримання інформації про кожне небесне тіло.

Отже, розроблена інтерактивна модель Сонячної системи у середовищі Unity 3D є значною науковою новизною, надаючи користувачеві можливість зануритися в унікальний віртуальний досвід та дослідити Сонячну систему у новому інтерактивному форматі.

Отримані результати дипломного проєкту мають таке практичне значення:

- Для освітньої сфери – розроблена інтерактивна модель може бути використана в освітніх закладах для навчання астрономії та вивчення Сонячної системи. Студенти можуть взаємодіяти з моделлю, досліджувати орбіти планет, вивчати їх характеристики та отримувати інформацію про небесні тіла. Це допомагає візуалізувати складні концепції та сприяє глибшому розумінню астрономії.
- Для наукової сфери – інтерактивна модель може бути використана у наукових дослідженнях у галузі астрономії та космології. Вчені можуть використовувати модель для вивчення динаміки Сонячної системи, аналізу взаємодій між небесними тілами та проведення різних експериментів. Це може сприяти новим відкриттям та поглибленому розумінню процесів, що відбуваються у Сонячній системі.
- Для популяризації астрономії – інтерактивна модель може бути використана у розважальних, науково-популярних центрах та інших місцях для

популяризації астрономії. Відвідувачі зможуть взаємодіяти з моделлю, вивчати Сонячну систему та отримувати нові знання про космос. Це допомагає залучити інтерес до науки та підвищити загальну астрономічну грамотність.

Рекомендації щодо використання отриманих результатів включають:

- впровадження моделі в освітні програми та навчальні плани для навчання астрономії у школах та університетах;
- створення віртуальних астрономічних екскурсій та навчальних програм на основі інтерактивної моделі для широкої аудиторії;
- використання моделі у наукових дослідженнях для вивчення динаміки Сонячної системи та проведення експериментів;
- інтеграція моделі на публічні платформи для популяризації астрономії.

Таким чином, отримані результати мають значне практичне значення та можуть бути застосовані в освітній сфері, наукових дослідженнях та популяризації астрономії для збагачення знань та інтересу до космосу.

Особистий внесок випускника включає:

- вивчення та аналіз існуючих методів моделювання Сонячної системи та інтерактивних програм у середовищі Unity 3D;
- проєктування архітектури та функціональності інтерактивної моделі, включаючи візуалізацію об’єктів та правильний рух небесних тіл;
- розробка програмного коду та реалізація моделі у середовищі Unity 3D, забезпечуючи точне моделювання та взаємодію користувача з моделлю;
- розробка алгоритмів обчислення орбітальних параметрів для об’єктів у моделі;
- тестування та налагодження моделі, виявлення та усунення помилок та недоліків.

Особистий внесок випускника полягає у здійсненні всіх етапів розробки проєкту, застосуванні знань та навичок у галузі програмування та астрономії, а

також уважному та ретельному підході до деталей проєкту для досягнення поставлених цілей.

РОЗДІЛ 1
МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ІНТЕРАКТИВНОЇ МОДЕЛІ СОНЯЧНОЇ СИСТЕМИ У СЕРЕДОВИЩІ UNITY 3D

1.1. Основні поняття

Сонячна система (рис. 1.1) – це одна із безлічі планетарних систем у галактиці Чумацький Шлях. Вона складається із однієї зорі класу «жовтий карлик», восьми планет, п’яти карликових планет, а також багатьох супутників, астероїдів, комет та інших об’єктів. Зоря у цій системі названа Сонцем. Навколо нього в одному напрямку обертається вісім планет, перераховані у порядку зростання відстані до зорі: Меркурій, Венера, Земля, Марс, Юпітер, Сатурн, Уран і Нептун [1]. Перші чотири з них – кам’яні, а останні чотири – газові гіганти, зокрема Уран і Нептун відносяться до підкласу льодяних гігантів. Також навколо більшості планет обертаються супутники, і найбільші серед них: Ганімед, Титан, Калісто, Іо, Місяць. Відомо п’ять карликових планет: Плутон, Ерида, Макемаке, Гаумеа, Церера [2].

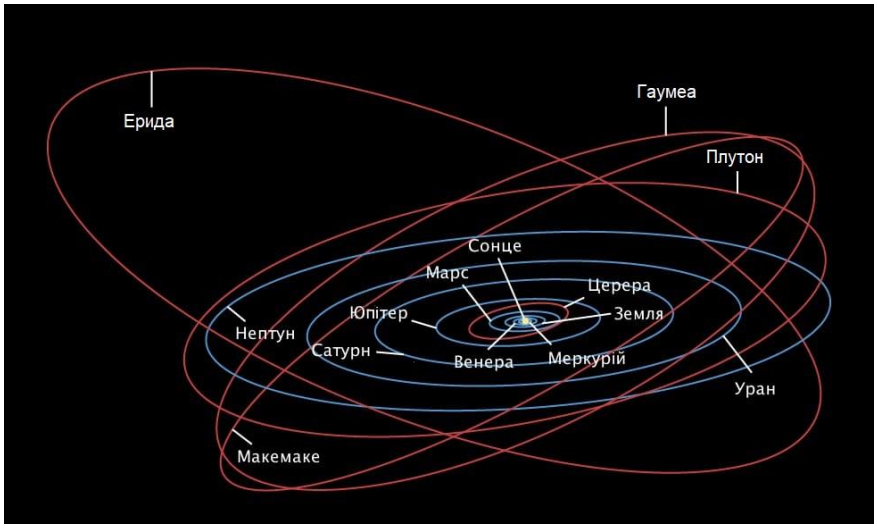

Рис. 1.1. Вигляд орбіт об’єктів Сонячної системи з дотриманням масштабу

Unity 3D – це кросплатформний ігровий рушій та інтегроване середовище розробки, у якому виконується розробка 2D та 3D відеоігор, симуляцій та будь-яких інших програм. Перевагою Unity 3D є те, що він простий у використанні та має широкий спектр функцій та інструментів, зокрема підтримку мови програмування C# для написання сценаріїв [3]. Оскільки цей рушій є кросплатформним, то він підтримує велику кількість платформ, тобто створені на ньому відеоігри можна легко портувати між операційними системами Windows, Linux, OS X, Android, iOS, консолями PlayStation, Xbox, Nintendo, а також пристроями віртуальної та доповненої реальності. Таким чином, Unity 3D є зручним та ефективним інструментом для розробки будь-яких відеоігор, навіть високоякісних, для різних платформ. У нашому випадку, цей рушій можна використати для створення інтерактивної моделі Сонячної системи.

Складові Сонячної системи у середовищі Unity 3D створюються за допомогою ігрових об’єктів. Ігровий об’єкт – це фундаментальний будівельний блок, який використовується для представлення будь-якого предмета чи елемента, який з’явиться в сцені, наприклад персонажів, перешкод, декорацій тощо. Кожен ігровий об’єкт має властивості, які визначають його зовнішній вигляд, поведінку та функціональність [4]. Ігровий об’єкт “Sphere” зображений на рис. 1.2.

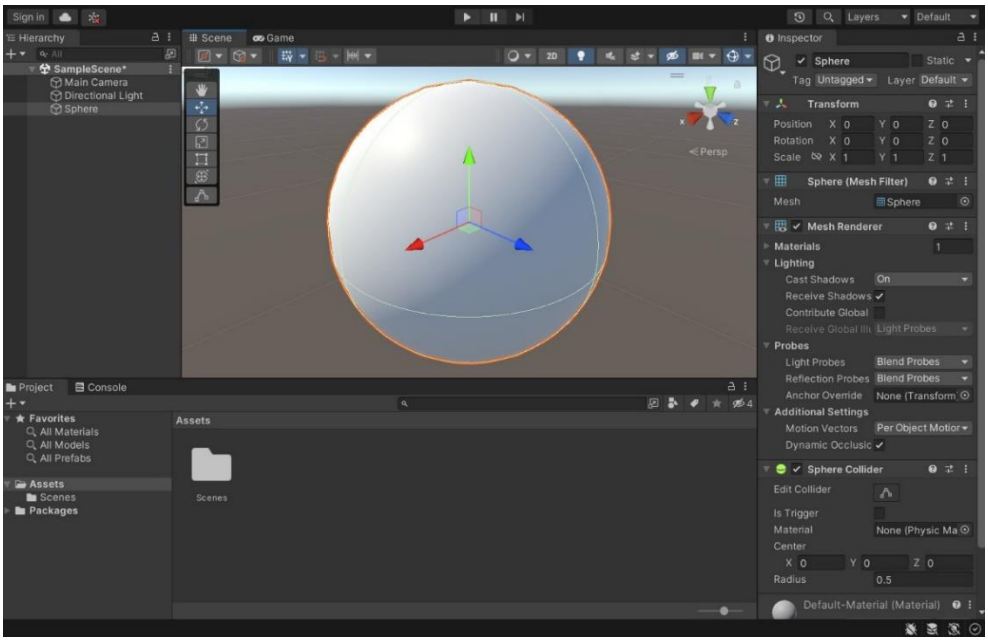

Рис. 1.2. Ігровий об’єкт у сцені

Компоненти – це сценарії або інші ресурси, які можна прикріпити до ігрових об’єктів в Unity 3D. Тобто компоненти можуть включати візуальні та звукові ресурси, сценарії, за допомогою яких визначається поведінка ігрового об’єкта, фізичні властивості, які дозволяють йому взаємодіяти з іншими ігровими об’єктами, тощо. І кожен ігровий об’єкт містить компонент трансформації, який визначає його положення у просторі, кути повороту та масштаб. Компоненти ігрового об’єкта можна побачити у правій панелі (див. рис. 1.2).

Сценарії в Unity 3D керують поведінкою та функціональністю об’єктів. У цьому рушії підтримується мова програмування C#, яку можна використовувати для написання сценаріїв. Сценарії можна прикріпляти до об’єктів, і вони будуть виконуватись під час виконання, завдяки чому буде забезпечена певна поведінка об’єкта. У даному випадку, наприклад, сценарій можна використати для керування напрямком і швидкістю переміщення космічного об’єкта (ігрового об’єкта), яке буде залежати від властивостей даного об’єкта та всіх інших об’єктів на сцені, а також фізичних законів. Щоб створити сценарій в Unity 3D, потрібно на панелі Assets (рис. 1.3) натиснути правою кнопкою миші і натиснути Create > C# Script, і, щоб прикріпити його до ігрового об’єкта, можна просто перетягнути його з панелі Assets до панелі з компонентами, або натиснути кнопку “Add Component” і вибрати в меню потрібний компонент.

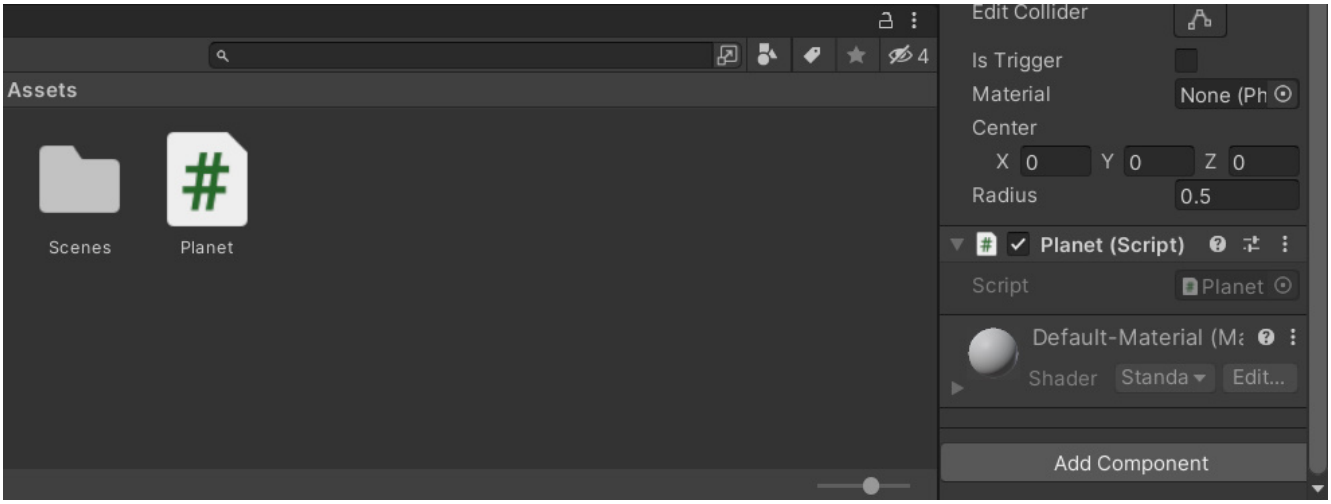

Рис. 1.3. Створення і прикріплення компонента сценарію

Щоб забезпечити зручне керування моделлю, необхідно створити інтерфейс користувача. Інтерфейс користувача в Unity 3D – це набір графічних елементів, які дозволяють користувачу взаємодіяти з програмою. Елементами інтерфейсу користувача можуть бути кнопки, поля, повзунки тощо.

1.2. Можливості Unity 3D

Unity 3D надає набір інструментів і функцій, які можна використовувати для створення графічної моделі Сонячної системи. Для цього можна використовувати наведені далі можливості [5].

Керування сценою. Unity 3D дозволяє створювати кілька сцен, кожну з яких можна використовувати для представлення іншої планети або небесного тіла в Сонячній системі.

3D-графіка. Надаються інструменти для створення 3D-графіки, включаючи різні параметри освітлення та затінення, які можна використовувати для імітації ефектів різних умов освітлення на кожному астрономічному об’єкті.

Текстури. Надається можливість накласти на 3D-об’єкти зображення, щоб, наприклад, зробити вигляд сфери подібним до справжньої планети, а також можна додавати кратери, гори тощо.

Фізичне моделювання. Міститься вбудований фізичний механізм, який можна використовувати для моделювання руху планет, астероїдів та інших небесних тіл у Сонячній системі.

Система частинок. Надається система частинок, яку можна використовувати для створення ефектів, таких як сонячні спалахи, випаровування та інші явища, які відбуваються в Сонячній системі.

Сценарії. Розробники можуть використовувати мову програмування C# для створення сценаріїв, які можна використовувати для задання поведінки об’єктам, реалізації інтерактивності та функціональності моделі Сонячної системи.

Інтерфейс користувача. Забезпечуються інструменти для створення інтерфейсів користувача, до яких можуть входити кнопки, повзунки, поля та інше,

що можна використовувати для надання користувачам можливості керувати моделлю.

Використовуючи ці та інші функції Unity 3D, розробники можуть створити реалістичну інтерактивну модель Сонячної системи.

1.3. Обмеження Unity 3D

Хоча Unity 3D – це потужний ігровий рушій із багатьма функціями та можливостями, але, як і будь-яке інше програмне забезпечення, він має свої обмеження. Деякі обмеження наведено далі.

Обмеження продуктивності. Unity 3D може бути ресурсомістким, особливо під час роботи зі складними сценами або великою кількістю ігрових об’єктів, тому можуть виникнути проблеми з продуктивністю, якщо не застосовувати певні методи оптимізації.

Обмеження на відстань. Чим далі об’єкт від початку координат, тим більше буде його координата і тим меншою буде точність її представлення (рис. 1.4). Через це рушій Unity 3D не може правильно відобразити об’єкт на екрані. Якщо модуль значення координати буде більший за 100000, об’єкт зникне, і рушій відобразить наступне повідомлення: «Через обмеження точності числа з плаваючою комою, рекомендовано вводити світові координати GameObject у меншому діапазоні». Цю проблему можна вирішити, наприклад, переміщуючи сцену ближче до початку координат, або використовуючи кілька сцен, або використовуючи деякі техніки для роботи з великими координатами, як масштабування об’єктів або використання чисел с плаваючою крапкою з подвійною точністю [6].

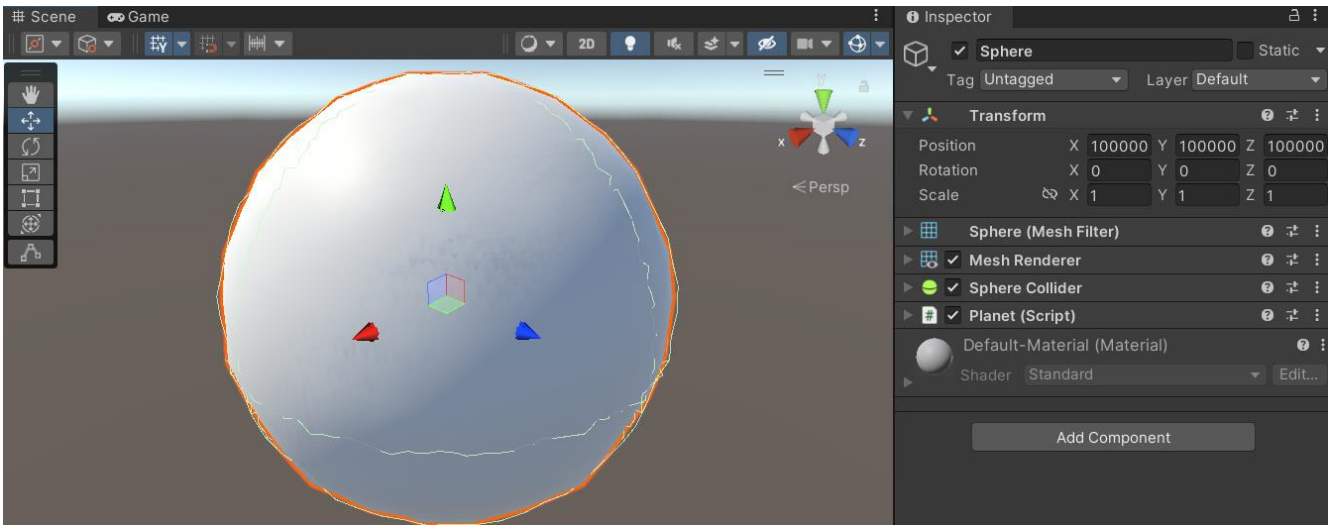

Рис. 1.4. Об’єкт на великих координатах відображається представляється не точно

Обмеження на кількість вершин у сітці об’єкта. В Unity 3D може підтримуватися до 65536 вершин у сітці при 16-бітному буфері індексу, або до 4 мільярдів при 32-бітному. Буфер індексу – це частина графічного процесору, яка використовується для створення 2D зображень з 3D об’єктів («рендеринг»), і в ній міститься послідовність індексів вершин 3D об’єкта. За замовчуванням формат індексу 16-бітний, оскільки він потребує менше пам’яті та пропускної здатності.

Відсутність вбудованої підтримки багатокористувацьких ігор. Хоча Unity 3D пропонує деякі функції для кількох гравців, він не надає можливості створити вбудований сервер, який приймає підключення від інших копій даного програмного продукту.

Обмеження в Unity 3D існують, оскільки рушій працює на пристроях, які мають обмежені обчислювальні можливості, пам’ять тощо. Також деякі обмеження введені для того, щоб забезпечити сумісність з певними платформами, і щоб допомогти оптимізувати продуктивність рушія.

1.4. Математичні моделі для опису руху астрономічних об’єктів

Для створення інтерактивної моделі Сонячної системи у середовищі Unity 3D необхідні математичні моделі для опису руху астрономічних об’єктів.

Закони Кеплера. Ці закони описують рух планет навколо Сонця. Вони базуються на спостереженнях астронома Йоганна Кеплера і стверджують, що планети рухаються по еліптичних орбітах навколо Сонця, причому Сонце знаходиться в одному з фокусів еліпса.

Закони Ньютона. Ці закони описують рух об’єктів під дією сил. Вони є основою класичної механіки і використовуються для обчислення руху планет, супутників та інших астрономічних об’єктів.

Закон всесвітнього тяжіння. Цей закон описує силу тяжіння між двома об’єктами. У ньому стверджується, що кожен об’єкт у Всесвіті притягує будь-який інший об’єкт із силою, яка пропорційна добутку їх мас і обернено пропорційна квадрату відстані між ними.

1.4.1. Закони Кеплера

Закони руху планет Кеплера – це три закони, які описують рух планет навколо Сонця. Їх сформулював німецький астроном Йоганн Кеплер на початку XVII століття на основі спостережень Тихо Браге. Закони такі [7]:

1) Перший закон Кеплера (закон еліпсів) (рис. 1.5). Кожна планета Сонячної системи рухається по еліпсу, в одному з фокусів якого знаходиться Сонце.

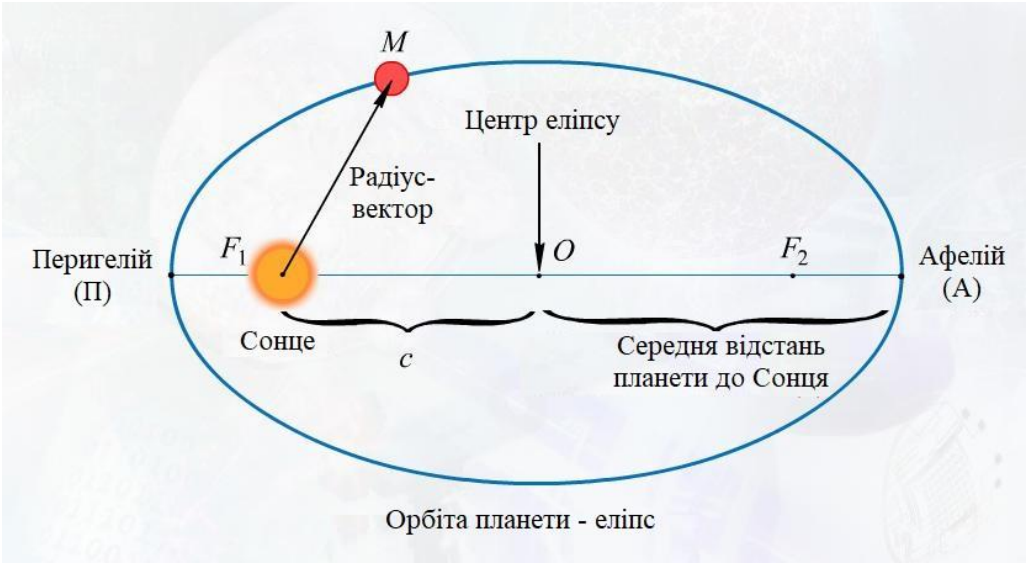

Рис. 1.5. Перший закон Кеплера

Оскільки планета рухається по орбіті, що описує еліпс, то відстань від неї до Сонця змінюється постійно під час обертання навколо Сонця.

