МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«___» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
СТУПЕНЯ «МАГІСТР»

Тема: Програмний засіб візуалізації руху об'єктів при розробці *3D* симуляторів

Виконавець: Владислав МІКУЛЬСЬКИЙ

Керівник: Олена НЕЧИПОРУК

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«_____»_____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Мікульського Владислава Владиславовича

1.Тема роботи: «Програмний засіб візуалізації руху об'єктів при розробці 3*D* симуляторів»

затверджена наказом ректора від «28» серпня 2023 року № 1494 /ст.

2. Термін виконання роботи: з 02.10.2023 до 31.12.2023

3. Вихідні дані до проєкту (роботи): постановка задачі до виконання роботи, середовище розробки *Unity.*

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) сучасні методи і засоби 3*D*-візуалізації, що використовуються при впровадженні систем дистанційного навчання;
2) принципи розробки проєкту на *Unity* 3*D*;
3) Розробка програмного модуля імітації руху об'єктів в 3*D* симуляторах

5. Перелік обов'язкового графічного матеріалу:

1) діаграма класів програмного модуля;
2) модель збереження даних про 3*D*-об'єкт в системі навчання;
3) компоненти програмного модуля;
4) основні вікна розробленого програмного модуля;
5) схема алгоритму руху об'єктів.

6. Календарний план-графік

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою дипломного проєкту та аналіз існуючих систем	02.10.2023-12.10.2023	
2	Зробити вибір компонентів програмного модуля	13.10.2023-29.10.2023	
3	Розробити структуру програмного модуля	30.10.2023-06.11.2023	
4	Розробити програмні засоби для імітації руху об'єктів в 3*D*	07.10.2023-14.11.2023	
5	Провести налаштування програмних засобів на сервері	15.10.2023-26.11.2023	
6	Написати пояснювальну записку	27.10.2023-11.12.2023	
7	Підготувати презентацію	12.12.2023-18.12.2023	
8	Оформити супрводжувальну документацію	19.12.2023-23.12.2023	

7. Дата видачі завдання «02» жовтня 2023 р.

Керівник кваліфікаційної роботи_____Олена НЕЧИПОРУК_____
(підпис)

Завдання прийняв до виконання_____Владислав МІКУЛЬСЬКИЙ___
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний засіб візуалізації руху об'єктів при розробці 3*D* симуляторів» складається з: 97 с., 22 рис., 2 таблиці, 30 літературних джерел, 1 додаток.

3*D* СИМУЛЯТОР, ОБ’ЄМНІ ОБ’ЄКТИ, СИСТЕМИ ПРОЄКТУВАННЯ

Об’єкт дослідження – процес візуалізації руху об'єктів в 3*D* симуляторі.

Предмет дослідження – програмний засіб візуалізації руху об'єктів при розробці 3*D* симуляторів.

Мета роботи – розробити програмний засіб візуалізації руху об'єктів для використання в 3*D* симуляторі.

Результатом виконання кваліфікаційної роботи є закінченим програмним продуктом, що забезпечує візуалізацію руху об'єктів в 3*D* симуляторах.

Апробація результатів була проведена на *XVII* міжнародній науково-практичній конференції “*System analysis and intelligent systems for management*” ***(м.*** Анкара, Туреччина, 02-05 травня 2023 р.), міжнародній науковій конференції “Інтелектуальні технології лінгвістичного аналізу”, науковій практичній конференції “Сучасні тенденції розвитку системного програмування” (м. Київ, 23-24 грудня 2023 р.).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API – *Application Programming Interface*

UI – *User Interface*

UX – *User Experience*

ЕСН – електронна система навчання

КТ – комп’ютерний тренажер

ТЗН – технічий засіб навчання

ВСТУП

Актуальність. На сучасному етапі розвитку суспільства та технологій, освіта та навчання претендують на роль ключових чинників успіху в кожній галузі діяльності. Завдяки постійному зростанню обсягу інформації та швидкості технологічного розвитку, люди вимушені навчатися та перепідготовлюватися протягом усього життя. У такому контексті особливої актуальності набуває використання віртуальних тренажерів та симуляцій у професійній освіті та підготовці, які забезпечують інтерактивне, реалістичне та ефективне навчання.