Форма еліпсу та ступінь його подібності до кола характеризується відношенням

$$e = \frac{c}{a}, \tag{1.1}$$

де c – відстань від центру еліпса до його фокусу (фокальна відстань), a – велика піввісь. Величина e називається ексцентриситетом еліпсу. При c = 0, і, отже, e = 0, еліпс перетворюється в коло.

Причому відстань від центру еліпса до його фокусу обчислюється за наступною формулою:

$$c = \sqrt{a^2 - b^2}, \tag{1.2}$$

де a – велика піввісь, b – мала піввісь.

2) Другий закон Кеплера (закон площ) (рис. 1.6). Кожна планета рухається в площині, що проходить через центр Сонця, причому за рівні проміжки часу радіус-вектор, що з'єднує Сонце і планету, описує собою рівні площини.

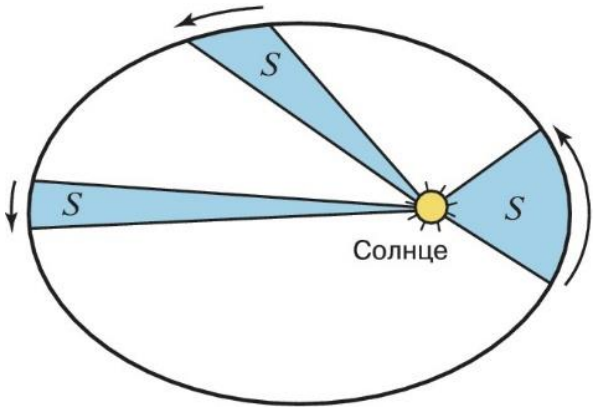

Рис. 1.6. Другий закон Кеплера

Це означає, що швидкість планети на орбіті не постійна, а змінюється. Найбільш віддалену точку на орбіті планети від Сонця називають афелієм, а найближчу – перигелієм. Таким чином, планета рухається найшвидше в перигелії та найповільніше в афелії.

3) Третій закон Кеплера (гармонічний закон) (рис. 1.7). Квадрати періодів обертання планет навколо Сонця ставляться, як куби великих півосей орбіт планет.

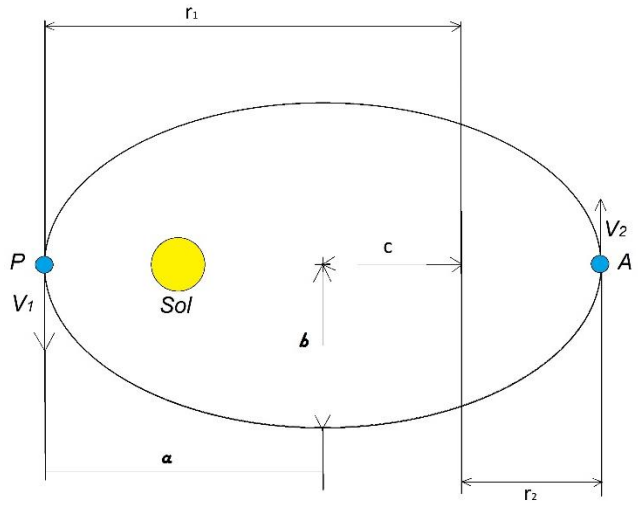

Рис. 1.7. Третій закон Кеплера

Цей закон стверджує, що квадрати періодів обертання планет прямо пропорційні кубам великих півосей їхніх орбіт. Це означає, що збільшення відстані між планетою та Сонцем призводить до збільшення часу, необхідного для повного обертання навколо Сонця. За цим законом, Меркурій, який перебуває ближче до Сонця, обертається навколо нього швидше, ніж Земля або інший більш віддалений об’єкт.

$$\frac{T_1^2}{T_2^2} = \frac{a_1^3}{a_2^3}, \tag{1.3}$$

де T_1 і T_2 – періоди обертання двох планет навколо Сонця, а a_1 і a_2 – довжини великих півосей їхніх орбіт.

Закони Кеплера відіграли фундаментальну роль у розвитку сучасної астрономії та дослідженні космосу. Їх використовували для розрахунку орбіт планет, супутників та інших небесних тіл, а також відіграли важливу роль у проєктуванні та експлуатації космічних кораблів. У випадку створення інтерактивної моделі Сонячної системи в Unity 3D закони Кеплера можна використовувати для точного моделювання орбіт планет навколо Сонця, дозволяючи користувачам спостерігати за Сонячною системою та взаємодіяти з нею.

1.4.2. Закони Ньютона

Закони Ньютона руху планет дають математичний опис руху небесних тіл у Сонячній системі. Ці три закони такі [8]:

1) Закон інерції. Об’єкт у стані спокою залишається в спокої, а об’єкт у русі продовжує рухатися з постійною швидкістю та прямолінійно, якщо на нього не діє неврівноважена сила.

$$\vec{F} = 0 \rightarrow \vec{a} = 0, \tag{1.4}$$

де $\vec{F}$ – врівноважена сила, що діє на об’єкт, $\vec{a}$ – прискорення об’єкта.

2) Закон прискорення. Прискорення об’єкта залежить від маси об’єкта та величини прикладеної сили. Це означає, що сила дорівнює зміні імпульсу (маси, помноженої на швидкість) на зміну в часу. Імпульс визначається як маса об’єкта, помножена на його швидкість.

$$\vec{a} = \frac{\vec{F}}{m}, \tag{1.5}$$

де $\vec{a}$ – прискорення матеріальної точки, $\vec{F}$ – рівнодіюча всіх сил, прикладених до матеріальної точки, m – маса матеріальної точки.

3) Закон рівності дії та протидії. Кожного разу, коли один об’єкт чинить силу на інший об’єкт, другий об’єкт чинить таку ж протилежну силу на перший.

$$\overrightarrow{F_1} = -\overrightarrow{F_2}, \tag{1.6}$$

де $\overrightarrow{F_1}$ – сила, з якою перше тіло діє на друге, $\overrightarrow{F_2}$ – сила, з якою друге тіло діє на перше.

Ці закони можна використовувати для розрахунку орбіт небесних тіл і прогнозування їхнього майбутнього положення та руху. У середовищі Unity 3D їх можна запрограмувати в інтерактивну модель Сонячної системи для створення реалістичної та точної симуляції руху планет та інших об’єктів.

1.4.3. Закон всесвітнього тяжіння

Закон всесвітнього тяжіння – фундаментальний закон фізики, який пояснює силу тяжіння між будь-якими двома об’єктами у Всесвіті. Він також відкритий І. Ньютоном.

Закон стверджує, що сила F гравітаційного тяжіння між двома матеріальними точками (об’єктами) з масами m_1 і m_2, розділеними відстанню r, діє вздовж прямої, що з’єднує їх, пропорційна обом масам і обернено пропорційна квадрату відстані:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}, \tag{1.7}$$

де G – гравітаційна стала, що дорівнює $6{,}67430(15)\cdot 10^{-11}\ \frac{м^3}{кг\cdot с^2}$, m_1 – маса першого об’єкта, m_2 – маса другого об’єкта, r – відстань між об’єктами.

У випадку розробки інтерактивної моделі Сонячної системи в Unity 3D закон всесвітнього тяжіння є вирішальним елементом у точному моделюванні руху небесних тіл. Враховуючи дані про масу та відстань до планет, супутників та інших

об’єктів Сонячної системи, можна точно розрахувати сили тяжіння між ними та використовувати їх для моделювання їхніх орбіт і траєкторій.

Реалізація закону всесвітнього тяжіння за допомогою відповідних інструментів і ресурсів рушія Unity 3D можна створити дуже точну інтерактивну модель Сонячної системи.

1.4.4. Спосіб застосування

Щоб використовувати математичні моделі для опису руху астрономічних об’єктів під час створення інтерактивної моделі Сонячної системи в Unity 3D, потрібно реалізувати рівняння руху для кожного небесного тіла з урахуванням сил тяжіння між ними. Найпоширенішими рівняннями руху є рівняння, розроблені І. Ньютоном, які описують рух об’єктів під дією сил гравітації, зокрема закони Ньютона і закон всесвітнього тяжіння.

Також може знадобитися використовувати закони Кеплера про рух планет, щоб обчислити орбітальні параметри планет та інших небесних тіл, такі як їхній ексцентриситет, велика піввісь і період.

Математичні моделі для опису руху астрономічних об’єктів можуть бути реалізовані в сценаріях Unity 3D у вигляді коду, який обчислює та оновлює позиції та рухи небесних об’єктів на основі їхніх фізичних характеристик, таких як маса, швидкість і сили тяжіння. Код можна написати на C#, мові сценаріїв Unity 3D, і інтегрувати в середовище для створення інтерактивної моделі.

1.5. Алгоритми і методи розрахунку параметрів орбіт астрономічних об’єктів

Для даного проєкту алгоритми і методи розрахунку параметрів орбіт астрономічних об’єктів мають важливе значення для створення точної симуляції. Ці розрахунки необхідні для точного визначення положення та руху об’єктів у моделі, а також для забезпечення їх реалістичної поведінки під дією сил гравітації.

Наприклад, обчислення точних параметрів еліптичних орбіт, таких як ексцентриситет і велика піввісь, дозволяє точніше моделювати положення та рух планет. Також за допомогою цих розрахунків можна точно моделювати події, такі як затемнення, з'єднання та транзити.

Існує кілька алгоритмів і методів розрахунку параметрів орбіт астрономічних об'єктів, серед яких є такі [9]:

- Закони Кеплера. Це набір із трьох законів, які описують рух планет навколо Сонця. Вони засновані на ідеї, що планети рухаються по еліптичних орбітах, а Сонце знаходиться в одному з фокусів еліпса. Закони Кеплера можна використовувати для обчислення періоду, великої півосі і ексцентриситету орбіти планети.
- Закони Ньютона. Ці закони описують рух об'єктів під дією сил. Їх можна використовувати для обчислення положення та швидкості астрономічних об'єктів у будь-який момент часу на основі їх початкових умов.
- Метод Гауса. Цей метод заснований на використанні спостережень об'єкта в різні моменти часу та розрахунку орбіти за цими спостереженнями. Суть методу у тому, що використовують три спостереження об'єкта у різні моменти часу, у своїй кожне спостереження визначає положення об'єкта на орбіті (див. рис. 1.8). Використовуючи ці дані, можна розрахувати орбіту об'єкта, використовуючи систему рівнянь Гауса.

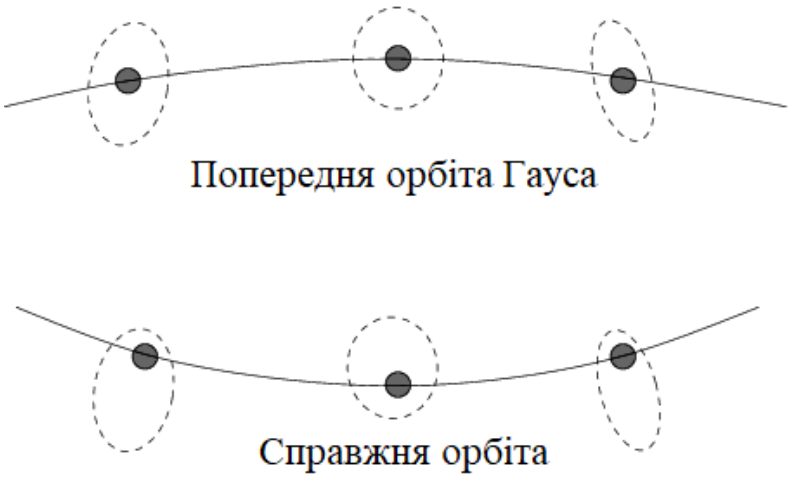

Рис. 1.8. Представлення методу Гауса

- Числове інтегрування. Це метод розв’язування диференціальних рівнянь (які описують рух об’єктів) шляхом апроксимації розв’язку на дискретних кроках у часі. Існує кілька методів чисельного інтегрування, які можна використовувати для розрахунку параметрів орбіт, включаючи метод Ейлера, метод Рунге-Кутта та метод Верле.

Ці алгоритми та методи можна реалізувати в Unity 3D для створення інтерактивної моделі Сонячної системи, яка точно відображає орбіти планет та інших астрономічних об’єктів. Для цього можна використовувати мову програмування C# і вбудовані функції Unity 3D, такі як Mathf.Sin(), Mathf.Cos(), Mathf.Sqrt(), функції об’єкта Quaternion., тип даних Vector3 та багато інших. Основні кроки:

1) Визначити астрономічні об’єкти, їх початкові положення та швидкості.
2) Визначити гравітаційну сталу та маси астрономічних об’єктів.
3) Використати метод розрахунку параметрів орбіт і реалізувати відповідний алгоритм в коді сценарію. Наприклад, якщо використовуються закони Кеплера, то потрібно застосувати відповідні рівняння, які описують положення та швидкість планет, коли вони обертаються навколо Сонця. Це включало б розв’язання рівняння Кеплера, яке пов’язує орбітальну позицію планети з часом з моменту останнього проходження перицентру.
4) Виконуємо оновлення положень астрономічних об’єктів в Unity 3D, використовуючи обчислені положення та обертання.
5) Повторюємо крок 4, щоб постійно оновлювати результати розрахунків.

Варто мати на увазі, що точність і ефективність моделювання залежать від вибору методу, розміру кроку в часі та інших факторів, тому рекомендується провести тестування та оптимізацію для досягнення бажаного рівня точності та продуктивності.

1.6. Елементи інтерактивності

Під час створення інтерактивної моделі Сонячної системи у середовищі Unity 3D існують різні елементи інтерактивності, які можна вивчати та вибирати для покращення взаємодії з користувачем.

Елементи керування камерою. Користувач повинен мати можливість керувати камерою для перегляду різних частин Сонячної системи під різними кутами та з різних відстаней, зокрема окремих об’єктів. Це можна зробити за допомогою клавіатури або миші.

Додавання об’єктів. Користувач за допомогою функції додавання об’єктів може зі списку обрати планету і розмістити у місці, у якому забажає. Це відкриває можливість побудувати власну планетарну систему.

Перегляд планетарних даних. Користувач може натискати на планети чи інші астрономічні об’єкти, щоб переглянути таку інформацію, як назва об’єкта, маса, діаметр, відстань від сонця, ексцентриситет орбіти та інше.

Контроль часу. Користувач може контролювати час, щоб побачити, як планети рухаються по своїх орбітах. Це можна зробити за допомогою повзунка або кнопок прискорення та сповільнення часу.

Взаємодія з об’єктами. Користувач може взаємодіяти з астрономічними об’єктами різними способами, наприклад, змінюючи масу, розмір, швидкість обертання навколо осі, швидкість руху по орбіті, зовсім замінюючи об’єкт на інший та багато чого іншого.

Події. В інтерактивній моделі можуть виконуватися певні події, які викликають інші. Реалізація подій представляє більш складну взаємодію між компонентами моделі. Наприклад, якщо маса планети буде встановлена надто велика, то вона перетворюється в чорну діру, чи якщо її густина буде надто мала, то вона зникає або перетворюється в об’єкт «хмара». І, звісно, якщо користувач натисне на планету, то йому буде представлена панель з даними.

Ці елементи можна поєднувати та налаштовувати для забезпечення інтерактивності в моделі. Звісно, це лише найголовніше, і в процесі створення

інтерактивної моделі можна додати ще більше дрібних елементів. Таким чином буде створена насичена інтерактивна модель Сонячної системи.

1.7. Інтерфейс користувача

1.7.1. Принципи розробки інтерфейсу користувача

Розробка інтерфейсу користувача для інтерактивної моделі Сонячної системи в Unity 3D є важливим аспектом створення успішного проєкту. Інтерфейс користувача має бути розроблений таким чином, щоб забезпечити користувачам легкий доступ до всіх функцій і функцій моделі, а також бути візуально привабливим та інтуїтивно зрозумілим у використанні. Принципи щодо розробки зручного інтерфейсу такі [10-11]:

- Спрощення. Це означає, що потрібно уникати завантаження інтерфейсу великою кількістю кнопок або параметрів. Включати лише найважливіші функції, необхідні для взаємодії з моделлю.
- Мінімізація дій. Потрібно мінімізувати кількість кроків на екрані. Завдання та дії оптимізовані, щоб їх можна було виконати за мінімальну кількість кроків. Інтерфейс має бути розроблений з урахуванням мінімально можливої кількості кроків для виконання будь-яких завдань.
- Використання чітких міток та піктограм. Використовувати мітки та піктограми, які чітко пояснюють функцію кожної кнопки чи опції.
- Надання спливаючих підказок. Надавати спливаючі підказки, які з’являються, коли користувач наводить курсор на кнопку чи опцію, щоб надати додаткову інформацію та вказівки.
- Доступність. Інтерфейс має бути розроблений так, щоб він був доступним для користувачів з обмеженими можливостями, наприклад надаючи альтернативний текст для зображень або використовуючи висококонтрастні кольори.

Загалом, добре розроблений інтерфейс користувача може значно покращити взаємодію з інтерактивною моделлю Сонячної системи в Unity 3D.

1.7.2. Планування інтерфейсу

Розробка інтерфейсу для інтерактивної моделі Сонячної системи в середовищі Unity 3D вимагає ретельного врахування потреб і вподобань користувача. Інтерфейс має бути інтуїтивно зрозумілим і простим у використанні, дозволяючи користувачам взаємодіяти з моделлю та отримувати доступ до інформації про планети та інші небесні тіла Сонячної системи.

Головні елементи, які слід враховувати під час розробки інтерфейсу для моделі Сонячної системи:

- Елементи керування навігацією. Користувач повинен мати можливість переміщатися по моделі Сонячної системи за допомогою таких елементів керування, як масштабування, обертання, перехід до об’єкта. Ці елементи керування мають бути простими у використанні та інтуїтивно зрозумілими.
- Інформаційні дисплеї. Користувач повинен мати доступ до інформації про планети та інші об’єкти Сонячної системи. В ній можуть міститися різні дані про об’єкт, починаючи від назви, закінчуючи довжиною великої півосі орбіти.
- Елементи керування часом. Користувач повинен мати можливість керувати протіканням часу в моделі Сонячної системи, дозволяючи їм бачити, як планети та інші небесні тіла рухаються та взаємодіють з часом.
- Параметри налаштування. Користувач повинен мати можливість налаштовувати модель Сонячної системи відповідно до своїх уподобань, наприклад, змінювати розмір або зовнішній вигляд (текстуру) планети, додавати або видаляти об’єкти та регулювати масштаб моделі.
- Зворотній зв’язок з користувачем. Згідно з принципом про надання спливаючих підказок, інтерфейс має забезпечувати зворотний зв’язок з користувачем, наприклад, відображення повідомлень в окремому вікні, коли

виконується певна дія, або надання візуальних підказок для вказівки на зміни в моделі.

Загалом інтерфейс для інтерактивної моделі Сонячної системи у середовищі Unity 3D має бути зручним, інтуїтивно зрозумілим та інформативним. Це має дозволити користувачам досліджувати Сонячну систему та дізнаватися про неї в цікавій та інтерактивній формі.

1.8. Висновки до розділу 1

В розділі 1 було розглянуто можливості середовища Unity 3D для створення графічної моделі Сонячної системи, ознайомлення з її обмеженнями та розробка концепції інтерактивної моделі Сонячної системи. Розробка концепції полягала в вивченні математичних моделей опису руху астрономічних об’єктів та алгоритмів розрахунку параметрів орбіт астрономічних об’єктів, а також було досліджено можливості використання інтерактивних елементів та обрано найбільш підходящі елементи для створення інтерактивної моделі Сонячної системи. Крім того, були вивчені принципи проєктування інтерфейсу користувача та розроблено дизайн інтерфейсу для інтерактивної моделі Сонячної системи.

Виконання цих завдань дозволило зрозуміти роботу інтерактивної моделі Сонячної системи у середовищі Unity 3D. Тепер відомо, які роботи потрібно виконати, щоб досягти результату. Ця модель буде мати як навчальне, так і розважальне значення. І результати цієї роботи демонструють, що середовище Unity 3D можна використовувати для створення візуально привабливих та інтерактивних моделей складних систем, таких як Сонячна система, і забезпечувати привабливий досвід для користувачів.

Метою роботи визначено створення функціональної та інформативної інтерактивної моделі Сонячної системи, яку можна використовувати в освітніх та розважальних цілях. Для цього потрібно буде в рушії Unity 3D створити тривимірні моделі Сонця, планет, супутників, карликових планет та деяких інших об’єктів і задати їм відповідні характеристики. Також потрібно використати текстури,

освітлення та інші ефекти, щоб зробити модель більш близькою до реальної Сонячної системи. У сценаріях використаємо математичні моделі та алгоритми для розрахунку руху, орбіт і положення об’єктів. Використаємо інтерактивні елементи, такі як елементи керування камерою, перегляд планетарних даних, керування часом тощо, щоб надати користувачу можливість взаємодіяти з моделлю. Згідно з принципами, створимо інтуїтивно зрозумілий і простий у використанні інтерфейс користувача.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Основні вимоги

Основні функції та можливості інтерактивної моделі Сонячної системи у середовищі Unity 3D повинні включати:

1) Візуалізація моделі. Модель має забезпечувати реалістичну візуалізацію планет, супутників та інших об’єктів Сонячної системи з деталізацією та текстурами. Зокрема мають бути відображені такі об’єкти, як Сонце, 8 планет, карликові планети та супутники.
2) Анімація руху. Повинен виконуватись рух астрономічних об’єктів згідно з реальними законами фізики. Наприклад, є закон всесвітнього тяжіння, і сила, яка обчислюється, для кожного об’єкту окремо, штовхає його у певному напрямку.
3) Зміна швидкості часу і його зупинка. Користувач повинен мати можливість збільшувати швидкість плину часу, щоб не очікувати результат моделювання надто довго, або щоб спостерігати короткочасні події у сповільненому часі, та навіть зупиняти симуляцію у будь-який момент.
4) Управління камерою. Користувачу повинна бути надана можливість змінювати масштаб відображення моделі, наближаючись або віддаляючись для біль детального або загального огляду об’єктів Сонячної системи. Користувач також повинен мати можливість обертати камеру навколо обраного об’єкта, щоб роздивитись його з різних ракурсів, і перемикатися між об’єктами, щоб зосередитись на конкретному об’єкті та дослідити його.

5) Інтерактивна взаємодія з моделлю. Можливість зміни параметрів об’єктів, таких як радіус, маса, швидкість обертання тощо. Можливість додавати нові астрономічні об’єкти в модель, такі як планети, супутники тощо, а також їх видаляти.
6) Обчислення орбітальних параметрів. За допомогою певних алгоритмів має обчислюватись орбітальний родич об’єкта (навколо якого обертається), велика піввісь, ексцентриситет, нахил орбіти, довгота висхідного вузла, аргумент перицентру, істинна та середня аномалія.
7) Інтерфейс користувача. Він повинен мати інтуїтивну навігацію, мати кнопки для управління об’єктами, введення та виведення параметрів об’єктів, додавання та видалення об’єктів.

Знаючи ці основні вимоги, можна спланувати подальшу роботу.

2.2. Варіанти використання системи

Розробимо діаграму використання, яка представить взаємодію одного користувача з підсистемою «Інтерактивна модель Сонячної системи у середовищі Unity 3D». Вона покаже різні варіанти використання, які користувач може виконувати в цій системі (рис. 2.1). Кожен варіант використання описує конкретну функцію або можливість, яку користувач може використовувати для взаємодії з моделлю Сонячної системи. Крім того, діаграма використання показує зв’язки між різними варіантами використання, такі як розширення (<<extend>>) і включення (<<include>>), що вказують на залежності між функціями та можливостями системи.

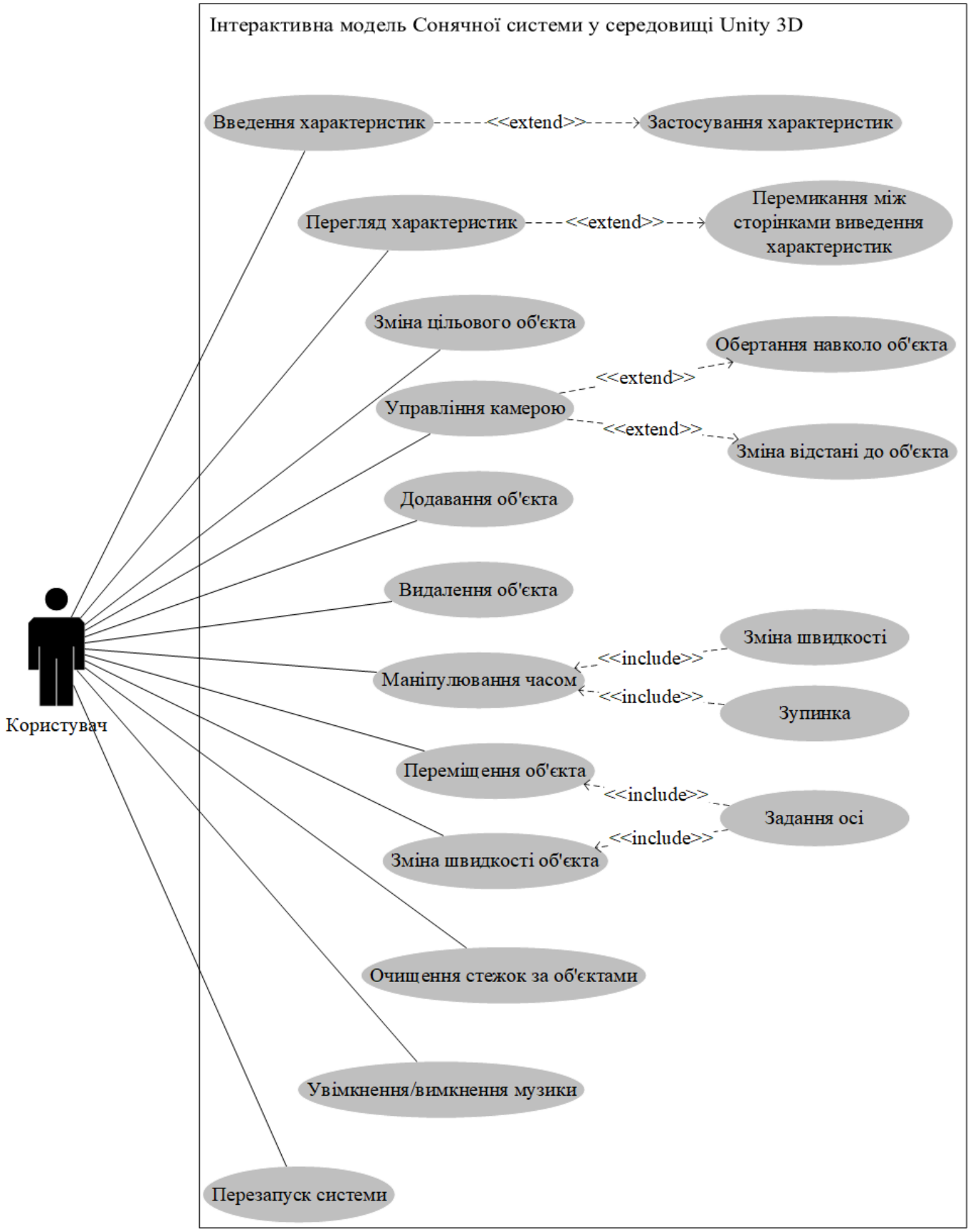

Рис. 2.1. Діаграма варіантів використання

Варіанти використання:

1) «Введення характеристик». Користувач може вводити характеристики об’єктів (наприклад, масу, радіуси, період обертання тощо), що впливають на їхню поведінку у моделі. Цей варіант використання може розширюватися варіантом «Застосування характеристик», коли введені характеристики застосовуються до об’єктів.
2) «Перегляд характеристик». Користувач може переглядати характеристики об’єктів, такі як масу, радіуси, період обертання тощо. Цей варіант використання може розширюватися варіантом «Перемикання між сторінками виведення характеристик», коли користувач може перемикатися між різними сторінками для перегляду різних груп характеристик.
3) «Зміна цільового об’єкта». Користувач може змінювати цільовий об’єкт, на який спрямовані дії у моделі. Наприклад, він може вибрати іншу планету, зорю або місяць для подальшої взаємодії.
4) «Управління камерою». Користувач може управляти камерою, яка відображає модель. Цей варіант використання може розширюватися варіантами «Обертання навколо об’єкта» (користувач може обертати камеру навколо цільового об’єкта), «Зміна відстані до об’єкта» (користувач може збільшувати або зменшувати відстань між камерою та цільовим об’єктом).
5) «Додавання об’єкта». Користувач може додавати нові об’єкти (наприклад, планети, місяці, зорі) до моделі Сонячної системи.
6) «Видалення об’єкта». Користувач може видаляти цільовий об’єкт з моделі Сонячної системи.
7) «Маніпулювання часом». Користувач може контролювати час у моделі. Цей варіант використання включає в себе варіанти «Зміна швидкості» (користувач може змінювати швидкість програвання часу) і «Зупинка» (користувач може зупиняти рух об’єктів).
8) «Переміщення об’єкта». Користувач може переміщати об’єкт у просторі. Цей варіант використання включає в себе варіант «Задання осі» (користувач може вказувати вісь для переміщення об’єкта).

9) «Зміна швидкості об’єкта». Користувач може змінювати швидкість руху об’єкта у просторі. Цей варіант використання включає в себе варіант «Задання осі» (користувач може вказувати вісь для зміни швидкості об’єкта).
10) «Очищення стежок за об’єктами». Користувач може очищати стежки, які залишили об’єкти у моделі. Стежки відображають пройдену траєкторію руху об’єктів.
11) «Увімкнення/вимкнення музики». Користувач може увімкнути або вимкнути музику в моделі.
12) «Перезапуск системи». Користувач може перезапустити модель Сонячної системи, починаючи зі стандартних налаштувань і об’єктів.

2.3. Основні класи

Класи використовуються для опису об’єктів та функціональності. Вони відіграють важливу роль у розробці програм, зокрема у середовищі Unity 3D, надаючи структуру та поведінку для різних елементів програми, таких як об’єкти, сцени, анімації тощо.

Далі наведено поширені класи Unity 3D, які часто будуть використовуватися у даному проєкті [12].

MonoBehaviour. Цей клас є основним класом для створення компонентів, які можна додати до ігрових об’єктів. Він надає функції для управління життєвим циклом об’єкта, включаючи методи Awake(), Start(), Update() і FixedUpdate(), а також безліч інших функцій для обробки подій та виконання різних дій.

GameObject. Клас GameObject представляє ігровий об’єкт у сцені. Він містить методи та властивості для керування положенням, поворотом, масштабуванням, активацією/деактивацією та іншими атрибутами об’єкта.

Transform. Клас Transform відповідає за перетворення та положення ігрового об’єкта у сцені. Він дозволяє отримувати та встановлювати позицію, поворот та масштаб об’єкта, а також виконувати операції над ними, такі як переміщення, обертання та масштабування.

Rigidbody. Клас Rigidbody є фізичним тілом об’єкта. Він дозволяє об’єктам взаємодіяти з фізичним рушієм Unity 3D, імітуючи гравітацію, сили, зіткнення та інші фізичні ефекти.

У даному проєкті будуть створені власні класи, які будуть мати своє призначення (рис. 2.2, 2.3). Цими класами є:

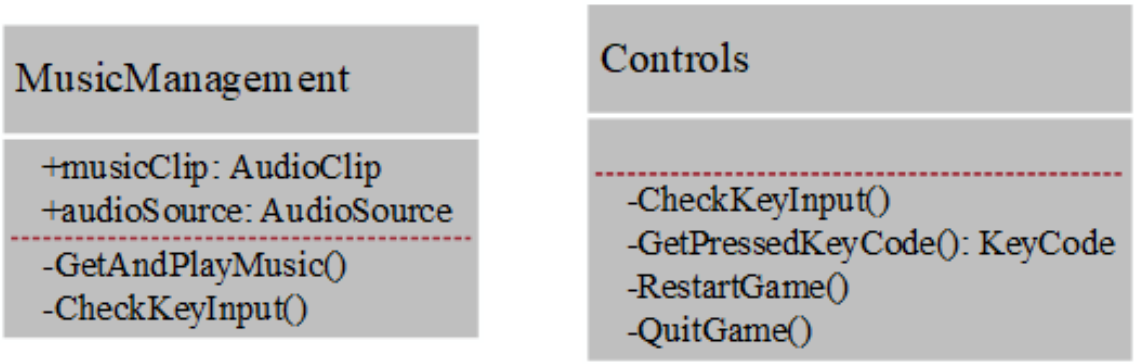

Рис. 2.2. Діаграма класів

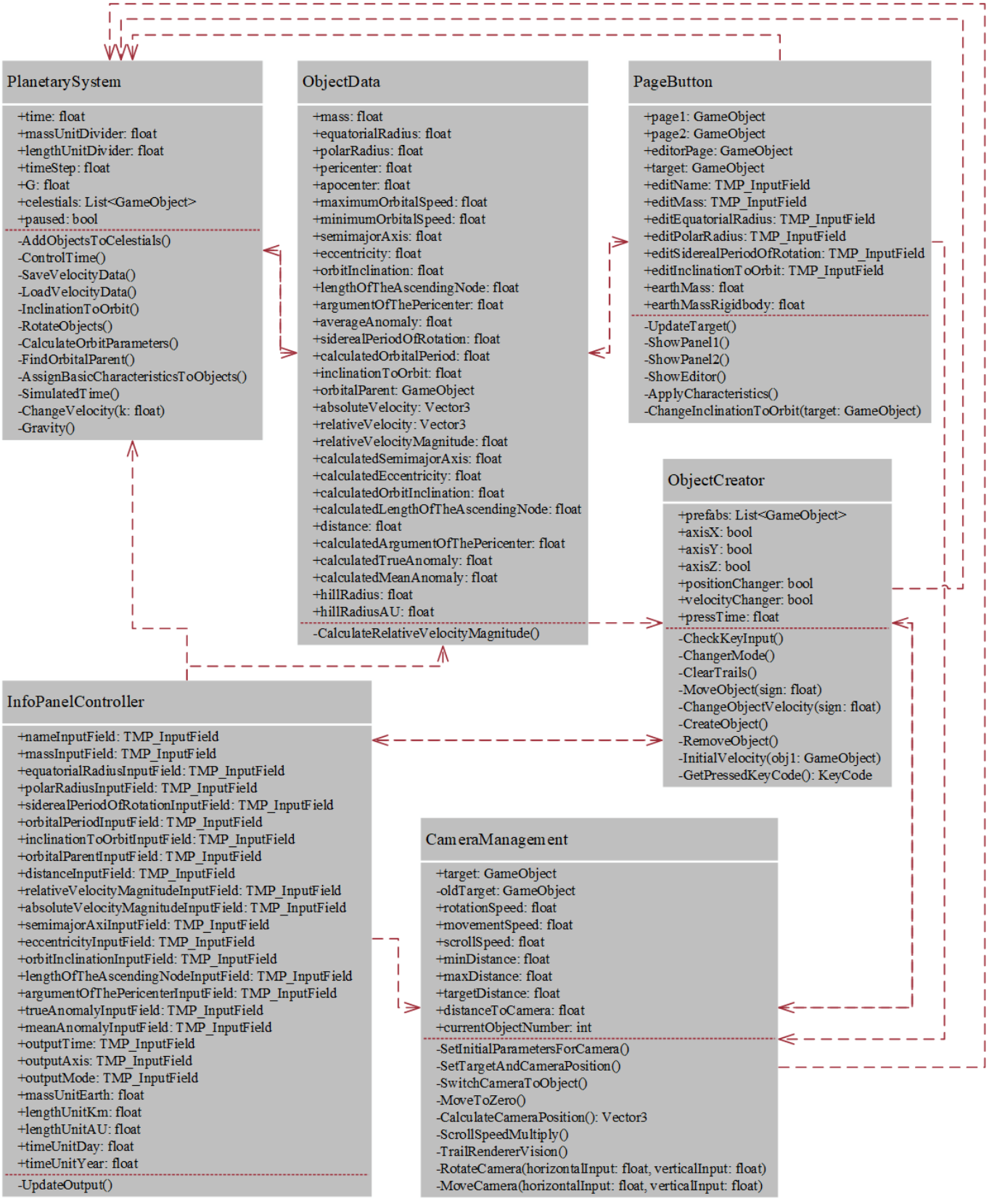

Рис. 2.3. Діаграма класів

1) PlanetarySystem – відповідає за керування фізичною взаємодією між об’єктами на сцені та обчислення;

2) ObjectData – відповідає за збереження інформації про астрономічний об’єкт, такої як маса, радіуси, перицентри і апоцентри, обчислювальні параметри та багато іншої у зручних одиницях вимірювання;
3) PageButton – відповідає за відображення панелей з полями і кнопками, можливість зміни деяких параметрів астрономічного об’єкта;
4) ObjectCreator – відповідає за створення і видалення об’єктів, їх переміщення і зміну швидкості;
5) InfoPanelController – відповідає лише за переведення результатів обчислення у зручні одиниці вимірювання і подальше їх виведення у поля на відповідній панелі;
6) CameraManagement – відповідає за управління камерою, фіксування цільового об’єкта, переміщення об’єктів до нульових координат, видимість слідів об’єктів;
7) MusicManagement – відповідає за відтворення музики;
8) Controls – відповідає за перезапуск сцени і завершення роботи програми.

2.4. Характеристики основних астрономічних об’єктів

Перед тим, як почати створювати об’єкти в середовищі Unity 3D, необхідно дізнатися, які параметри вони повинні мати, та задати значення цим параметрам. У таблицях В.1 і В.2 наведено зібрані дані. Курсивом виділені астрономічні об’єкти, які обертаються навколо орбітального родича, що не є Сонцем, тобто це супутники.

Отже, основними параметрами, які повинен мати астрономічний об’єкт в контексті програмування, будуть ті, що вказані у першому стовпчику таблиці 1. Але спочатку достатньо буде маси, екваторіального і полярного радіусу, відстані (перицентру і апоцентру), швидкості (максимальної і мінімальної) і сидеричного періоду обертання [13-14]. Інші параметри можна обчислити і порівняти з тими, що є в таблицях А.1 і А.2 Додатку А.

Оскільки середовище Unity 3D має обмеження на величину чисел і координат, значення доведеться переводити в інші одиниці вимірювання. Хоча значення в

системі СІ можна записати в екземпляри класу астрономічних об’єктів, у такому разі в алгоритми для обчислення параметрів або представлення положення та розміру об’єкта доведеться додавати множники. Наприклад, якщо потрібно розмістити об’єкт на відстані 100 астрономічних одиниць (а.о.), а Unity 3D має обмеження на величину координати в 100 000 одиниць, то потрібно визначити, скільки координатних одиниць Unity 3D буде в 1 а.о. Нехай розмістимо об’єкт на координаті X = 100 000, тоді 1 а.о. дорівнює 1000 одиниць координат в Unity 3D, і також ми не зможемо розмістити об’єкт на відстані, припустимо, 101 а.о. Але також варто мати на увазі, що чим далі об’єкт від початку координат, тим менш деталізованим він стає. Це пов’язано з обмеженням точності чисел з плаваючою крапкою в Unity 3D. Можна зменшити масштаб моделі, щоб вмістились об’єкти на більшій відстані (в а.о.), але малі об’єкти буде проблематично роздивитись навіть у середовищі Unity 3D, і це не вирішить проблему зі зниженням деталізації відображення при збільшенні координат.

2.5. Формули для обчислення орбітальних параметрів

Вектори орбітального стану, також відомі як кеплерові елементи орбіти, – це набір параметрів, які описують орбіту небесного об’єкта, наприклад, планети, супутника або космічного корабля навколо центрального тіла. Ці елементи надають інформацію про форму, розмір, орієнтацію та положення орбіти. Орбітальні вектори стану зазвичай включають такі елементи [15]:

- Велика піввісь (a) – вона визначає середній розмір орбіти та є половиною найдовшого діаметра еліптичної орбіти.
- Ексцентриситет (e) – він визначає форму орбіти та вказує на те, наскільки витягнутим є еліпс. Значення 0 відповідає круговій орбіті, тоді як значення між 0 і 1 представляють еліптичні орбіти.
- Нахил (i) – визначає нахил орбіти відносно площини відліку, якою зазвичай є площина екватора центрального тіла.

– Довгота висхідного вузла (Ω) – визначає кут між опорним напрямком (наприклад, точкою весняного рівнодення) та лінією, де орбіта перетинає базову площину, відомою як висхідний вузол.
– Аргумент перицентру (ω) – визначає кут між висхідним вузлом і точкою, де орбіта наближається найближче до центрального тіла (перицентру).
– Істинна аномалія (f) та середня аномалія (M) – ці елементи описують положення об'єкта вздовж його орбіти в певний момент часу. Істинна аномалія визначає фактичний кут між перицентром і поточним положенням, тоді як середня аномалія – це частка орбітального періоду, який минув з моменту проходження об'єкта через перицентр.

Знаючи ці орбітальні вектори стану, можна точно визначити та розрахувати положення та рух небесних об'єктів у межах їхніх орбіт.

Велика піввісь (a) для еліптичної орбіти може бути розрахована з векторів стану орбіти (кеплерових елементів орбіти) за наступною формулою:

$$a = -\frac{\mu}{2\varepsilon}, \tag{2.1}$$

де μ – стандартний гравітаційний параметр; ε – питома енергія тіла, що обертається, які у свою чергу обчислюються за наступними формулами відповідно:

$$\mu = G(m_1 + m_2), \tag{2.2}$$

$$\varepsilon = \frac{v^2}{2} - \frac{\mu}{|\vec{r}|}, \tag{2.3}$$

де G – гравітаційна стала ($G = 6{,}67430(15) \times 10^{-11} \frac{\text{м}^3}{\text{кг}\cdot\text{с}^2}$); m_1 – маса тіла, що обертається; m_2 – маса тіла, навколо якого обертається тіло; v – орбітальна швидкість від вектору швидкості орбітального об'єкта (відносна швидкість); $\vec{r}$ – декартовий вектор положення об'єкта на орбіті в координатах системи відліку,

відносно якої мають бути розраховані елементи орбіти (наприклад, геоцентричний екваторіальний для орбіти навколо Землі або геліоцентричний екліптичний для орбіти навколо Сонця) (відстань до орбітального родича). Відносна швидкість обчислюється наступним чином:

$$v = |\vec{v}|, \tag{2.4}$$

де $\vec{v}$ – вектор швидкості орбітального об’єкта, який обчислюється наступним чином:

$$\vec{v} = \overrightarrow{v_o} - \overrightarrow{v_c}, \tag{2.5}$$

де $\overrightarrow{v_o}$ – вектор швидкості об’єкта, що обертається; $\overrightarrow{v_c}$ – вектор швидкості центрального об’єкта.

Декартовий вектор положення об’єкта на орбіті обчислюється аналогічним чином:

$$\vec{r} = \overrightarrow{r_o} - \overrightarrow{r_c}, \tag{2.6}$$

де $\overrightarrow{r_o}$ – вектор положення об’єкта, що обертається; $\overrightarrow{r_c}$ – вектор положення центрального об’єкта.

Ексцентриситет орбіти (e) можна розрахувати за векторами стану орбіти як величину вектору ексцентриситету:

$$e = |\vec{e}|, \tag{2.7}$$

де $\vec{e}$ – вектор ексцентриситету, який у свою чергу розраховується за формулою:

$$\vec{e} = \frac{\vec{v} \times \vec{h}}{\mu} - \frac{\vec{r}}{|\vec{r}|}, \tag{2.8}$$

де $\vec{h}$ – вектор питомого кутового моменту:

$$\vec{h} = \vec{r} \times \vec{v}. \tag{2.9}$$

Нахил орбіти (i) можна обчислити за вектором орбітального моменту $\vec{h}$ (або будь-яким вектором, перпендикулярним до площини орбіти):

$$i = arccos\frac{h_y}{|\vec{h}|}, \tag{2.10}$$

де h_y – y-компонента (висота) вектору орбітального моменту $\vec{h}$. Орієнтація осей зображена на рис. 2.4.