Сучасний світ характеризується стрімким розвитком технологій, постійними змінами в різних галузях, високою конкуренцією на ринку праці та потребою у постійному навчанні. При цьому, зростає вимога до якості підготовки фахівців, які повинні бути готовими до вирішення складних завдань та приймати обґрунтовані рішення. Один із способів підготовки таких фахівців - використання віртуальних тренажерів та симуляцій.

Важливою проблемою, яка виникає у контексті використання віртуальних тренажерів, є їхній ефективний розробник, впровадження та використання у навчальних та професійних цілях. Розробка якісних та реалістичних віртуальних тренажерів вимагає багато ресурсів та експертизи, а їх ефективне використання в навчанні потребує врахування психологічних, педагогічних та технічних аспектів.

З іншого боку, велика частина сучасного навчання і підготовки проходить в онлайн-режимі, дистанційно, що ставить під загрозу можливість здобуття практичного досвіду та вмінь. Віртуальні тренажери можуть вирішувати цю проблему, надаючи можливість отримати практичний досвід без присутності у фізичному просторі.

Актуальність проблеми полягає в необхідності розробки та впровадження віртуальних тренажерів у професійну освіту та навчання, а також в дослідженні їхньої ефективності та визначенні кращих практик використання.

Візуалізація руху об'єктів в 3*D* симуляторах є важливою складовою для створення реалістичних віртуальних середовищ. Важливість даної теми пояснюється тим, що віртуальні симуляції широко використовуються у великій кількості галузей:

– у науці (допомагають у дослідженнях, які важко або небезпечно проводити в реальному світі);

– у авіаційній промисловості (для навчання пілотів та тестування літаків у віртуальних умовах);

– у військовій справі (для симуляції проведення воєнних операцій тощо);

– у медицині (для тренування хірургів та створення точних медичних симуляцій);

– у торгівлі (для показу об’єктів нерухомості, які ще не збудовані, для представлення товарів з різних ракурсів в режимі онлайн).

Мета роботи – розробити програмний засіб візуалізації руху об'єктів для використання в 3*D* симуляторі.

Об’єкт дослідження – процес візуалізації руху об'єктів в 3*D* симуляторі.

Предмет дослідження – програмний засіб візуалізації руху об'єктів при розробці 3*D* симуляторів.

Завдання дослідження:

1) проаналізувати сучасні тенденції у наукових дослідженнях щодо візуалізації в 3*D* симуляторах;

2) модифікувати алгоритми фізичної симуляції шляхом впровадження моделювання руху об'єктів з різними фізичними властивостями та врахування взаємодії об'єктів зі середовищем (зштовхування, руйнування та впливи на поверхні);

3) покращити наявні графічні двигуни шляхом впровадження нових методів реалізації ефектів (освітлення, тіні, рух води тощо);

4) розробити програмне забезпечення для підтримки навчального процесу, яке буде враховувати поведінку об'єктів з точки зору взаємодії з учнями та іншими об'єктами у симуляції.

Наукова новизна в поставлених задачах:

1) аналіз сучасних тенденцій у наукових дослідженнях щодо візуалізації в 3*D* симуляторах дає можливість визначити нові тенденції у візуалізації, які можуть бути корисними для подальших досліджень;

2) модифікація алгоритмів фізичної симуляції передбачає розробку нових алгоритмів моделювання руху об'єктів з різними фізичними властивостями та врахування взаємодії об'єктів зі середовищем, що включає в себе інноваційні підходи до обчислювання фізичних взаємодій, які не були добре вивченими в минулому;

3) розробка програмного забезпечення для підтримки навчального процесу поєднує в собі високоякісну візуалізацію руху об'єктів і систему навчання, що є новаторським підходом до покращення навчального процесу та може підвищити доступність навчальних матеріалів.