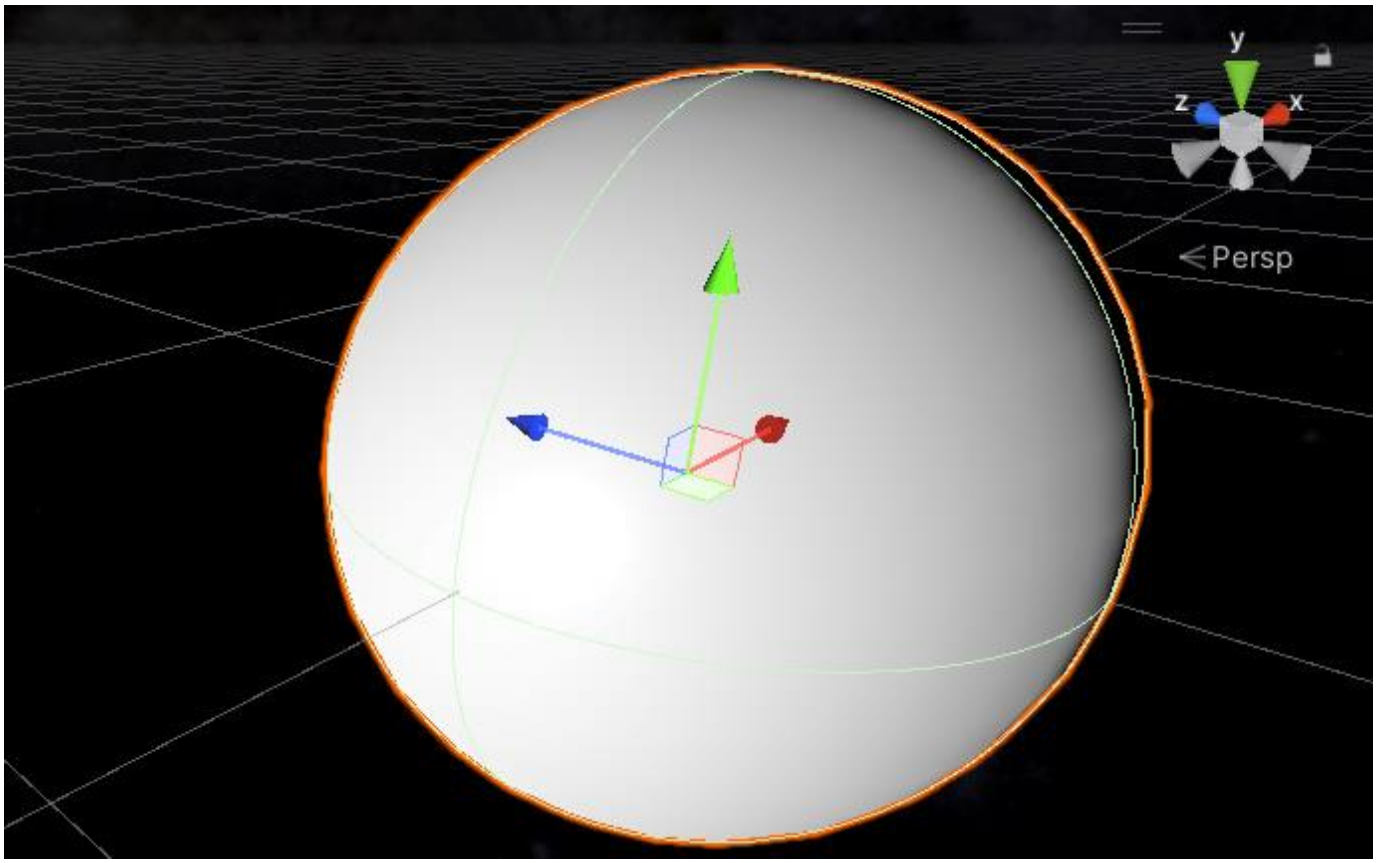

Рис. 2.4. Розміщення і найменування координатних осей у середовищі Unity 3D

Довгота висхідного вузла (Ω) може бути обчислена за конкретним вектором відносного кутового моменту $\vec{h}$ наступним чином. Спочатку обчислюється вектор, що вказує на висхідний вузол:

$$\vec{n} = \vec{k} \times \vec{h} = (h_z, 0, -h_x), \tag{2.11}$$

де $\vec{k}$ – одиничний вектор $(0,1,0)$, який є вектором нормалі до базової площини Oxz. А початок довготи приймається за додатну вісь Ox. І тепер довгота висхідного вузла розраховується так:

$$\Omega = \begin{cases} arccos\dfrac{n_x}{|\vec{n}|}, & n_z \geq 0; \\ 2\pi - arccos\dfrac{n_x}{|\vec{n}|}, & n_z < 0. \end{cases} \tag{2.12}$$

Для орбіт без нахилу (нахилом, що дорівнює нулю) довгота висхідного вузла не визначена. Тоді для обчислень вона за замовчуванням встановлюється рівною нулю, тобто висхідний вузол розміщується в опорному напрямку, що еквівалентний тому, що $\vec{n}$ вказує на додатну вісь Ox.

Аргумент перицентру (ω) можна розрахувати наступним чином:

$$\omega = \arccos\frac{\vec{n} \cdot \vec{e}}{|\vec{n}||\vec{e}|}, \tag{2.13}$$

Якщо $e_z < 0$, то $\omega \to 2\pi - \omega$. А у випадку орбіт, у яких немає висхідного вузла, аргумент перицентру строго не визначений.

Середня аномалія (M) може бути обчислена за допомогою рівняння Кеплера:

$$M = E - e\,sin\,E\,, \tag{2.14}$$

де E – ексцентрична аномалія – параметр, що використовується для вираження змінної довжини r радіус-вектору $\vec{r}$. Її можна обчислити за наступною формулою:

$$E = arccos\left(\frac{1}{e} - \frac{|\vec{r}|}{a \cdot e}\right). \quad (2.15)$$

Істинна аномалія (f) може бути обчислена за векторами стану орбіти як:

$$f = arccos\frac{\vec{e} \cdot \vec{r}}{|\vec{e}||\vec{r}|}. \quad (2.16)$$

Якщо $r \cdot f < 0$, то замінити f на $2\pi - f$.

Щоб визначити, навколо якого об'єкта обертається об'єкт, потрібно визначити, у сфері Хілла якого об'єкта він знаходиться. Якщо тіло з меншою масою обертається навколо більш важкого тіла з великою піввіссю та ексцентриситетом, то радіус сфери Хілла меншого тіла, розрахований в перицентрі, складає приблизно:

$$r_H \approx a(1-e) \cdot \sqrt[3]{\frac{m}{3M}}. \quad (2.17)$$

2.6. Висновки до розділу 2

У розділі 2 було визначено функціональні вимоги до системи, які описують її можливості та властивості. Функціональні вимоги включають операції, які система повинна забезпечувати, такі як додавання, видалення та переміщення астрономічних об'єктів, управління камерою тощо.

Також було побудовано діаграму варіантів використання, яка ілюструє взаємодію користувача з системою та її функціональні можливості. Варіанти використання включають введення та перегляд характеристик об'єктів, зміну

цільового об’єкта, управління камерою, додавання та видалення об’єктів, маніпулювання часом та інші операції.

Сформовано діаграми класів, у яких представлені змінні та функції, якими володіють класи у програмі.

Описано характеристики, які можуть бути задані для астрономічних об’єктів у системі. Ці характеристики включають масу, радіуси, періоди обертання, орбітальні параметри об’єктів та багато інших

Наведено формули, які дозволяють обчислити орбітальні параметри астрономічних об’єктів. Ці формули використовуються для розрахунків, таких як визначення величини піввісі, ексцентриситету, нахилу орбіти, аргументу перицентра тощо.

Отже, визначено основні вимоги до системи, описано варіанти використання, класи, характеристики астрономічних об’єктів та формули для обчислення орбітальних параметрів. Ці елементи допомагають легше приступити до реалізації системи, яка буде чітко задовольняти потреби користувача та забезпечувати потрібні функціональності.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ У СЕРЕДОВИЩІ UNITY 3D

3.1. Реалізація базової Сонячної системи

Щоб додати астрономічні об’єкти на сцену в Unity 3D і створити реалістичну і, поки що, статичну модель Сонячної системи (рис. 3.1), потрібно виконати наступні кроки:

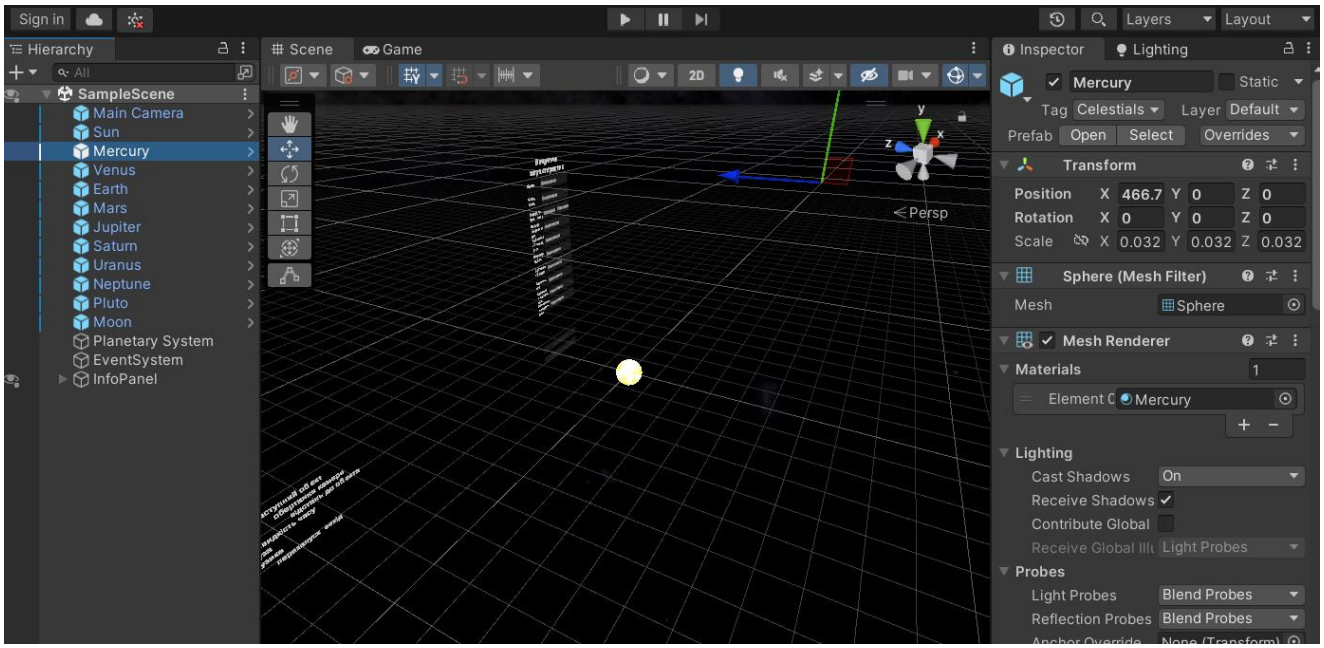

Рис. 3.1. Базова Сонячна система

1) Створити астрономічні об’єкти. Створити астрономічні об’єкти, такі як планети, зірки, комети або інші космічні тіла. Це можна зробити шляхом створення 3D моделей або використання готових моделей, текстур і матеріалів.

2) Створити сцену. В Unity створити нову сцену або відкрити існуючу сцену, де потрібно розмістити астрономічні об’єкти.
3) Додати астрономічні об’єкти на сцену:
    - додати 3D моделі астрономічних об’єктів на сцену, розмістивши їх у відповідних позиціях та масштабах;
    - застосувати текстури (рис. 3.2) на моделі, щоб вони виглядали реалістично, і ці текстури мають відповідають зовнішньому вигляду астрономічних об’єктів.

Рис. 3.2. Текстурування

- Освітлення. Налаштувати освітлення сцени, щоб забезпечити правильну інтерпретацію світла на астрономічних об’єктах.
- Додавання фону. За допомогою використання Skybox можна додати фон на сцені. Skybox – це спосіб відображення фону за допомогою текстури, яка охоплює всю сцену.

Реалізація відбувається у створеному класі PlanetarySystem (Додаток Б), у якому виконуються основні обчислення при відтворенні моделі Сонячної системи. Таким чином, цей клас можна вважати основним.

Спочатку у коді оголошуємо статичні змінні та метод Awake(). Статичні змінні (time, massUnitDivider, lengthUnitDivider, timeStep, G, celestials, paused) мають загальне значення для всіх екземплярів класу та доступні з інших методів або класів без потреби створення екземпляра об’єкта.

Значення цих змінних такі:

- *time* – значення часу, спочатку встановлене на 0;
- *massUnitDivider* – коефіцієнт, який використовується для ділення значень маси об’єктів;
- *lengthUnitDivider* – коефіцієнт, який використовується для ділення значень довжини або відстані;
- *timeStep* – крок часу, який використовується в фізичному розрахунку, обчислений як 2629800f (секунд у місяці) поділене на 30 (ділено на кількість днів);
- *G* – гравітаційна стала, обчислена з використанням відомих фізичних констант, коефіцієнтів маси та довжини;
- *celestials* – список ігрових об’єктів, які мають тег “Celestials” або “Central Object”, у методі Awake() відбувається пошук та додавання цих об’єктів до списку;
- *paused* – булеве значення, що вказує, чи гра перебуває у призупиненому стані чи ні.

Метод Awake є одним із методів, які визначають життєвий цикл компонента в Unity. Він є частиною процесу ініціалізації об’єкта перед його використанням у сцені гри. Метод Awake викликається автоматично під час завантаження сцени або активації об’єкта, до початку виконання інших методів.

Основна роль методу Awake полягає в підготовці об’єкта до виконання різних операцій, таких як налаштування початкових значень, ініціалізація змінних, пошук та збереження посилань на інші об’єкти або компоненти, налаштування розмірів та інших характеристик об’єкта.

Основна важливість методу Awake полягає в тому, що він надає можливість виконати необхідні налаштування до початку роботи з об’єктом. Це може включати налаштування початкових значень, ініціалізацію змінних, пошук та збереження посилань на інші об’єкти або компоненти, а також підготовку об’єкта для роботи з фізикою, анімацією, звуком, мережевими функціями та іншими аспектами гри.

Загалом, метод Awake є важливим кроком при розробці ігрових об’єктів і дозволяє виконати підготовчі дії, необхідні для коректної роботи об’єкта під час виконання програми.

У нашому випадку в Awake() викликаються наступні методи:

1) AddObjectsToCelestials();
2) AssignBasicCharacteristicsToObjects();
3) CalculateOrbitParameters();
4) InclinationToOrbit().

У методі AddObjectsToCelestials() створюється новий екземпляр списку celestials, який буде містити ігрові об’єкти, а потім відбувається пошук усіх ігрових об’єктів з тегом “Celestials” та додавання його до списку celestials, і аналогічно з ігровим об’єктом з тегом “Central Object”.

У методі AssignBasicCharacteristicsToObjects() застосовуються вказані в компоненті ObjectData базові характеристики для об’єктів (які знаходяться у списку celestials), деякі з яких були вказані у таблиці В.1 і В.2. Для кожного об’єкта в списку виконуються певні дії:

1) Обчислення маси та налаштування маси фізичного компонента (Rigidbody):
   - значення маси отримують з компонента ObjectData об’єкта (рис. 3.3);

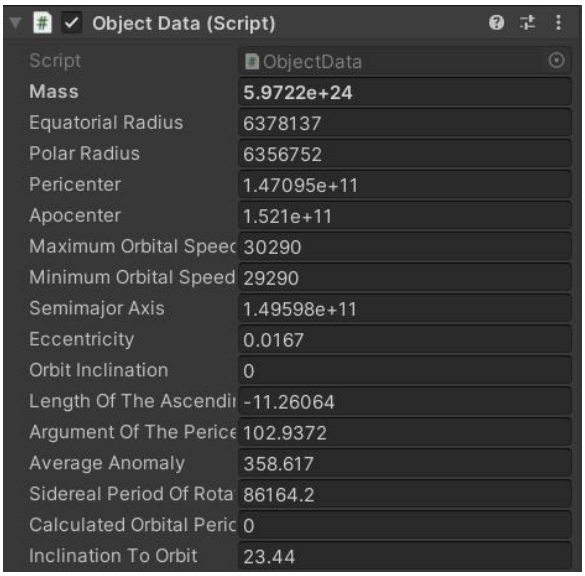

Рис. 3.3. Джерело характеристик для призначення на прикладі даних про планету Земля

- обчислюється нове значення маси, розділене на massUnitDivider (для перетворення маси від одиниць у системі Unity до реальних фізичних одиниць);
- призначається нове значення маси фізичному компоненту об’єкта;

2) Налаштування масштабу об’єкта (transform.localScale):
   - значення радіусів (екваторіального та полярного) отримують з компонента ObjectData об’єкта;
   - обчислюються нові значення масштабу, які враховують радіуси і lengthUnitDivider (для перетворення радіусів від одиниць у системі Unity до реальних фізичних одиниць);
   - призначається новий масштаб об’єкта;
3) Позиціонування та надання початкової швидкості об’єктам:
   - перевіряється, чи об’єкт має orbitalParent (батьківський об’єкт у орбітальній системі);
   - якщо об’єкт не має orbitalParent і не є центральним об’єктом:
     - встановлюється позиція об’єкта на основі значення pericenter та lengthUnitDivider;
     - встановлюється швидкість об’єкта, де значення maximumOrbitalSpeed і timeStep враховуються та перетворюються на реальні фізичні одиниці;
   - якщо об’єкт має orbitalParent і не є центральним об’єктом:
     - встановлюється позиція об’єкта, яка враховує pericenter, lengthUnitDivider та позицію orbitalParent;
     - встановлюється швидкість об’єкта, яка враховує maximumOrbitalSpeed, timeStep та швидкість orbitalParent.

Отже, цей метод встановлює базові характеристики для об’єктів на основі даних, які зберігаються в ObjectData та інших компонентах, та позиціонує їх у просторі згідно з визначеними параметрами орбіт.

У методі CalculateOrbitParameters() обчислюються орбітальні параметри за формулами, які були наведені у розділі 2, а також орбітальний період за наступною формулою:

$$T = 2\pi \cdot \sqrt{\frac{a^3}{\mu}}. \tag{3.1}$$

У цьому методі обчислюються параметри орбіти для об’єктів (рис. 3.4).

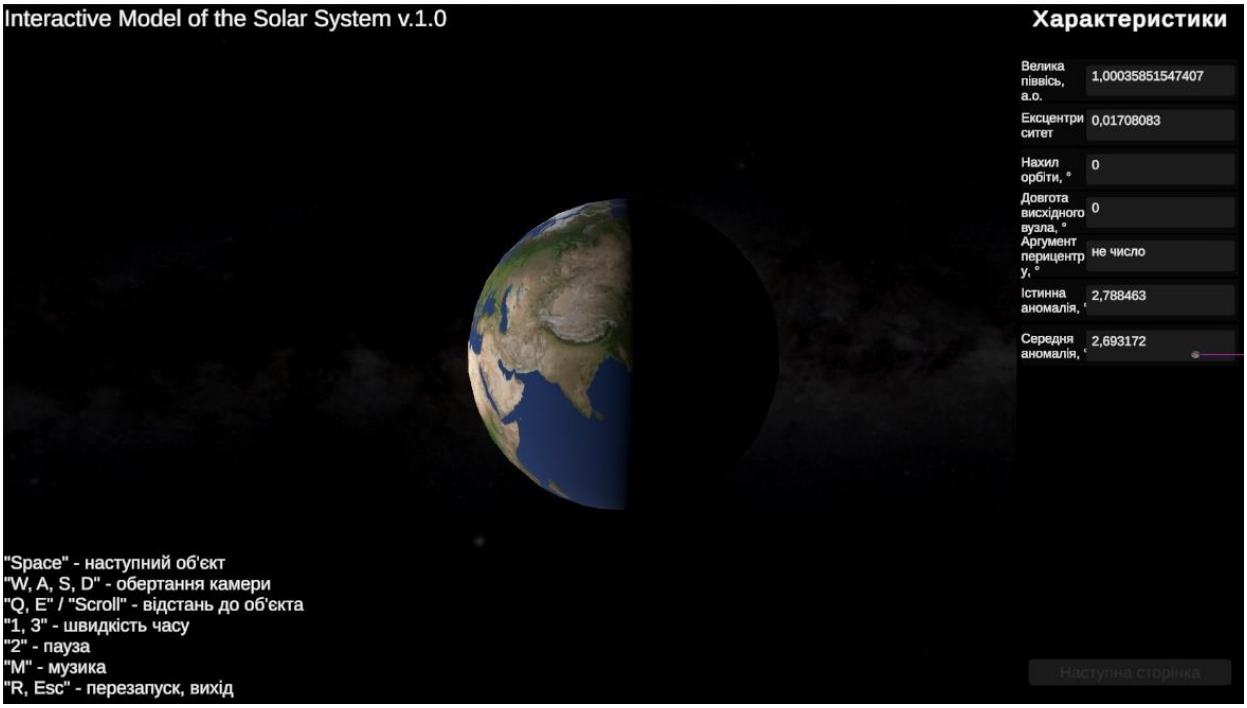

Рис. 3.4. Представлення обчислень основних орбітальних параметрів при орбіті з нульовим нахилом

Зокрема, обчислюються такі параметри:

- велика піввісь орбіти (calculatedSemimajorAxis);
- ексцентриситет орбіти (calculatedEccentricity);
- нахил орбіти (calculatedOrbitInclination);
- довгота висхідного вузла (calculatedLengthOfTheAscendingNode);
- аргумент перицентру (calculatedArgumentOfThePericenter);

- істинна аномалія (calculatedTrueAnomaly);
- середня аномалія (calculatedMeanAnomaly);
- орбітальний період (calculatedOrbitalPeriod).

Ці параметри обчислюються на основі властивостей об’єктів, таких як маса, швидкість, положення і відстань від батьківського об’єкта. Обчислені значення зберігаються в компонентах ObjectData об’єктів.

І, на завершення, у методі Awake() виконується метод InclinationToOrbit(). Він виконує нахил об’єктів у системі.

Для кожного об’єкта *a* отримується значення нахилу орбіти (inclination) з компонента ObjectData.

Створюється нова кватерніонна ротація newRotation, яка представляє нахил орбіти. Значення -inclination використовується для обертання об’єкта а навколо осі X з метою нахилу орбіти. Решта значень у кватерніоні (0f) залишаються незмінними.