Апробація результатів була проведена на *XVII* міжнародній науково-практичній конференції "*System analysis and intelligent systems for management*" ***(м.*** Анкара, Туреччина, 02-05 травня 2023 р.), міжнародній науковій конференції "Інтелектуальні технології лінгвістичного аналізу", науковій практичній конференції "Сучасні тенденції розвитку системного програмування" (м. Київ, 23-24 грудня 2023 р.).

За результатами наукових конференцій опубліковано тези доповідей:

1. Нечипорук О.П., Мікульський В.В. Автоматизація тестування при візуалізації руху об'єктів в 3*d* симуляторах. – *System analysis and intelligent systems for management*: *proceedings of the XVII International Scientific and Practical Conference* (*Ankara, Turkey, May* 02-05, 2023). *Ankara,* 2023. *P*. 458-466.

2. Мікульський В.В. Формалізація даних про об'єкти в 3*d* симуляторах // Тези доповідей міжн. наук.-техн. конф. "Інтелектуальні технології лінгвістичного аналізу" (м. Київ, 24-25 жовтня 2023 р.) – К.: НАУ, 2023. – С. 67.

3. Мікульський В.В. Тестування зіткнень між об'єктами в 3*D* симуляторі // Тези доповідей наук.-практ. конф. "Сучасні тенденції розвитку системного програмування" (23-24 листопада 2023 р.). – К.: НАУ, 2023. – С. 43-44.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ ВІЗУАЛІЗАЦІЇ РУХУ В 3*D* СИМУЛЯТОРАХ

1.1. Основні поняття та терміни

3*D*-візуалізація - це технологія, що дозволяє створювати тривимірні візуальні представлення об'єктів та середовищ на двовимірних дисплеях. Основні принципи 3*D*-візуалізації включають:

1. Моделювання: Це процес створення геометричних моделей об'єктів та середовищ, які будуть відображатися у 3*D*-просторі. Моделювання може включати в себе створення форми об'єктів, їх текстур, кольорів та фізичних властивостей.
2. Рендеринг: Рендеринг - це процес обчислення зображення об'єктів і їх подання на екрані. Він враховує освітлення, тіні, кольори та текстури для створення реалістичного відображення.
3. Інтерактивність: Однією з ключових особливостей 3*D*-візуалізації є можливість взаємодії користувача з віртуальним середовищем. Це дозволяє користувачам маніпулювати об'єктами, досліджувати середовище та взаємодіяти з ним.
4. Оптимізація: Врахування продуктивності є важливою частиною 3*D*-візуалізації. Оптимізація забезпечує плавну роботу системи та мінімізує витрати ресурсів.

3*D*-візуалізація відкриває безліч можливостей для покращення дистанційного навчання:

1. Віртуальні лабораторії: Створення віртуальних лабораторій дозволяє студентам виконувати експерименти та спостерігати їх результати, навіть не покидаючи дому.
2. Медична освіта: Візуалізація анатомії, процедур та операцій допомагає медичним студентам отримувати практичні навички та знання.

3. Архітектурна освіта: Моделювання будівель та архітектурних об'єктів для студентів архітектурних спеціальностей.

4. Інтерактивні навчальні ігри: Створення навчальних ігор, що розвивають креативні та логічні навички.

5. Симулятори для навчання водінню та авіації: Тренування водіїв та пілотів у віртуальних умовах з використанням 3*D*-симуляцій.

6. Навчання мистецтву: Візуалізація мистецьких творів та створення віртуальних музеїв для навчання мистецтвознавства.

Переваги 3*D*-візуалізації включають:

- Збільшена залученість та інтерес студентів.
- Можливість доступу до віртуальних навчальних ресурсів з будь-якого місця.
- Покращена уявна сприйняття складних концепцій.

Недоліки 3*D*-візуалізації включають:

- Вимоги до технічного обладнання та програмного забезпечення.
- Потребу в тривалому процесі створення віртуальних моделей.
- Можливість відчуття дистанції між викладачем та студентами.

Дистанційне навчання з кожним роком набуває все більшої важливості для сучасних вузів. Ця тенденція є відображенням глобалізаційних змін у сфері вищої освіти та освітніх послуг. Глобалізація вищої освіти полягає у тому, що вузи та навчальні заклади розробляють та пропонують свої освітні програми та технології навчання, доступ до яких можливий з будь-якої точки світу. Це створює конкурентну середовище, в якому вузи змушені змагатися за студентів та партнерство з іншими установами.

Зменшення бар'єрів завдяки глобалізації

Завдяки глобалізації освіти, студенти мають доступ до широкого спектру освітніх ресурсів і можуть обирати навчальні програми з усього світу. Це зменшує географічні обмеження і відкриває можливість здобувати якісну освіту, навіть якщо студенти не можуть фізично перебувати в іншій країні.

Однією з важливих переваг дистанційного навчання є можливість отримувати якісну освіту без потреби переїзду або великих фінансових витрат на проживання в іншому місті або країні. Це особливо актуально для студентів, які мають сімейні або професійні зобов'язання, а також для тих, хто мешкає в регіонах з обмеженим доступом до вищої освіти.

Виклики для вищої освіти

Проте ця глобалізація вищої освіти ставить перед вузами ряд викликів. По-перше, конкуренція вищих навчальних закладів за студентів стає значною жорсткою. Студенти мають можливість обирати серед великої кількості вузів і програм, і ті, які надають найкращі можливості для дистанційного навчання, зазвичай здобувають перевагу.

Другий виклик полягає в тому, що вузи повинні забезпечити високий стандарт освіти у відповідності до міжнародних норм та очікувань студентів. Це означає, що вони повинні інтегрувати сучасні методи навчання та технології, зокрема 3*D*-візуалізацію, у свої освітні програми.

3*D*-візуалізація як відповідь на виклики

3*D*-візуалізація виявляється ключовим інструментом для вузів у розв'язанні цих викликів. Вона дозволяє створювати інтерактивні та захопливі навчальні матеріали, які сприяють залученню студентів та поліпшенню якості освіти. Вузи можуть створювати власні віртуальні лабораторії, симуляції та інші освітні ресурси, які допомагають студентам краще розуміти складні концепції та процеси.

Таким чином, впровадження 3*D*-візуалізації в дистанційному навчанні допомагає вузам залишатися конкурентоспроможними на глобальному ринку освітніх послуг і забезпечувати студентам доступ до сучасної та якісної освіти незалежно від їхнього місця проживання. У наступних розділах роботи ми розглянемо, як саме використовується 3*D*-візуалізація в дистанційному навчанні та які її переваги та недоліки в цьому контексті.

Швидкий розвиток комп'ютерних технологій, а також широке поширення персональних комп'ютерів в кінці *XX* століття серед населення призвело до появи комп'ютерної мультимедійної моделі дистанційного навчання. Цей етап розвитку

комп'ютерної технології сприяв створенню нових можливостей у сфері освіти та навчання. Ось детальний розгляд цієї еволюції:

Зростання обчислювальної потужності

Комп'ютери почали ставати доступними для широкого кола користувачів, а їх обчислювальна потужність значно зросла. Це дозволило використовувати комп'ютери для більш складних завдань, включаючи створення та відтворення великого обсягу мультимедійного контенту, такого як графіка, відео та аудіо.

Завдяки інтернету

Разом із зростанням обчислювальної потужності почало активно розвиватися Інтернет-пристосування, що відкрило нові можливості для дистанційного навчання. Мережа стала платформою для обміну інформацією та навчальним контентом між викладачами та студентами навіть на великі відстані.

Мультимедійна модель навчання

Поєднання обчислювальної потужності та доступу до Інтернету сприяло створенню комп'ютерної мультимедійної моделі дистанційного навчання. Ця модель використовує різні мультимедійні елементи, такі як відеоуроки, анімація, віртуальні лабораторії та інтерактивні симуляції, для подачі навчального матеріалу.

Переваги комп'ютерної мультимедійної моделі

- Доступність: Мультимедійна модель дозволяє студентам отримувати якісну освіту у будь-якому місці та в будь-який час, маючи лише доступ до комп'ютера та Інтернету.
- Залученість студентів: Використання мультимедійних елементів робить навчальний процес більш цікавим і захоплюючим для студентів. Вони можуть більше запам'ятовувати та краще розуміти складні концепції завдяки візуальному та інтерактивному навчанню.
- Можливість індивідуалізації: Мультимедійна модель дозволяє студентам вибирати темп та послідовність навчання, що найкраще відповідає їхнім потребам і рівню підготовки.