Кватерніон (Quaternion) в Unity 3D – це тип даних, який використовується для представлення обертання або орієнтації об’єктів у тривимірному просторі. Він складається з чотирьох чисел: (x, y, z, w), де (x, y, z) – вектор у тривимірному просторі, а w – скаляр.

Нова ротація newRotation застосовується до локальної ротації об’єкта *a*, що змінює його орієнтацію. Це призводить до нахилу орбіти об’єкта *a* відносно початкового положення.

3.2. Впровадження динамічності у систему

Для впровадження динамічності у модель Сонячної системи можна використати наступні підходи:

- Анімація. Використання анімаційних ефектів для руху та зміни положення об’єктів у системі. Це може включати обертання планет навколо своїх осей, рух супутників навколо планет, та інші рухові ефекти, що додають реалізм до моделі.

- Фізичне моделювання. Використання фізичного рушія, як от Unity 3D, для моделювання гравітаційної взаємодії між об’єктами у системі. Це дозволить реалістично відтворювати рух об’єктів та їх взаємодію.
- Взаємодія користувача. Надання можливості користувачу взаємодіяти з об’єктами системи, наприклад, змінювати їх положення, швидкість руху або масштаб. Дозвіл користувачу контролювати відображення певних аспектів системи, таких як відображення планет з різних ракурсів, слідів руху та інших візуальних елементів.
- Сценарії та події. Розробка сценаріїв та подій, які можуть впливати на динамічність системи. Наприклад, введення додаткових об’єктів або зміна параметрів об’єктів під час виконання певних дій користувача.

Усе це можна реалізувати в методах Update() і FixedUpdate(). Вони використовуються для оновлення логіки і руху об’єктів у сцені. Ці методи можуть бути використані для реалізації динамічності в системі наступним чином:

1) Метод Update() викликається кожен кадр і може використовуватися для оновлення логіки та стану об’єктів у системі. Для реалізації динамічності можна використовувати цей метод для:
   - Контролювання руху об’єктів. Зміна положення, обертання або швидкості об’єктів.
   - Взаємодії з користувачем. Обробка введення користувача і виконання відповідних дії, такі як переміщення об’єктів, зміна параметрів або запуск анімацій.
   - Модифікації параметрів. Зміна параметрів об’єктів відповідно до потреб користувача, такі як зміна швидкості, напрямку руху або масштабу відображення моделі.
2) Метод FixedUpdate() викликається з фіксованою частотою і призначений для розрахунку фізики об’єктів у сцені. Цей метод особливо корисний для реалізації реалістичного руху та фізики об’єктів. Для впровадження динамічності можна:

- Моделювати гравітацію. Розрахунок гравітаційного впливу на об’єкти у системі, так щоб вони притягувалися один до одного або до центрального об’єкта.
- Синхронізувати рух об’єктів. Використовувати метод FixedUpdate() для синхронізації руху об’єктів у системі з фіксованою частотою оновлення, щоб досягти більш точної та передбачуваної поведінки.

Використання методів Update() і FixedUpdate() відкриває широкі можливості для реалізації динамічності в системі, дозволяючи контролювати рух, взаємодію та поведінку об’єктів у віртуальному середовищі.

У даному випадку в методі Update() виконується функція ControlTime(). Вона відповідає за керування часом у грі на основі натискань клавіш. Якщо натиснути клавішу «1», відбувається уповільнення часу. Якщо натиснути клавішу «3», відбувається прискорення часу. Якщо натиснуто клавішу «2», то або зберігаються дані про швидкість і швидкість об’єктів стає нульовою, або завантажуються збережені дані про швидкість. Код дозволяє контролювати процес часу у моделі.

Дана функція виконується саме в Update() для того, щоб забезпечувати управління часом кожен кадр, що дозволяє реагувати на натиснення клавіш і оновлювати час у грі в режимі реального часу.

В методі FixedUpdate() виконуються такі функції:

1) Gravity() – виконується обчислення гравітаційної сили між об’єктами у системі за допомогою закону всесвітнього тяжіння, обчислюється прискорення за другим законом Ньютона і це прискорення ділиться на частоту виконання методу FixedUpdate() і додається до значення швидкості об’єкта [16];
2) SimulatedTime() – виконується обчислення часу, який був симульований у моделі Сонячної системи (наприклад, за 1 секунду реального часу було симульовано 1 місяць в Сонячній системі);
3) CalculateOrbitParameters() – визначається ієрархія (що до чого сильніше притягується) і обчислюються значення кеплерових елементів орбіти для кожного об’єкта в системі;

4) RotateObjects() – виконується обертання об’єктів навколо своєї осі;
5) FindOrbitalParent() – обчислюються сфери Хілла для кожного об’єкту і визначаються орбітальні родичі;
6) SaveVelocityData() – зберігаються абсолютні значення швидкостей об’єктів для певних обчислень і відновлення руху об’єктів після закінчення зупинки (паузи) моделі.

До того ж, оці шість методів виконуються, якщо модель не стоїть на паузі. Таким чином, якщо не виконуються ці методи, то модель вважається зупиненою.

3.3. Створення і видалення об’єктів

У класі ObjectCreator реалізовано код, який відповідає за створення, видалення та управління об’єктами.

Змінна prefabs – це список префабів (шаблонів) об’єктів, які можна створити. У даному випадку вони були задані при створенні проєкту в середовищі Unity 3D. В цьому списку містяться такі об’єкти: Сонце, Меркурій, Венера, Земля, Марс, Юпітер, Сатурн, Уран, Нептун, Плутон, Місяць.

Змінні axisX, axisY, axisZ, positionChanger, velocityChanger визначають вісь та режим для управління позицією об’єкта або ж його швидкістю.

Функція Update() викликає дві інші функції – CheckKeyInput() та ChangerMode(). CheckKeyInput() реагує на натискання клавіш та виконує відповідні дії, такі як створення об’єкту, видалення об’єкту, вибір осі або зміна режиму зміни позиції/швидкості. ChangerMode() в залежності від активного режиму викликає функції MoveObject() або ChangeObjectVelocity(), що змінюють позицію або швидкість об’єктів відповідно.

CreateObject() створює об’єкт за вибраним префабом, додає його до списку об’єктів PlanetarySystem.celestials та встановлює його як активний об’єкт камери.

RemoveObject() видаляє активний об’єкт, знаходить найближчий об’єкт до камери та встановлює його як новий активний об’єкт камери.

InitialVelocity() встановлює таку початкову швидкість для нового об’єкта, враховуючи швидкість активного об’єкта камери, щоб новий об’єкт обертався по круговій орбіті.

GetPressedKeyCode() повертає код натиснутої клавіші.

Цей код дозволяє користувачу створювати, переміщувати, змінювати швидкість та видаляти об’єкти в грі, використовуючи введення клавіатури.

3.4. Розв’язання проблеми обмеження точності плаваючої крапки

Складність реалізації таких великих світів, як Сонячна система, у середовищі Unity 3D полягає в тому, що цей рушій відображає об’єкти менш деталізовано по мірі віддалення об’єкту від нульових координат, тому під час розробки масштабного світу потрібно розуміти обчислювальне обмеження.

Усі обчислення в Unity 3D працюють з типом даних float («плаваючий»), а одне значення з плаваючою крапкою має усього 4 байти на цілу частину і числа після крапки, і тому з цього випливає те, що чим більше чисел до крапки, тим менше чисел після крапки. Через це зменшується точність обчислень, і починається тремтіння усього простору.

Відомий метод вирішення цієї проблеми – “Floating Origin” [6]. Він полягає у тому, що умовний персонаж переходить відмітку в, припустимо, 1000 одиниць, то він переміщується на нульові координати, і весь світ переміщується відносно нього, тобто персонаж стає нулем координат.

Реалізація у коді використовує ідею про те, щоб переміщувати всі об’єкти відносно камери, а не навпаки, щоб забезпечити більш чітке відображення об’єктів у системі. У результаті, наприклад, далека карликова планета Плутон буде виглядати дуже чітко у порівнянні з тим випадком, коли лише камера переміщується відносно об’єктів.

Процес вирішення проблеми виглядає наступним чином:

1) Обчислюється відстань від позиції об’єкта до точки (0, 0, 0);

2) Якщо відстань більша або дорівнює 10 (значення може бути змінено в залежності від того, наскільки дрібні об’єкти мають бути чітко відображені), то починається процес корекції;
3) Обчислюється зміщення на основі поточної позиції об’єкта;
4) Для кожного об’єкта в списку відбувається наступне:
    - об’єкт тимчасово стає неактивним, щоб запобігти його відображення під час корекції;
    - якщо об’єкт має компонент TrailRenderer, то для кожної позиції в його траєкторії виконується зміщення;
    - сам об’єкт переміщується на величину зміщення;
    - об’єкт знову стає активним;
5) В кінці процесу корекції виконується зміщення позиції камери.

Таким чином, код обходить проблему точності з плаваючою крапкою, переносить усі об’єкти в системі відносно камери (рис. 3.5) і оновлює їх позиції і траєкторії для досягнення більш чіткого відображення. Відображення «до» і «після» зображено у Додатку В (рис. В.1, В.2).

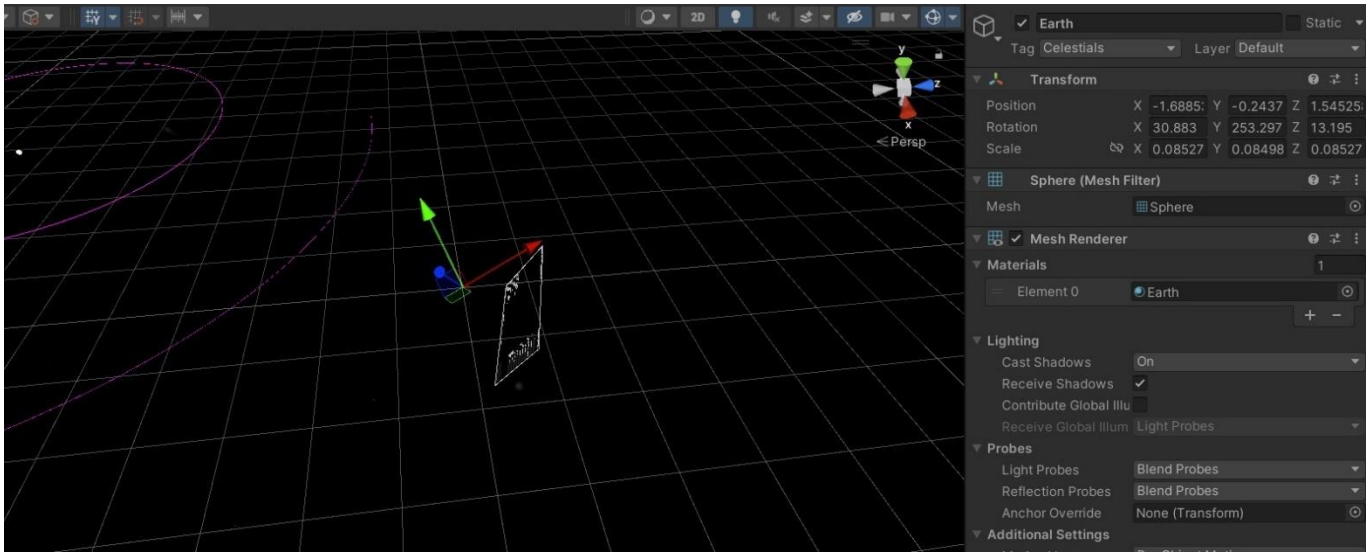

Рис. 3.5. Камера біля нульових координат

3.5. Управління камерою

Управління камерою виконується в класі CameraManagement. У ньому містяться такі змінні:

- *target* – визначає активний об’єкт, на який камера;
- *rotationSpeed*, *movementSpeed*, *scrollSpeed*, *minDistance*, *maxDistance*, *targetDistance* – встановлюються параметри руху та повороту камери, зокрема швидкість обертання, швидкість переміщення вздовж площини, швидкість наближення і віддалення, мінімальне і максимальне наближення до об’єкта, а також зберігається відстань до об’єкта.

Також у цьому класі виконуються такі функції:

- *SetInitialParametersForCamera()* – у методі Awake(), задаються початкові параметри камери;
- *SetTargetAndCameraPosition()* – у методі Start(), встановлюється активний об’єкт та початкова позиція камери;
- *SwitchCameraToObject()* – відповідає за переключення активного об’єкта та обробку введення клавіш для руху та зміни відстані камери;
- *MoveToZero()* – здійснює перехід камери до початкової позиції, якщо вона знаходиться далеко від неї, а також рухає всю модель до камери, не змінюючи відносну позицію камери до моделі;
- *CalculateCameraPosition()* – обчислює нову позицію камери на основі активного об’єкта та відстані до нього;
- *ScrollSpeedMultiply()* – змінює швидкість прокрутки камери в залежності від відстані до цільового об’єкта;
- *TrailRendererVision()* – налаштовує відображення стежок руху об’єктів на основі відстані до камери (якщо близько – не відображає, і навпаки);
- *RotateCamera()* – відповідає за поворот камери навколо цільового об’єкта з використанням введення клавіш;

- *MoveCamera()* – здійснює переміщення камери вздовж площини з використанням введення клавіш.

Цей код дозволяє користувачу керувати камерою в моделі, перемикати цільовий об’єкт, повертати та переміщувати камеру, а також збільшувати або зменшувати відстань до цільового об’єкта.

3.6. Панелі для введення і виведення

У класі InfoPanelController знаходяться змінні, які відображають поля введення тексту на панелі [17], а також статичні змінні, які встановлюють одиниці виміру для маси, довжини та часу.

У функції Update викликається функція UpdateOutput, яка оновлює вміст полів введення тексту на панелі.

У функції UpdateOutput отримується активний об’єкт, який встановлено в класі CameraManagement. Потім значення властивостей цього об’єкта виводяться в поля введення тексту на панелі.

Наприклад, властивість name активного об’єкта встановлюється у поле nameInputField.text, маса об’єкта ділиться на змінну massUnitEarth і виводиться в поле massInputField.text, радіуси об’єкта діляться на змінну lengthUnitKm і виводяться у поля equatorialRadiusInputField і polarRadiusInputField відповідно, і так далі.

Крім того, виводяться інші значення, такі як період обертання, нахил орбіти, відстань, швидкість, ексцентриситет орбіти та інші.

Також оновлюються поля, які відображають поточний час, активну вісь руху та режим редагування (позиція або швидкість) об’єкта.

Отже, цей код дозволяє оновлювати панель інформації на основі активного об’єкта та виводити різноманітну інформацію про нього на екран користувача.

У класі PageButton знаходяться змінні, які відповідають за посилання на різні об’єкти інтерфейсу користувача, такі як панель 1 (page1), панель 2 (page2), редактор (editorPage) та цільовий об’єкт (target).

Також в цьому класі є змінні, що відповідають за поля введення тексту на панелі, а також константи, які представляють значення маси Землі та її маси.

У функції Start при запуску моделі викликається функція ShowPanel1, яка відображає першу панель і приховує інші.

У функції Update викликається функція UpdateTarget, яка оновлює значення змінної target на цільовий об’єкт, який встановлено в класі CameraManagement.

Функції ShowPanel1, ShowPanel2 та ShowEditor відповідають за відображення відповідних панелей на екрані шляхом встановлення відповідних значень властивостей SetActive об’єктів page1, page2 та editorPage.

Функція ApplyCharacteristics застосовує характеристики, введені користувачем у поля введення тексту, до цільового об’єкта. Наприклад, якщо поле введення editName не є порожнім, значення цього поля присвоюється властивості name цільового об’єкта. Аналогічно, якщо поля введення editMass, editEquatorialRadius, editPolarRadius, editSiderealPeriodOfRotation та editInclinationToOrbit не є порожніми і містять числові значення, то ці значення використовуються для зміни відповідних характеристик цільового об’єкта.

Наприклад, маса об’єкта змінюється на основі значення з поля введення editMass, а радіуси об’єкта змінюються на основі значень з полів введення editEquatorialRadius та editPolarRadius.

У функції ChangeInclinationToOrbit змінюється нахил осі цільового об’єкта на основі значення, збереженого в класі ObjectData в змінній inclinationToOrbit цільового об’єкта. Зміна нахилу осі досягається шляхом обертання об’єкта за використанням Quaternion.Euler.

Отже, цей код дозволяє керувати відображенням панелей, змінювати характеристики цільового об’єкта на основі введених значень та змінювати нахил орбіти цільового об’єкта.

3.7. Додаткові можливості програмного продукту

Додатковими можливостями у даному програмному продукті є управління музичним супроводом, перезавантаження сцени і вихід з програми.

Управління музичним супроводом відбувається в класі MusicManagement. У класі ньому знаходяться змінні musicClip (аудіофайл музики) та audioSource (компонент Audio Source), які використовуються для відтворення музики. Значення змінної musicClip присвоюється через середовище Unity 3D, тобто в неї вставляється файл з музикою (рис. 3.6).

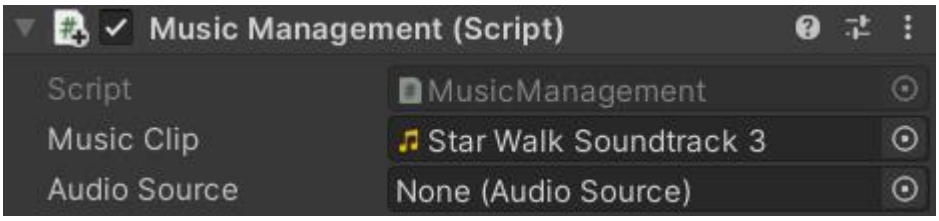

Рис. 3.6. Прикріплення файлу з музикою

У функції Start при запуску гри викликається функція GetAndPlayMusic, яка отримує посилання на компонент Audio Source, встановлює аудіофайл для відтворення та запускає його відтворення.

У функції Update перевіряється натискання клавіші M за допомогою Input.GetKeyDown(KeyCode.M). Якщо музика відтворюється (тобто audioSource.isPlaying дорівнює true), то функція Pause() зупиняє відтворення музики. У випадку, якщо музика призупинена, функція Play() відновлює її відтворення.

Отже, цей код дозволяє відтворювати музику в грі та управляти її відтворенням за допомогою натискання клавіші M.

У класі Controls знаходяться функції, CheckKeyInput, GetPressedKeyCode, RestartGame та QuitGame. Викликаються вони з функції Update. У функції Update кожен кадр перевіряється натискання клавіш за допомогою функції CheckKeyInput.

У функції CheckKeyInput перевіряється, чи було натиснуто будь-яку клавішу за допомогою. Якщо так, то викликається функція GetPressedKeyCode, яка повертає

код натиснутої клавіші. Залежно від коду клавіші виконується певна дія. В даному випадку, якщо натиснута клавіша LeftShift, викликається функція RestartGame, яка перезапускає гру. Якщо натиснута клавіша Escape, викликається функція QuitGame, яка закриває гру.

Функція GetPressedKeyCode перебирає всі можливі значення клавіш за допомогою System.Enum.GetValues(typeof(KeyCode)) та перевіряє, чи була натиснута яка-небудь клавіша за допомогою Input.GetKeyDown(keyCode). Якщо так, то повертається код натиснутої клавіші. Якщо ніяка клавіша не була натиснута, повертається KeyCode.None.

Функція RestartGame отримує поточний індекс активної сцени та викликає завантаження сцени з цим же індексом, щоб перезавантажити поточну сцену.

Функція QuitGame викликає функцію виходу з додатку, щоб завершити роботу в моделі.

Таким чином, цей код дозволяє перезапускати гру при натисканні клавіші LeftShift та закривати гру при натисканні клавіші Escape.

3.8. Перевірка правильності результатів обчислень

Щоб перевірити правильність обчислень орбітальних параметрів, необхідно надати орбіті планети певний нахил. Нехай цією планетою буде Меркурій, оскільки він швидше виконує повний оберт навколо Сонця. Натискаємо кнопку “Y” (зміна осі на Oy), потім “P” (зміна позиції) і на кілька секунд натискаємо “+”, щоб підняти планету по осі Oy.

Зафіксуємо орбітальні параметри Меркурія, які можна приблизно виміряти транспортиром, щоб переконатися, що вони обчислюються правильно. Такими параметрами будуть (рис. 3.7):

Рис. 3.7. Орбітальні параметри

- нахил орбіти: -20,2564°;
- довгота висхідного вузла: 321,3581° або -38,6419°;
- аргумент перицентру: 56,40721°;
- істинна аномалія: 55,19026°;
- середня аномалія: 34,17659°.

Вимірювати кутові орбітальні параметри будемо орієнтуючись на рис. 3.8.

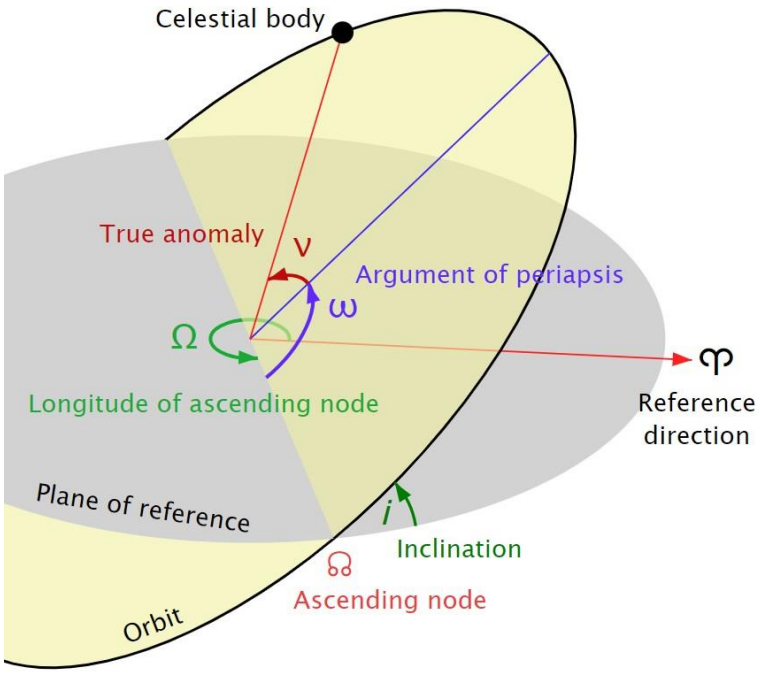

Рис. 3.8. Кеплерові елементи орбіти

Нахил орбіти – це кут між площиною орбіти і базовою площиною.