Виклики та обмеження

• Необхідність інфраструктури: Для успішного впровадження мультимедійного навчання потрібна належна комп'ютерна інфраструктура та доступ до швидкого Інтернету, що може бути недоступним у деяких регіонах.

• Потреба у зміні підходів до навчання: Викладачам та студентам доводиться адаптувати свої методи та підходи до нового формату навчання, що може бути викликом для деяких.

Завдяки швидкому розвитку комп'ютерних технологій та появі Інтернету комп'ютерна мультимедійна модель стала ключовим інструментом для дистанційного навчання. Вона надає можливість навчати та навчатися в будь-який час та в будь-якому місці, використовуючи інтерактивні та візуальні методи. Це сприяє підвищенню доступності та якості освіти, що є важливим для глобалізованого освітнього середовища.

Візуалізація руху об'єктів в 3*D* симуляторах є складним процесом, який включає в себе ряд основних понять та термінів. Давайте розглянемо детальніше найважливіші з них:

1. Віртуальний об'єкт (*Virtual Object*): Це абстрактна або конкретна сутність, яка існує віртуально в 3*D* середовищі. Віртуальні об'єкти можуть бути представлені графічно та мати фізичні характеристики, такі як маса, розміри, та параметри руху.

2. Графічний об'єкт (*Graphics Object*): Це конкретний об'єкт, який відображається на екрані користувача. Він може бути створений із використанням графічних моделей та матеріалів і мати властивості, такі як текстури та освітлення.

3. Текстурна карта (*Texture Map*): Графічний об'єкт може мати текстурну карту, яка визначає зовнішній вигляд об'єкта. Текстури використовуються для накладання зображень на поверхні об'єктів, що робить їх більш реалістичними.

4. Рендеринг (*Rendering*): Це процес обчислення та створення зображення графічного об'єкта на екрані. Рендеринг включає в себе обчислення колірів, тіней, та освітлення об'єкта.

5. Модель руху (*Motion Model*): Це математична модель, яка визначає рух об'єкта відносно часу. Модель руху може бути лінійною або не лінійною і враховувати різні фізичні закони.

6. Іммерсія (*Immersion*): Іммерсія відноситься до ступеня відчуття реалістичності та вовлеченості користувача у віртуальне середовище. Якщо візуалізація руху об'єктів відбувається високоякісно та реалістично, іммерсія може бути вищою.

7. Трасування променів (*Ray Tracing*): Це метод рендерингу, який моделює рух світла у віртуальному середовищі. Він дозволяє отримувати дуже реалістичні зображення з реалістичними відбитками світла.

8. Освітлення (*Lighting*): Освітлення відповідає за те, як світло взаємодіє з графічними об'єктами та створює тіні, відбитки та ефекти.

9. Фізична модель (*Physics Model*): Це математична модель, яка описує фізичні властивості об'єктів, такі як маса, швидкість, інерція та динаміка.

10. Віртуальна реальність (*Virtual Reality - VR*): *VR* - це технологія, яка забезпечує користувача можливість взаємодіяти з віртуальними об'єктами та середовищем у режимі реального часу, використовуючи спеціальні пристрої, такі як шоломи та контролери.

11. Реальний час (*Real-Time*): Візуалізація руху об'єктів в реальному часі означає, що зображення обчислюється та оновлюється на екрані негайно під час взаємодії користувача з симуляцією.

Візуалізація руху об'єктів в 3*D* симуляторах відіграє ключову роль у створенні реалістичних та іммерсивних віртуальних середовищ. Розглянемо детальніше, які функції та завдання виконує візуалізація в симуляціях:

1. Подання інформації користувачу: Основною функцією візуалізації є представлення об'єктів та подій у віртуальному світі користувачу. Візуалізація допомагає перетворити дані та інформацію про рух об'єктів в зрозумілі та доступні для сприйняття візуальні образи.