У середовищі Unity 3D виконаємо один повний оберт планети навколо Сонця і подивимось на залишений слід від планети таким чином, щоб камера була направлена паралельно базовій площині (площина Oxz) і орбіта планети виглядала як відрізок (рис. 3.9).

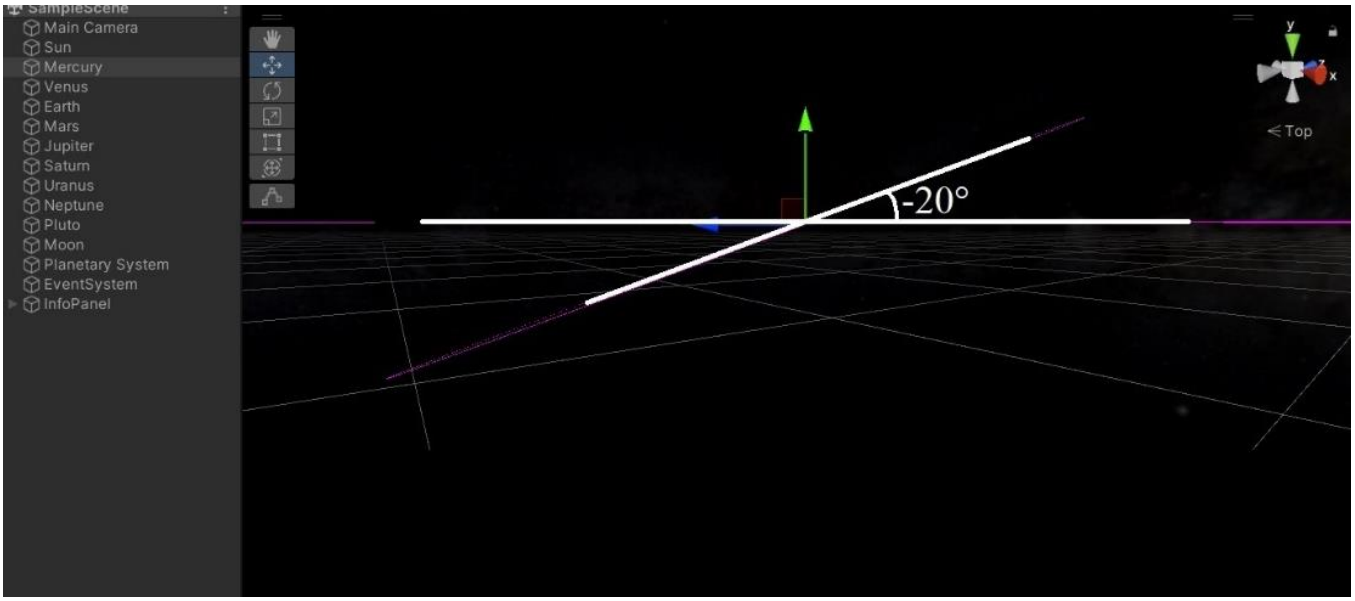

Рис. 3.9. Нахил орбіти

Отримали приблизно -20°, що відповідає результатам обчислення.

Довгота висхідного вузла – це кут між напрямком на нульову точку (додатній напрямок осі Ox) і напрямком на точку висхідного вузла (місце, де планета перетинає площину в північному напрямку).

Зупиняємо планету приблизно у висхідному вузлі і дивимось на орбіту перпендикулярно до базової площини (рис. 3.10).

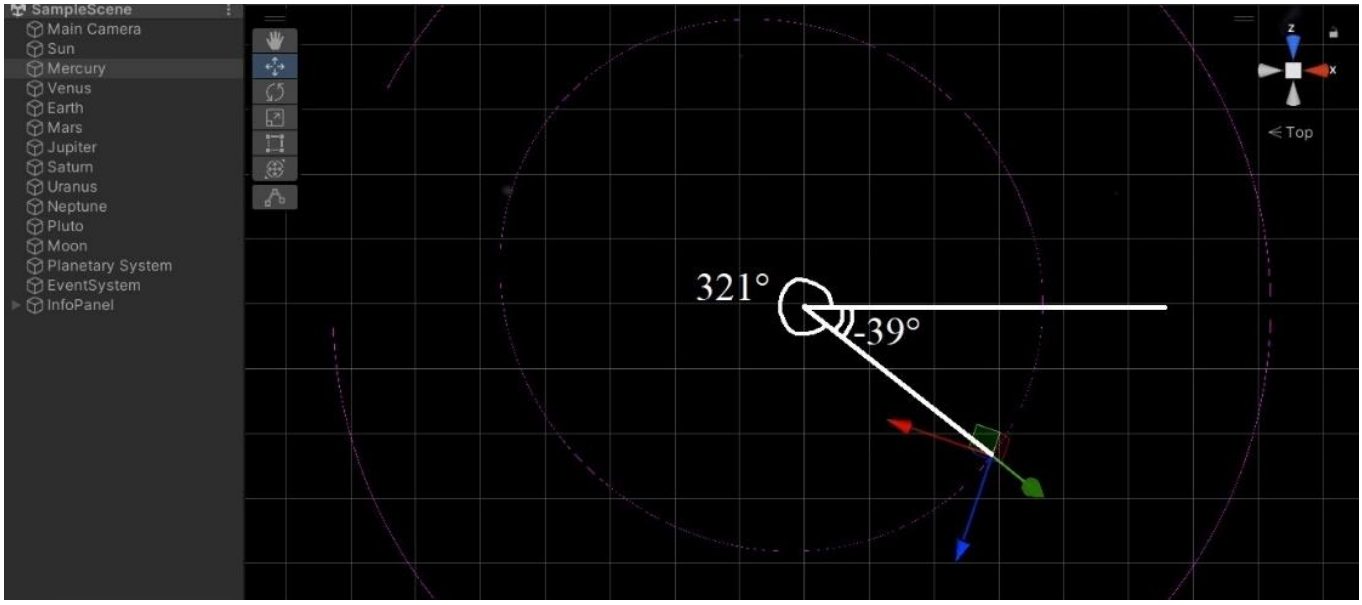

Рис. 3.10. Довгота висхідного вузла

Отримали приблизно -39° (або 321°), що відповідає результатам обчислення.

Аргумент перицентру – це кут між напрямком на висхідний вузол і напрямком на перицентр.

Визначити точне місце перицентру без певних обчислень складно, але, оскільки орбіта явно є еліптичною, то це місце можна визначити приблизно.

На рис. 3.11 видно, що планета все ще знаходиться у висхідному вузлі. Повертаємо камеру так, щоб вона дивилась приблизно перпендикулярно на площину орбіти. Між Сонцем і висхідним вузлом проводимо лінію, а потім проводимо ще лінію під відомим кутом аргументу перицентра з обчислень з місця, де ймовірно знаходиться перицентр.

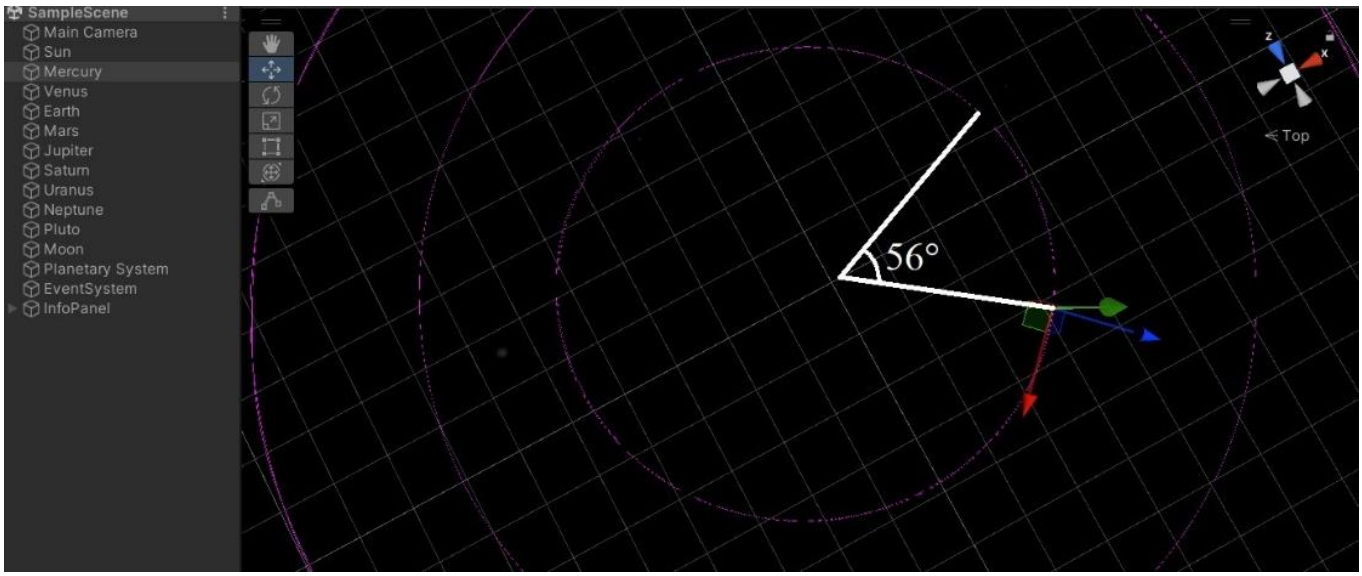

Рис. 3.11. Аргумент перицентру

Таким чином, друга лінія дійсно вказує на перицентр, і тому аргумент перицентру обчислений правильно.

Істинна аномалія – це кут між напрямком на перицентр і напрямком на поточне положення планети. Оскільки зараз планета знаходиться приблизно у висхідному вузлі, то і значення буде приблизно дорівнювати аргументу перицентру (рис. 3.12).

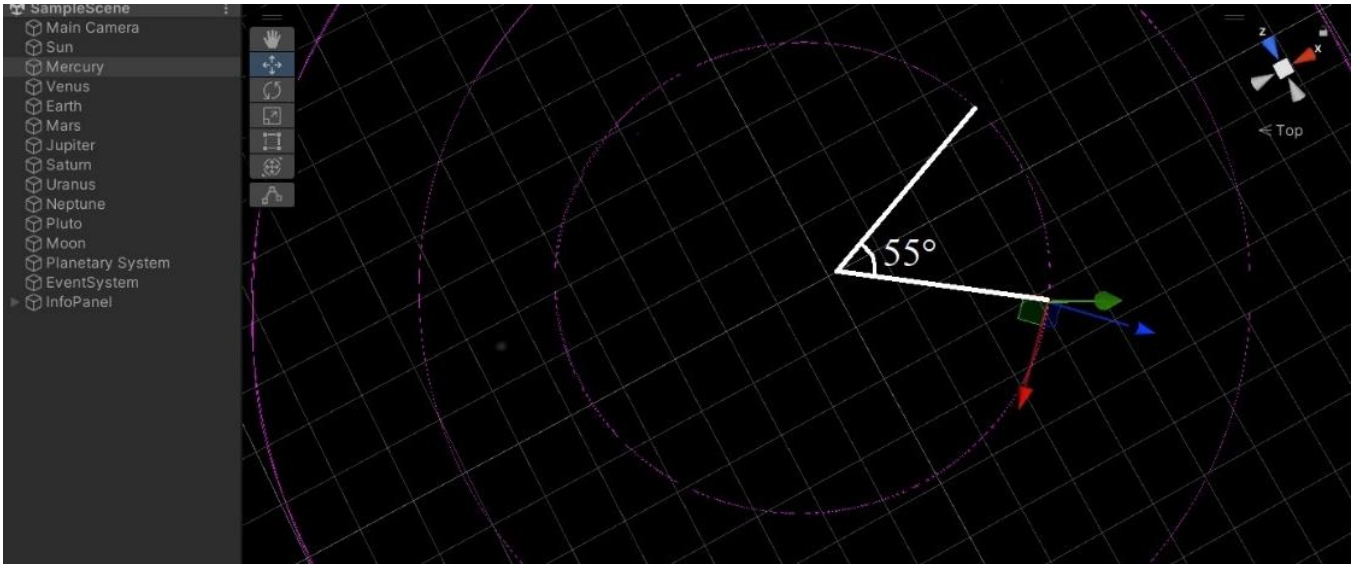

Рис. 3.12. Істинна аномалія

Таким чином, значення істинної аномалії в 55° є правильним, оскільки це значення близьке до аргументу перицентру і планета знаходиться приблизно у висхідному вузлі.

Щодо середньої аномалії, то її не можна виміряти безпосередньо за допомогою транспортира або іншого фізичного інструмента, оскільки вона являє собою параметр, пов’язаний з часом. Але чим менший нахил орбіти і чим більш кругла орбіта, тим ближче значення середньої аномалії до істинної аномалії. Так як нахил орбіти -20° та ексцентриситет 0,25 є досить великими, то і різниця між середньою та істинною аномалією в понад 21° є очікуваною.

Щоб отримати більш прийнятний результат, потрібно дослідити планету з більш круглою орбітою, наприклад, Венера (рис. 3.13, 3.14). Зараз у всіх планет нахил орбіти нульовий.

Рис. 3.13. Середня та істинна аномалія

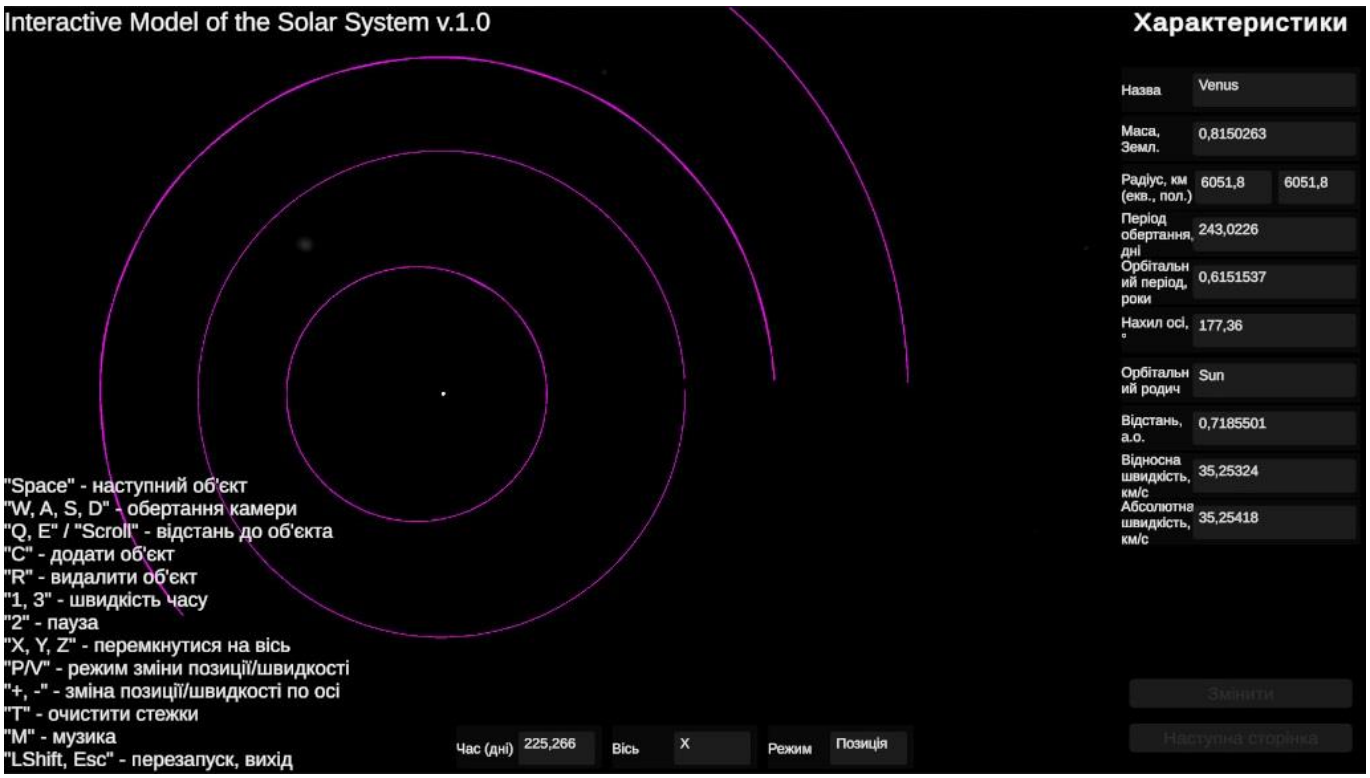

Рис. 3.14. Зовнішній вигляд орбіти Венери, інші характеристики планети

Отже, раніше висунуте ствердження є правильним, і обчислення середньої аномалії теж є правильними.

Також можна перевірити такі обчислювальні параметри, як:

- орбітальний період: 0,6151537 роки;
- відстань: 0,7185501 а.о.;
- велика піввісь: 0,723 а.о.;
- ексцентриситет: 0,00674.

Згідно з таблицею В.1, для Венери значення цих параметрів такі:

- орбітальний період: 19414166,4 с / (365,25 днів * 24 год * 60 хв * 60 с) = 0,6151978 роки;
- відстань: від 107477000000 м (перицентр) до 108939000000 м (апоцентр), тобто ділимо оці обидва значення на значення а.о. (149597870700 м) і отримуємо 0,718 а.о і 0,728 а.о.;
- велика піввісь: 108208000000 м / 149597870700 м = 0,723 а.о.;
- ексцентриситет: 0,006772.

Таким чином, ці значення обчислюються досить точно.

Наступні значення вже краще перевірити на прикладі Місяця (рис. 3.15):

Рис. 3.15. Характеристики Місяця

- орбітальний родич;
- відносна швидкість.

Орбітальний родич для Місяця – дійсно Земля (Earth). А відносна швидкість має бути між мінімальною і максимальною орбітальною швидкістю, згідно з табл. В.1: 0,97 і 1,082 км/с. Обчислена відносна швидкість для Місяця у певний момент часу: 1,014633 км/с, а отже обчислення відносної швидкості правильне.

Можливості додавання астрономічних об’єктів у модель, зміни характеристик та інші представлені в Додатку Д (рис. Д.1-Д.8).

Усі інші можливості керування описані в інструкції у нижньому лівому куті вікна створеного програмного продукту.

3.9. Висновки до розділу 3

У розділі 3 було проведено комплексну реалізацію базової (статичної) Сонячної системи та додано ряд функціональних можливостей, що значно покращують реалізм і користувацький досвід від програмного продукту.

Використовуючи механізми оновлення сцени (Update і FixedUpdate), була впроваджена фізична взаємодія між об’єктами, що дозволило досягти динамічності у системі. Це означає, що об’єкти Сонячної системи можуть взаємодіяти між собою, змінювати свої позиції та швидкості відповідно до фізичних законів.

Додавання функцій створення і видалення об’єктів надає користувачу можливість динамічно налаштовувати склад системи та спостерігати за змінами, що відбуваються.

Окрім того, була реалізована панель керування камерою, що дозволяє користувачу змінювати перегляд Сонячної системи з різних ракурсів та кутів. Це забезпечує більш гнучкий та особистий досвід взаємодії з програмним продуктом.

Важливим етапом було вирішення проблеми обмеження точності плаваючої крапки, що використовується для чисельних обчислень. Це дозволило досягти більшої точності і надійності результатів обчислень, що відображають астрономічні характеристики об’єктів.

Розроблено панелі для введення і виведення характеристик про астрономічні об’єкти, що спрощує користувачам редагування параметрів об’єктів і спостереження за їхніми характеристиками.

Також були представлені додаткові можливості програмного продукту, включаючи управління музичним супроводом, перезавантаження сцени та вихід з програми. Ці функції додають елементи зручності та естетичної насолоди для користувачів.

Під час реалізації була проведена перевірка правильності результатів обчислень, зокрема кеплерових елементів орбіти, орбітального періоду, відстані та відносної швидкості. Це дозволяє підтвердити, що програмний продукт надає

достовірну інформацію про орбітальні параметри об’єктів та правильно визначає їхні орбітальні взаємозв’язки.

В цілому, розділ «Реалізація у середовищі Unity 3D» демонструє успішну розробку та імплементацію базової Сонячної системи з додатковими функціональними можливостями, що сприяють покращенню користувацького досвіду та точності представлених даних.

ВИСНОВКИ

Кваліфікаційна робота по темі «Інтерактивна модель Сонячної системи у середовищі Unity 3D» є цікавим та складним завданням, яке вимагає поєднання знань з астрономії, програмування та графіки. Проєкт передбачає створення віртуальної моделі Сонячної системи, де користувачі зможуть взаємодіяти з планетами, супутниками та іншими об’єктами, а також отримувати інформацію про них.

Далі перелічимо основні висновки по даній кваліфікаційній роботі.

Детальне вивчення астрономії. Реалізація інтерактивної моделі Сонячної системи вимагає глибокого розуміння астрономічних процесів, характеристик планет та інших об’єктів. Для успішної реалізації проєкту дослідження астрономії та збирання достовірної інформації є необхідним.

Використання Unity 3D. Unity 3D є потужним інструментом для розробки ігор та віртуальних середовищ. Використання Unity 3D дозволяє створити реалістичну та інтерактивну модель Сонячної системи, використовуючи його функціональність для моделювання, анімації та візуалізації.

Моделювання об’єктів. Проєкт вимагає моделювання планет, супутників, комет, астероїдів та інших об’єктів Сонячної системи. Розробка детальних 3D-моделей та їх анімація відтворюють реалістичний вигляд об’єктів та їх рухів у просторі.

Реалістична фізика. Для створення вірогідної інтеракції між об’єктами у моделі Сонячної системи необхідно враховувати фізичні закони, такі як гравітація, рух тіл у космічному просторі та інші параметри. Використання фізичних двигунів або самостійної реалізації фізичних обчислень дозволить забезпечити реалістичну поведінку об’єктів у моделі.

Інтерактивність та користувацький інтерфейс. Для забезпечення інтерактивності проєкту необхідно реалізувати функціонал, який дозволить користувачам взаємодіяти з об’єктами Сонячної системи. Це можуть бути

можливості зміни масштабу, обертання, переміщення об’єктів, відтворення анімацій та отримання інформації про планети та інші об’єкти.

Візуалізація та освітня цінність. Інтерактивна модель Сонячної системи може бути використана для освітніх цілей, дозволяючи користувачам вивчати астрономію та досліджувати просторові взаємодії між об’єктами. Правильна візуалізація та інтерактивність моделі сприяють кращому розумінню астрономічних процесів та підвищенню інтересу до науки.

У цьому дипломному проєкті використовуються знання з астрономії, програмування та графіки для розробки інтерактивної моделі Сонячної системи у середовищі Unity 3D. Результатом проєкту буде реалістична та інтерактивна віртуальна модель, яка дозволить користувачам досліджувати та вивчати Сонячну систему. Проєкт може мати значну освітню цінність та зацікавити широке коло людей, які цікавляться астрономією та космосом.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Solar system planets, order and formation: A guide | Space [Electronic resource]. – Access mode: https://www.space.com/16080-solar-system-planets.html (lastaccess: 10.06.2023). – Title from the screen.
2. Meet the Solar System's five official dwarf planets | The Planetary Society [Electronic resource]. – Access mode: https://www.planetary.org/articles/meet-the-dwarf-planets (lastaccess: 10.06.2023). – Title from the screen.
3. Hocking J. Unity in Action, Third Edition / J. Hocking – Shelter Island, NY: Manning Publications, 2022. – 416 с. – Бібліогр.: с. 3-22.
4. Bond J. Introduction to Game Design, Prototyping and Development / J. Bond – Boston, MA: Addison-Wesley, 2019. – 1024 с. – Бібліогр.: с. 330-348.
5. What is Unity? – A Guide for One of the Top Game Engines – GameDev Academy [Electronic resource]. – Access mode: https://gamedevacademy.org/what-is-unity/ (lastaccess: 10.06.2023). – Title from the screen.
6. Floating Origin in Unity [Electronic resource]. – Access mode: https://manuel-rauber.com/2022/04/06/floating-origin-in-unity/ (lastaccess: 10.06.2023). – Title from the screen.
7. Orbits and Kepler's Laws | NASA Solar System Exploration [Electronic resource]. – Access mode: https://solarsystem.nasa.gov/resources/310/orbits-and-keplers-laws/ (lastaccess: 10.06.2023). – Title from the screen.
8. Newton's Laws of Motion – Glenn Research Center | NASA [Electronic resource]. – Access mode: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/newtons-laws-of-motion/ (lastaccess: 10.06.2023). – Title from the screen.
9. A Study about the Integration of the Elliptical Orbital Motion Based on a Special One-Parametric Family of Anomalies | Hindawi [Electronic resource]. – 2014. – № 18, Vol. 2014 – Access mode: https://www.hindawi.com/journals/aaa/2014/162060/ (lastaccess: 10.06.2023). – Title from the screen.

10. The 6 Key Principles of UI Design | Maze [Electronic resource]. – Access mode: https://maze.co/collections/ux-ui-design/ui-design-principles/ (lastaccess: 10.06.2023). – Title from the screen.

11. What are the principles of user-interface design [Electronic resource]. – Access mode: https://www.tutorialspoint.com/what-are-the-principles-of-user-interface-design (lastaccess: 10.06.2023). – Title from the screen.

12. Unity – Manual: Unity User Manual 2021.3 (LTS) [Electronic resource]. – Access mode: https://docs.unity3d.com/Manual/index.html (lastaccess: 10.06.2023). – Title from the screen.

13. Lunar and Planetary Science at the NSSDCA [Electronic resource]. – Access mode: https://nssdc.gsfc.nasa.gov/planetary/ (lastaccess: 10.06.2023). – Title from the screen.

14. Objects in the Night Sky – In-The-Sky.org [Electronic resource]. – Access mode: https://in-the-sky.org/data/data.php (lastaccess: 10.06.2023). – Title from the screen.

15. Curtis. H. Orbital Mechanics for Engineering Students / H. Curtis; Embry-Riddle Aeronautical University – Daytona Beach, FL: Butterworth-Heinemann, 2005. – 692 с. – Бібліогр.: с. 158-164.

16. Simulating Gravity in Unity [Electronic resource]. – Access mode: https://mikhail-szugalew.medium.com/simulating-gravity-in-unity-ae8258a80b6d#:~:text=Explained%20simply%2C%20this%20law%20states,distance%20between%20them%20(r) (lastaccess: 10.06.2023). – Title from the screen.

17. Interaction Components | Unity UI | 2.0.0 [Electronic resource]. – Access mode: https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/comp-UIInteraction.html (last access: 10.06.2023). – Title from the screen.

ДОДАТКИ

Додаток А

Таблиця А.1

Необхідні дані про астрономічні об’єкти

	Сонце	Меркурій	Венера	Земля	Місяць	Марс
Маса (10^{24} кг)	1988500	0,33022	4,8675	5,972168	0,07342	0,64171
Екваторіальний радіус (м)	695508000	2440500	6051800	6378137	1738100	3396200
Полярний радіус (м)	695501700	2438300	6051800	6356752	1736000	3376200
Перицентр (м)	-	46001009000	107477000000	147098450000	362600000	206650000000
Апоцентр (м)	-	69817445000	108939000000	152097597000	405400000	249261000000
Максимальна орбітальна швидкість (м/с)	-	58970	35260	30290	1082	26500
Мінімальна орбітальна швидкість (м/с)	-	38860	34780	29290	970	21970
Велика піввісь (м)	-	57909227000	108208000000	149598023000	384399000	227939366000
Ексцентриситет	-	0,20563593	0,006772	0,0167086	0,0549	0,093412
Нахил орбіти (°)	-	7	3,39458	0	5,145	1,85

Закінчення таблиці А.1

	Сонце	Меркурій	Венера	Земля	Місяць	Марс
Довгота висхідного вузла (°)	-	48,33167	76,68	-11,26064	125,08	49,57854
Аргумент перицентру (°)	-	29,124279	54,884	114,20783	318,15	286,46
Середня аномалія (°)	-	174,795884	50,115	358,617	135,27	19,412
Сидеричний період обертання (с)	2164320	5067360	20997152,64	86164,1	2360620,8	88642,7
Орбітальний період (с)	-	7600530,24	19414166,4	31558149,76	2360591,5	59355072
Нахил до орбіти (°)	7,25	0,034	177,36	23,4392811	1,5424	25,19

Таблиця А.2

Необхідні дані про астрономічні об’єкти

	Юпітер	Сатурн	Уран	Нептун	Плутон
Маса (10^{24} кг)	1898,13	568,34	86,811	102,413	0,01303
Екваторіальний радіус (м)	71492000	60268000	25559000	24764000	1188300
Полярний радіус (м)	66854000	54364000	24973000	24341000	1188300

Продовження таблиці А.2

	Юпітер	Сатурн	Уран	Нептун	Плутон
Перицентр (м)	7405950000000	1352550000000	2735560000000	4471050000000	4436820000000
Апоцентр (м)	8163630000000	1514500000000	3006390000000	4558857000000	7375930000000
Максимальна орбітальна швидкість (м/с)	13720	10140	7130	5470	6100
Мінімальна орбітальна швидкість (м/с)	12440	9140	6490	5370	3710
Велика піввісь (м)	7784790000000	1433530000000	2870972000000	4514953000000	5906380000000
Ексцентриситет	0,048393	0,0565	0,04717	0,008586	0,2488
Нахил орбіти (°)	1,303	2,485	0,773	1,77	17,16
Довгота висхідного вузла (°)	100,464	113,665	74,006	131,783	110,299
Аргумент перицентру (°)	273,867	339,392	96,998857	273,187	113,834
Середня аномалія (°)	20,02	317,02	142,238600	259,883	14,53
Сидеричний період обертання (с)	35730	38018	62064	57996	551856
Орбітальний період (с)	374335776	929596608	2651486400	5200418592	7823780928

Закінчення таблиці А.2

	Юпітер	Сатурн	Уран	Нептун	Плутон
Нахил до орбіти (°)	3,13	26,73	97,77	28,32	122,53

PlanetarySystem.cs

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

public class PlanetarySystem : MonoBehaviour
{
    public static float time = 0f;
    public static float massUnitDivider = Mathf.Pow(10f, 24);
    public static float lengthUnitDivider = 149597870.700f;
    public static float timeStep = 2629800f / 30f;
    public static float G = 6.6743f * Mathf.Pow(10f, -11) * massUnitDivider / Mathf.Pow(lengthUnitDivider, 3);
    public static List<GameObject> celestials;
    public static bool paused = false;
    void Awake()
    {
        AddObjectsToCelestials();
        AssignBasicCharacteristicsToObjects();
        CalculateOrbitParameters();
        InclinationToOrbit();
    }

    void AddObjectsToCelestials()
    {
        celestials = new List<GameObject>();
        foreach (GameObject celestial in GameObject.FindGameObjectsWithTag("Celestials"))
        {
```

```
        celestials.Add(celestial);
    }
    celestials.Add(GameObject.FindGameObjectWithTag("Central Object"));
}

private void Update()
{
    ControlTime();
}

void ControlTime()
{
    if (Input.GetKey(KeyCode.Alpha1))
    {
        timeStep *= 0.99f;
        ChangeVelocity(0.99f);
    }
    else if (Input.GetKey(KeyCode.Alpha3))
    {
        timeStep *= 1.01f;
        ChangeVelocity(1.01f);
    }
    else if (Input.GetKeyDown(KeyCode.Alpha2))
    {
        if (!paused)
        {
            SaveVelocityData();
            ChangeVelocity(0f);
        }
```

```
        else
        {
            LoadVelocityData();
        }
        paused = !paused;
    }
}

void SaveVelocityData()
{
    foreach (GameObject a in celestials)
    {
        a.GetComponent<ObjectData>().absoluteVelocity =
a.GetComponent<Rigidbody>().velocity;
    }
}

void LoadVelocityData()
{
    foreach (GameObject a in celestials)
    {
        a.GetComponent<Rigidbody>().velocity =
a.GetComponent<ObjectData>().absoluteVelocity;
    }
}

private void FixedUpdate()
{
    if (!paused)
```

```
        {
            Gravity();
            SimulatedTime();
            CalculateOrbitParameters();
            RotateObjects();
            FindOrbitalParent();
            SaveVelocityData();
        }
    }

    void InclinationToOrbit()
    {
        foreach (GameObject a in celestials)
        {
            float inclination = a.GetComponent<ObjectData>().inclinationToOrbit;
            Quaternion newRotation = Quaternion.Euler(-inclination, 0f, 0f);
            a.transform.localRotation = newRotation * a.transform.localRotation;
        }
    }

    void RotateObjects()
    {
        foreach (GameObject a in celestials)
        {
            if (a.GetComponent<ObjectData>().siderealPeriodOfRotation != 0f)
            {
                float rotation = Time.fixedDeltaTime * timeStep * 360f /
a.GetComponent<ObjectData>().siderealPeriodOfRotation;
                a.transform.Rotate(0f, -rotation, 0f);
```

```
        } else
        {
            a.transform.Rotate(0f, 0f, 0f);
        }

    }
}

void CalculateOrbitParameters()
{
    foreach (GameObject obj in celestials)
    {
        if (obj.GetComponent<ObjectData>().orbitalParent != null)
        {
            // Велика піввісь
            float m1 = obj.GetComponent<ObjectData>().mass;
            float m2 = obj.GetComponent<ObjectData>().orbitalParent.GetComponent<ObjectData>().mass;
            float mu = (G * Mathf.Pow(lengthUnitDivider, 3) / massUnitDivider) * (m1 + m2);
            Vector3 vo_ = lengthUnitDivider * obj.GetComponent<Rigidbody>().velocity / timeStep;
            Vector3 vc_ = lengthUnitDivider * obj.GetComponent<ObjectData>().orbitalParent.GetComponent<Rigidbody>().velocity / timeStep;
            Vector3 v_ = vo_ - vc_;
            obj.GetComponent<ObjectData>().relativeVelocity = v_;
            float v = v_.magnitude;
            obj.GetComponent<ObjectData>().relativeVelocityMagnitude = v;
```

```
        Vector3 ro_ = lengthUnitDivider * obj.transform.localPosition;
        Vector3 rc_ = lengthUnitDivider * obj.GetComponent<ObjectData>().orbitalParent.transform.localPosition;
        Vector3 r_ = ro_ - rc_;
        float r = r_.magnitude;
        obj.GetComponent<ObjectData>().distance = r;
        float epsilon = (Mathf.Pow(v, 2) / 2) - (mu / r);
        float a = -mu / (2 * epsilon);
        obj.GetComponent<ObjectData>().calculatedSemimajorAxis = a;

        // Ексцентриситет
        Vector3 h_ = Vector3.Cross(r_, v_);
        Vector3 e_ = (Vector3.Cross(v_, h_) / mu) - (r_ / r);
        float e = e_.magnitude;
        obj.GetComponent<ObjectData>().calculatedEccentricity = e;

        // Нахил орбіти
        float i = -Mathf.Acos(-h_.y / h_.magnitude); // rad
        float iDeg = Mathf.Rad2Deg * i;
        if (iDeg >= 360f)
        {
            iDeg -= 360f;
        }
        obj.GetComponent<ObjectData>().calculatedOrbitInclination = iDeg;

        // Довгота висхідного вузла
        Vector3 n_ = Vector3.Cross(Vector3.up, h_);
        float omega = 0;
        if (i == 0)
```

```
        {
            omega = 0;
        }
        else if (n_.z >= 0)
        {
            omega = Mathf.Acos(n_.x / n_.magnitude);
        }
        else if (n_.z < 0)
        {
            omega = 2f * Mathf.PI - Mathf.Acos(n_.x / n_.magnitude);
        }
        float omegaDeg = Mathf.Rad2Deg * omega;
        if (omegaDeg >= 360f)
        {
            omegaDeg -= 360f;
        }
        obj.GetComponent<ObjectData>().calculatedLengthOfTheAscendingNode = omegaDeg;

        // Аргумент перицентру
        float w = Mathf.Acos(Vector3.Dot(n_, e_) / (n_.magnitude * e_.magnitude));
        float wDeg = Mathf.Rad2Deg * w;
        if (wDeg >= 360f)
        {
            wDeg -= 360f;
        }
        obj.GetComponent<ObjectData>().calculatedArgumentOfThePericenter = wDeg;
```

```
            // Істинна аномалія
            float f = Mathf.Acos(Vector3.Dot(e_, r_) / (e_.magnitude * r_.magnitude));
            if (r * f < 0)
            {
                f = 2f * Mathf.PI - f;
            }
            float fDeg = Mathf.Rad2Deg * f;
            if (fDeg >= 360f)
            {
                fDeg -= 360f;
            }
            obj.GetComponent<ObjectData>().calculatedTrueAnomaly = fDeg;

            // Середня аномалія
            float E = Mathf.Acos(1 / e - r_.magnitude / (a * e));
            float M = E - e * Mathf.Sin(E);
            float MDeg = Mathf.Rad2Deg * M;
            if (MDeg >= 360f)
            {
                MDeg -= 360f;
            }
            obj.GetComponent<ObjectData>().calculatedMeanAnomaly = MDeg;

            // Орбітальний період
            obj.GetComponent<ObjectData>().calculatedOrbitalPeriod = 2f * Mathf.PI *
Mathf.Sqrt(Mathf.Pow(a, 3) / mu);
        }
    }
}
```

```
    void FindOrbitalParent()
    {
        // Обчислення радіусу Хілла
        foreach (GameObject obj1 in celestials)
        {
            float hillRadius = float.MaxValue;
            float m = obj1.GetComponent<ObjectData>().mass;
            foreach (GameObject obj2 in celestials)
            {
                float M = obj2.GetComponent<ObjectData>().mass;
                if (m < M)
                {
                    float m1 = m;
                    float m2 = M;
                    float mu = (G * Mathf.Pow(lengthUnitDivider, 3) / massUnitDivider) * (m1 +
m2);
                    Vector3 vo_ = lengthUnitDivider *
obj1.GetComponent<Rigidbody>().velocity / timeStep;
                    Vector3 vc_ = lengthUnitDivider *
obj2.GetComponent<Rigidbody>().velocity / timeStep;
                    Vector3 v_ = vo_ - vc_;
                    float v = v_.magnitude;
                    Vector3 ro_ = lengthUnitDivider * obj1.transform.localPosition;
                    Vector3 rc_ = lengthUnitDivider * obj2.transform.localPosition;
                    Vector3 r_ = ro_ - rc_;
                    float r = r_.magnitude;
                    float epsilon = (Mathf.Pow(v, 2) / 2) - (mu / r);
                    float a = -mu / (2 * epsilon);
```

```
            Vector3 h_ = Vector3.Cross(r_, v_); // specific angular momentum vector

            Vector3 e_ = (Vector3.Cross(v_, h_) / mu) - (r_ / r);
            float e = e_.magnitude; // magnitude of eccentricity vector

            float hillRadiusX = a * (1f - e) * Mathf.Pow(m / (3f * (M)), 1f / 3); // Corrected calculation for Hill radius
            if (hillRadiusX < hillRadius && hillRadiusX > 0f)
            {
               hillRadius = hillRadiusX;
            }
         }
      }
      obj1.GetComponent<ObjectData>().hillRadius = hillRadius;
      obj1.GetComponent<ObjectData>().hillRadiusAU = hillRadius / 1000000000f;
   }
   foreach (GameObject obj1 in celestials)
   {
      float m = obj1.GetComponent<ObjectData>().mass;
      float vParent = float.MaxValue;
      GameObject parent = null;
      foreach (GameObject obj2 in celestials)
      {
         float M = obj2.GetComponent<ObjectData>().mass;
         float r = Vector3.Distance(obj1.transform.position, obj2.transform.position) * lengthUnitDivider;
         if (m < M && r <= obj2.GetComponent<ObjectData>().hillRadius)
         {
```

```
            Vector3 vo_ = lengthUnitDivider * obj1.GetComponent<Rigidbody>().velocity / timeStep;
            Vector3 vc_ = lengthUnitDivider * obj2.GetComponent<Rigidbody>().velocity / timeStep;
            Vector3 v_ = vo_ - vc_;
            float vCur = v_.magnitude;
            if (vCur < vParent)
            {
               vParent = vCur;
               parent = obj2;
            }
         }
      }
      obj1.GetComponent<ObjectData>().orbitalParent = parent;
   }
}

void AssignBasicCharacteristicsToObjects()
{
   foreach (GameObject a in celestials)
   {
      a.GetComponent<Rigidbody>().mass = a.GetComponent<ObjectData>().mass / massUnitDivider;
      a.transform.localScale = new Vector3(
         2f * a.GetComponent<ObjectData>().equatorialRadius / lengthUnitDivider,
         2f * a.GetComponent<ObjectData>().polarRadius / lengthUnitDivider,
         2f * a.GetComponent<ObjectData>().equatorialRadius / lengthUnitDivider);

      if (a.GetComponent<ObjectData>().orbitalParent == null)
```

```
        {
            a.transform.localPosition = new Vector3(
                a.GetComponent<ObjectData>().pericenter / lengthUnitDivider,
                0f / lengthUnitDivider,
                0f / lengthUnitDivider);
            a.GetComponent<Rigidbody>().velocity = new Vector3(
                0f * timeStep / lengthUnitDivider,
                0f * timeStep / lengthUnitDivider,
                a.GetComponent<ObjectData>().maximumOrbitalSpeed * timeStep / lengthUnitDivider);
        }
    }
    foreach (GameObject a in celestials)
    {
        if (a.GetComponent<ObjectData>().orbitalParent != null && !a.CompareTag("Central Object"))
        {
            a.transform.localPosition = new Vector3(
                a.GetComponent<ObjectData>().pericenter / lengthUnitDivider,
                0f / lengthUnitDivider,
                0f / lengthUnitDivider) + a.GetComponent<ObjectData>().orbitalParent.transform.localPosition;
            a.GetComponent<Rigidbody>().velocity = new Vector3(
                0f * timeStep / lengthUnitDivider,
                0f * timeStep / lengthUnitDivider,
                a.GetComponent<ObjectData>().maximumOrbitalSpeed * timeStep / lengthUnitDivider) + a.GetComponent<ObjectData>().orbitalParent.GetComponent<Rigidbody>().velocity;
        }
```