2. Створення реалістичних сцен: Візуалізація дозволяє створювати сцени, які виглядають як реальні об'єкти та середовища. Це важливо для симуляцій, де реалістичність має велике значення, наприклад, у медицинському тренажері, користувач повинен бачити реалістичну модель людського тіла.

3. Поведінка та реакція об'єктів на впливи: Візуалізація допомагає відтворити рух об'єктів та їх взаємодію відповідно до заданих фізичних законів. Об'єкти в симуляції повинні реагувати на сили, гравітацію, столкнення та інші фактори, і візуалізація руху грає важливу роль у цьому процесі.

4. Взаємодія користувача: Візуалізація створює можливість для користувача взаємодіяти з віртуальними об'єктами та середовищем. Це може включати в себе переміщення об'єктів, обертання, зміну параметрів, і взагалі дозволяє користувачеві впливати на симуляцію.

5. Створення іммерсії: Візуалізація руху об'єктів має велике значення для створення високого рівня іммерсії у користувачів. Якщо об'єкти рухаються та виглядають реалістично, це допомагає користувачам відчути себе "зануреними" у віртуальному середовищі.

6. Відображення структури та взаємозв'язків: Візуалізація може допомогти відображати складні структури та взаємозв'язки між об'єктами. У науці та наукових симуляціях це може бути важливим для вивчення складних процесів.

7. Оцінка та аналіз: Візуалізація дозволяє спостерігати за рухом об'єктів, аналізувати їхню поведінку та виносити висновки. Наприклад, у науці візуалізація допомагає дослідникам аналізувати результати експериментів.

Узагальнюючи, візуалізація руху об'єктів в 3*D* симуляторах є важливим компонентом, що допомагає створити реалістичні та ефективні віртуальні середовища з високою іммерсією та можливістю взаємодії користувачів.

Методи візуалізації руху об'єктів в 3*D* симуляторах включають в себе різні технології та алгоритми, що дозволяють представити рух об'єктів у віртуальному просторі. Розглянемо детальніше основні методи візуалізації руху об'єктів:

1. Рейтрейсінг (*Ray Tracing*): Цей метод є одним з найбільш точних та реалістичних для візуалізації руху об'єктів. Він базується на симуляції променів світла, які відбиваються та ламаються при взаємодії з об'єктами. Рейтрейсінг відтворює ефекти відбиття, тіней, розсіювання світла та інші, що робить зображення дуже реалістичними, але вимагає значних обчислювальних ресурсів.

2. Рендеринг в реальному часі (*Real-Time Rendering*): Цей метод застосовується у відомих відеоіграх та інтерактивних симуляторах, де швидкість оновлення кадрів є критичною. Він використовує різні алгоритми, такі як растровий рендеринг та буфери глибини, для створення зображень в реальному часі. Реалізація вимагає оптимізації та компромісів у якості графіки.

3. Віртуальна реальність (*Virtual Reality - VR*): *VR* використовує шоломи та спеціальні контролери для візуалізації руху об'єктів та взаємодії з ними у реальному часі. Ця технологія дозволяє користувачам занурюватися у віртуальну симуляцію та спостерігати рух об'єктів з будь-якого кута зору.

4. Анімація (*Animation*): Анімація використовується для створення плавного та реалістичного руху об'єктів. Це може включати в себе ключові кадри, скелетні анімації, фізичну анімацію та інші техніки для симуляції руху.

5. Графічні моделі (*Graphics Models*): Використання графічних моделей дозволяє створювати об'єкти та їх текстури для візуалізації руху. Це може бути 3*D*-моделі, які використовуються для представлення об'єктів, або 2*D*-текстури для роботи з поверхнями.

6. Фізичні моделі (*Physics Models*): Ці моделі дозволяють симулювати фізичні закони та взаємодію об'єктів у віртуальному світі. Вони визначають, як об'єкти реагують на сили та впливи.