```
    }
}

void SimulatedTime()
{
    time += timeStep * Time.fixedDeltaTime / 86400f;
}

void ChangeVelocity(float k)
{
    foreach (GameObject a in celestials)
    {
        if (!paused)
        {
            Rigidbody rb = a.GetComponent<Rigidbody>();
            if (rb != null)
            {
                rb.velocity *= k;
            }
        }
        else
        {
            a.GetComponent<ObjectData>().absoluteVelocity *= k;
        }
    }
}

void Gravity()
{
```

```
        foreach (GameObject a in celestials)
        {
            foreach (GameObject b in celestials)
            {
                if (!a.Equals(b))
                {
                    float m1 = a.GetComponent<Rigidbody>().mass;
                    float m2 = b.GetComponent<Rigidbody>().mass;
                    float r = Vector3.Distance(a.transform.position, b.transform.position);
                    Vector3 F = (b.transform.position - a.transform.position).normalized * (G * m1 * m2 / (r * r));
                    Vector3 acceleration = F / m1;
                    a.GetComponent<Rigidbody>().velocity += timeStep * timeStep * acceleration * Time.fixedDeltaTime;
                }
            }
        }
    }
}
```

Рис. В.1. Відображення далекого об’єкта до розв’язання проблеми обмеження точності плаваючої крапки

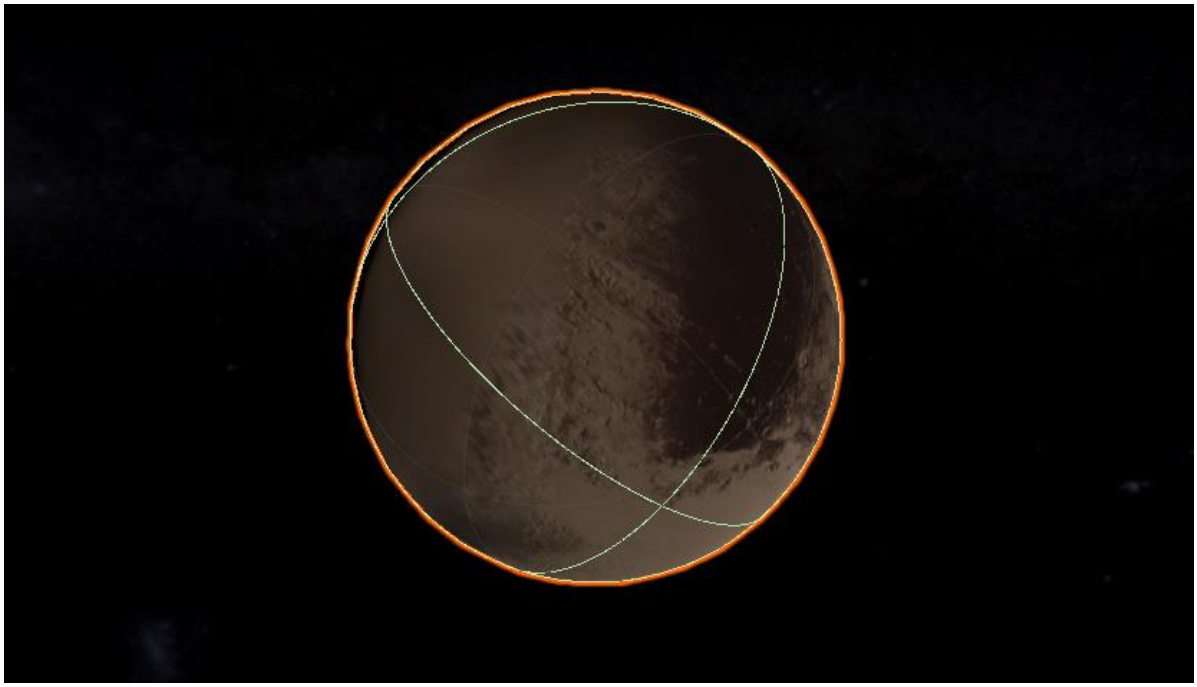

Рис. В.2. Відображення далекого об’єкта після розв’язання проблеми обмеження точності плаваючої крапки

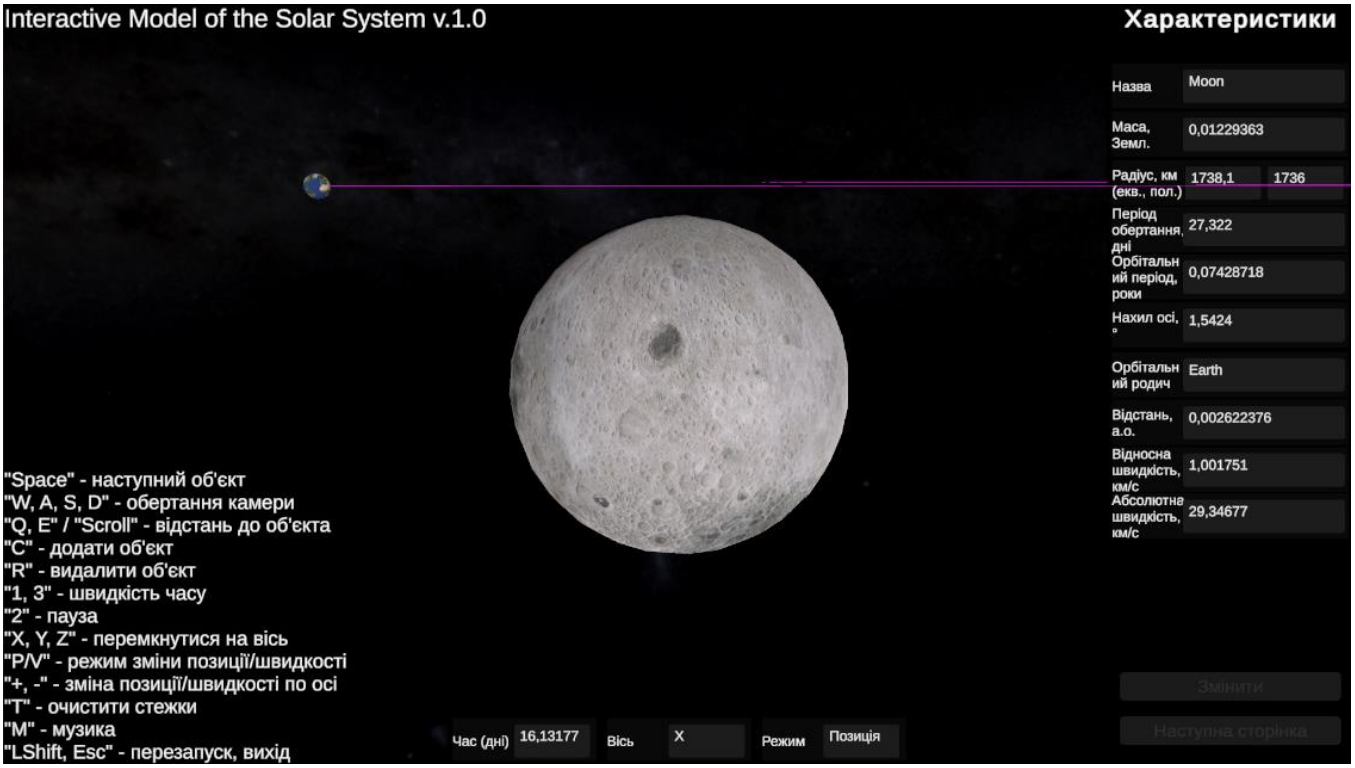

Рис. Д.1. Вигляд на Землю з Місяця

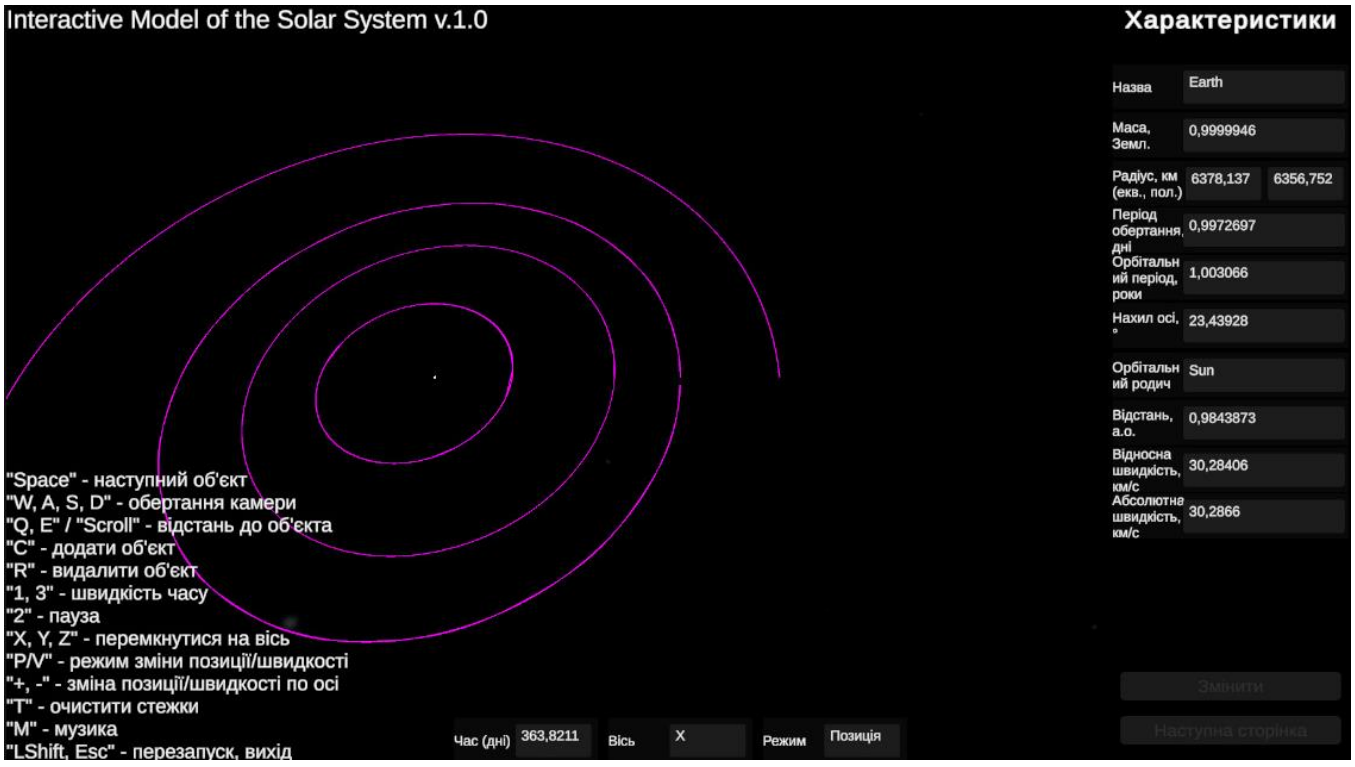

Рис. Д.2. Земля (третя орбіта) станом на 364-ий день майже закінчила один оберт навколо Сонця

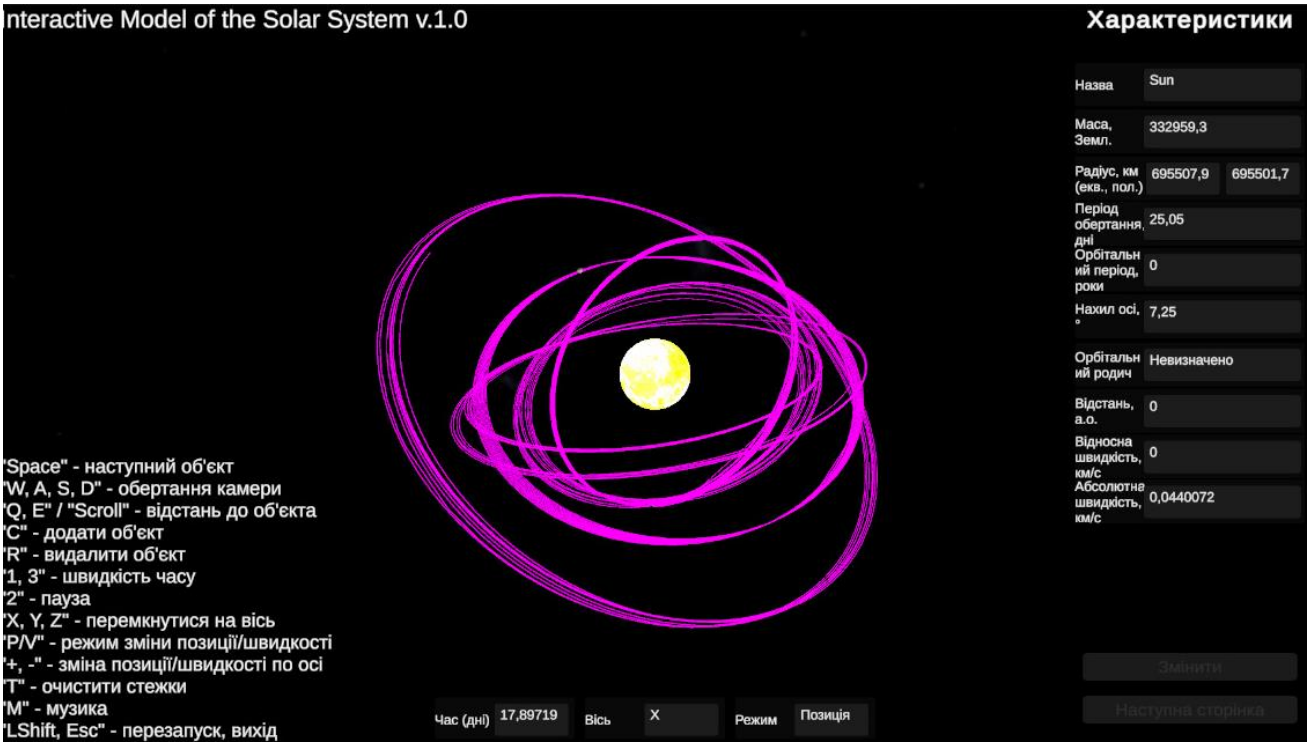

Рис. Д.3. Функція додавання об’єктів. П’ять астрономічних об’єктів запущено по круговим орбітам навколо Сонця

Рис. Д.4. Юпітер після застосування нових значень характеристик

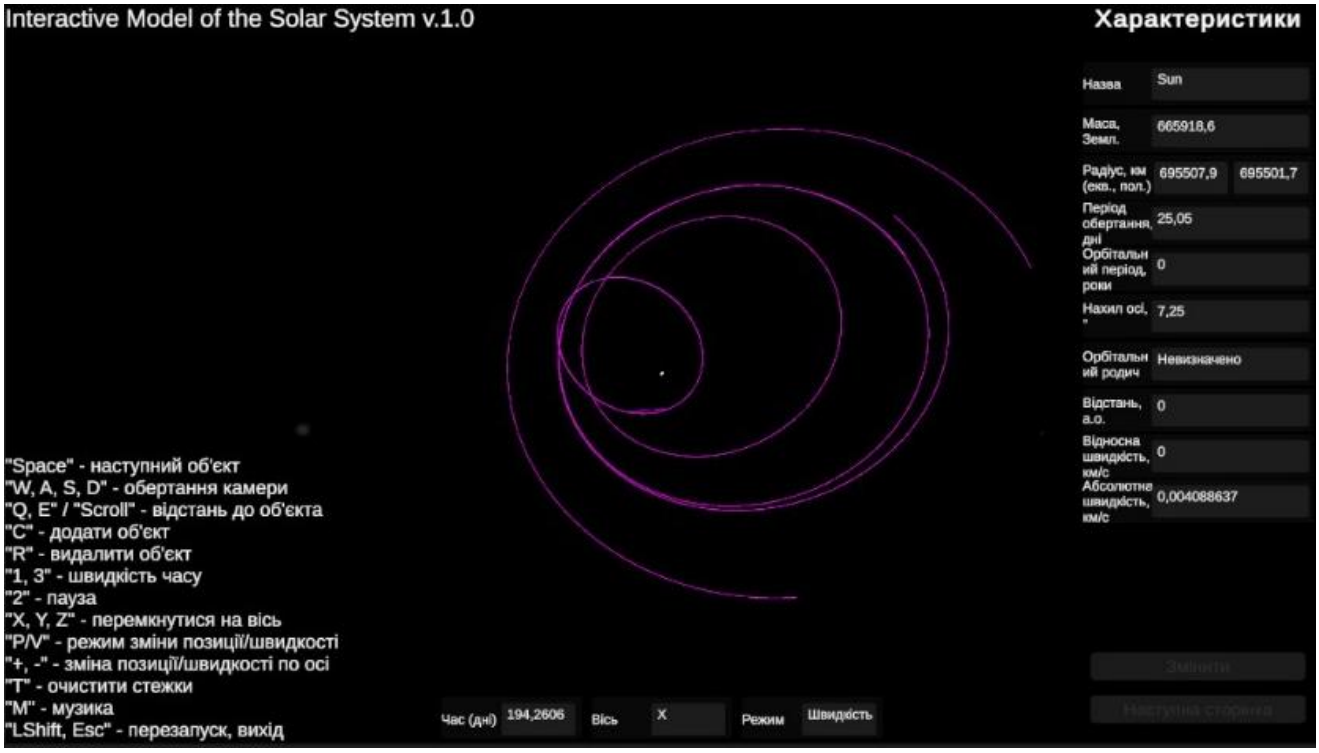

Рис. Д.5. Поведінка орбіт після збільшення маси Сонця в 2 рази

Рис. Д.6. Меркурій став орбітальним родичем Сонця після збільшення його маси до 600000 мас Землі, Венера викинута із системи

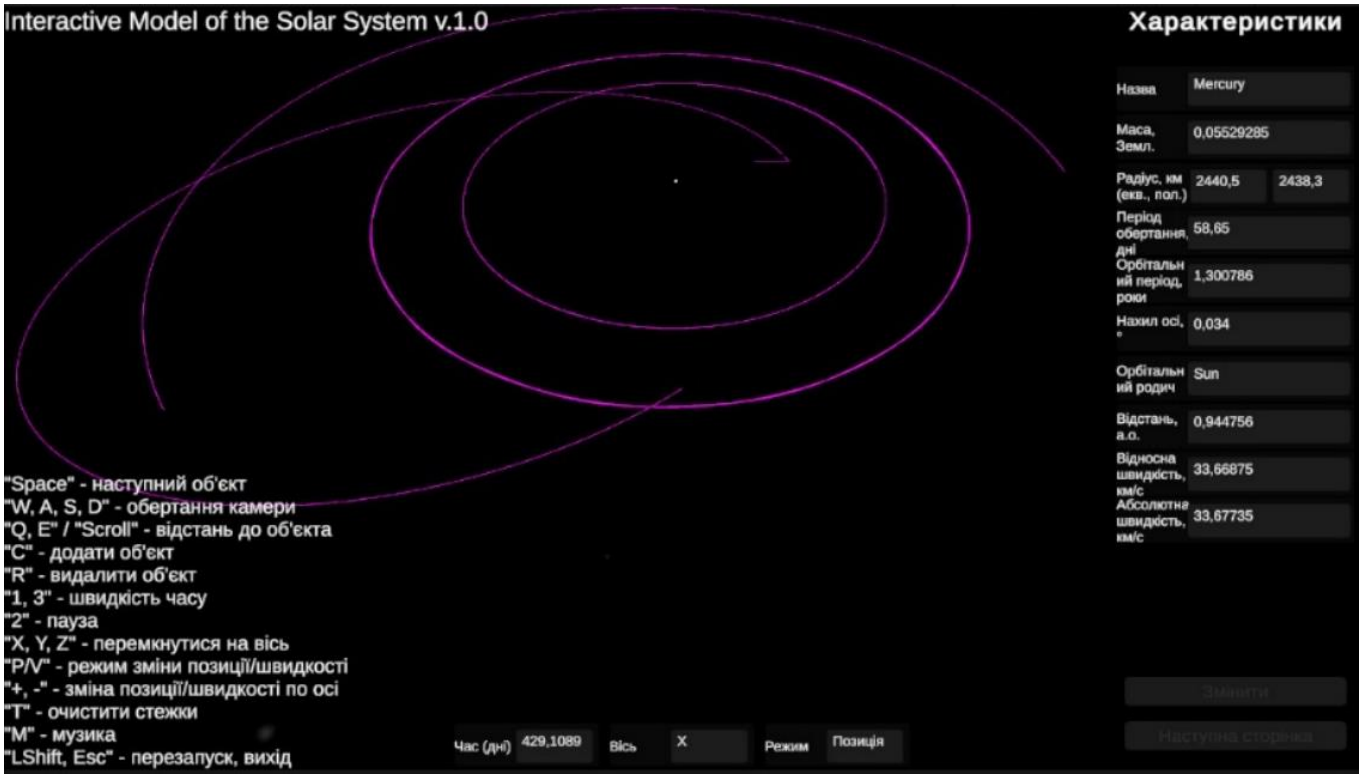

Рис. Д.7. Наслідок від переміщення Меркурія біля перигелію подалі від Сонця

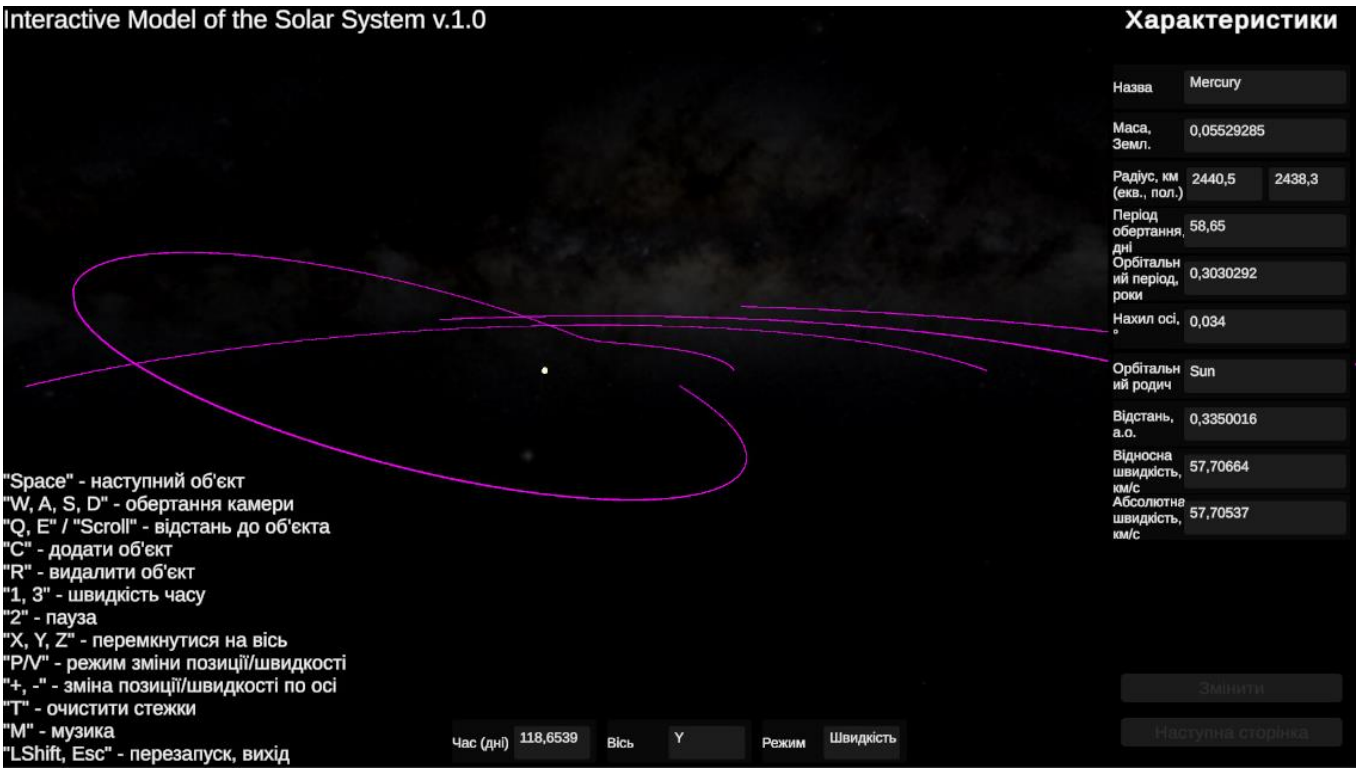

Рис. Д.8. Додавання Меркурію швидкості по осі Oy