7. Освітлення (*Lighting*): Освітлення відтворює вплив світла на об'єкти та ств

1.2. Фізичні аспекти моделювання руху

Фізичні аспекти моделювання руху об'єктів в 3*D* симуляторах відіграють важливу роль у створенні реалістичних імітацій руху. Цей підрозділ розкриє ключові аспекти фізичних моделей, які використовуються для візуалізації руху об'єктів:

1. Маса та інерція (*Mass and Inertia*): Фізичні об'єкти мають масу та інерцію, що визначає, як вони реагують на сили та прискорення. Візуалізація руху повинна враховувати ці фізичні властивості, щоб об'єкти рухалися реалістично.

2. Сили та прискорення (*Forces and Acceleration*): У фізичних моделях відтворюються різні сили, такі як гравітація, тяга, тиск, аеродинаміка та інші. Об'єкти рухаються під впливом цих сил та прискорення, і це повинно бути відображено в їхньому русі.

3. Динаміка (*Dynamics*): Моделювання динаміки об'єктів включає в себе визначення, як змінюються їхні швидкості та позиції у часі. Використовуються рівняння руху, такі як рівняння Ньютона, для опису цих змін.

4. Тертя (*Friction*): Тертя визначає, як сила опору впливає на рух об'єктів на поверхнях. Це важливо для точного моделювання руху по різних типах підстилки.

5. Столкнення (*Collision*): Моделювання столкнень включає в себе розрахунок, як об'єкти взаємодіють при зіткненні. Це може вимагати визначення точок контакту, розрахунку сил столкнення та зміни швидкості після столкнення.

6. Системи частинок (*Particle Systems*): Для моделювання руху розсіюваних частинок, таких як дим, вода або пил, використовуються системи частинок, що базуються на фізичних принципах, таких як закони розсіювання.

7. Флюїдна динаміка (*Fluid Dynamics*): Для моделювання руху рідин та газів використовуються фізичні моделі флюїдної динаміки. Це важливо для симуляції таких явищ, як рух повітря, водяних хвиль та гідродинамічні ефекти.

8. Реакція на зовнішні впливи (*External Forces*): Фізичні об'єкти можуть піддаватися зовнішнім впливам, таким як вітер, тиск або магнітні поля. Моделювання цих впливів допомагає зрозуміти, як об'єкти реагують на навколишнє середовище. Наприклад, у симуляціях авіаційного обладнання важливо враховувати вплив атмосферних умов та вітру на рух літака. Точне моделювання зовнішніх впливів забезпечує реалістичну поведінку об'єктів в симуляціях.

Фізичні аспекти моделювання руху в 3*D* симуляторах вимагають математичних моделей та обчислювальних методів, щоб точно відтворити реальну фізику та забезпечити реалістичну візуалізацію руху об'єктів. Однак вони є важливими для досягнення високої реалістичності та якості в симуляціях.

1.3. Графічні двигуни та їх використання

Графічні двигуни є ключовою складовою для візуалізації руху об'єктів в 3*D* симуляторах. Вони відповідають за генерацію графіки, яка відображає рух об'єктів та створює віртуальне середовище для користувачів. Розглянемо детальніше графічні двигуни та їх використання в симуляціях:

1. Графічний двигун (*Graphics Engine*): Графічний двигун - це програмне забезпечення, яке відповідає за візуалізацію 3*D* об'єктів та сцен у реальному часі. Він обробляє дані про геометрію об'єктів, текстури, освітлення, тіні та інші ефекти для створення зображення.

2. Рендерінг (*Rendering*): Графічний двигун використовує техніки рендерінгу для перетворення 3*D* об'єктів у 2*D* зображення, які можна відобразити на екрані. Цей процес включає в себе визначення видимості об'єктів, обчислення освітлення, тіней та кольорів.

3. Шейдери (*Shaders*): Шейдери - це програми, які виконуються на графічних картках та визначають, як об'єкти мають відображатися. Шейдери контролюють колір, текстуру та інші атрибути об'єктів, дозволяючи створити реалістичну графіку.

4. Освітлення (*Lighting*): Графічні двигуни підтримують різні моделі освітлення, такі як фізично засноване освітлення (*PBR*), точкове освітлення, сонячне освітлення тощо. Це дозволяє створювати реалістичні та захоплюючі ефекти освітлення на об'єктах.

5. Тіні (*Shadows*): Для створення реалістичних тіней графічні двигуни використовують алгоритми, такі як тінькартки, тіньові об'єкти та методи створення м'яких тіней. Це важливо для підвищення реалізму сцен.

6. Оптимізація (*Optimization*): Графічні двигуни включають в себе методи оптимізації для забезпечення плавної роботи симуляцій, навіть на потужних сценах з великою кількістю об'єктів. Це може включати в себе використання різних рівнів деталізації, оптимізовані методи рендерінгу та зменшення обсягу даних.

7. Віртуальна реальність (*VR*) та Розширена реальність (*AR*): Графічні двигуни дозволяють інтегрувати віртуальну реальність та розширену реальність у симуляції.

Вони підтримують генерацію 3*D* зображень для віртуальних шоломів та пристроїв *AR*.

8. Реалістична фізика: Графічні двигуни дозволяють інтегрувати фізичні обчислення для об'єктів та їх руху, забезпечуючи точне моделювання фізичних властивостей.

9. Спеціальні ефекти: Графічні двигуни підтримують створення спеціальних ефектів, таких як частинковий облік, водяні ефекти, вогонь, дим, блискавки тощо, що додає реалізму сценам.

Графічні двигуни (або рушії) є основними компонентами для відображення руху об'єктів в 3*D* симуляторах. Вони відповідають за обчислення та рендеринг 3*D* сцен, включаючи об'єкти, освітлення, тіні та текстури. Розглянемо детальніше графічні двигуни та їх роль у візуалізації руху:

1. *OpenGL*: *OpenGL* є відкритим стандартом для програмування графіки. Він надає низькорівневі функції для відображення 3*D* об'єктів та взаємодії з графічним апаратом комп'ютера. *OpenGL* широко використовується у відомих графічних двигунах та симуляторах для створення реалістичних візуальних ефектів.

2. *DirectX*: *DirectX* - це набір *API* для розробки ігор та програм з графікою на платформах *Windows*. Він включає в себе *DirectX* 11 та *DirectX* 12, які забезпечують високу продуктивність та підтримку сучасних графічних можливостей. *DirectX* використовується у багатьох відомих іграх та симуляторах.

3. *Unity*: *Unity* - це інтегроване середовище розробки для створення ігор та симуляторів. Він включає в себе власний графічний двигун, який дозволяє легко створювати 3*D* сцени та анімацію. *Unity* широко використовується для розробки симуляторів руху, включаючи автомобільні та літаки симулятори.

4. *Unreal Engine*: *Unreal Engine* - це інший популярний інтегрований двигун для розробки ігор та симуляцій. Він володіє потужними графічними можливостями, включаючи високоякісний рендеринг, фізично засноване освітлення та велику кількість готових ресурсів для створення реалістичних сцен.

5. *CryEngine*: *CryEngine* від *Crytek* є ще одним графічним двигуном, який використовується для розробки ігор та симуляторів. Він славиться своєю

високоякісною графікою та ефективністю, а також можливістю створення різноманітних погодних ефектів та ландшафтів.

6. *Custom Engines*: Деякі симулятори руху розробляють власні графічні двигуни, настроюючи їх для конкретних потреб симуляції. Це дозволяє досягти максимальної продуктивності та контролю над відображенням.

Графічні двигуни допомагають відтворити рух об'єктів у 3*D* симуляторах, забезпечуючи високоякісну та реалістичну графіку. Вони також підтримують інші функції, такі як обробка взаємодії з користувачем, анімація, фізика та оптимізація, що дозволяє створювати іммерсивні симуляції руху для різних цілей, від навчання до розваги.

Таблиця 1.1.

Графічні двигуни та їх порівняння з перевагами і недоліками

Графічний Двигун	Переваги	Недоліки
OpenGL	- Відкритий стандарт, що підтримується багатьма платформами.	- Низькорівневий *API,* вимагає більше коду для налаштування.
DirectX	- Розроблений для *Windows,* що дозволяє використовувати всі графічні можливості платформи.	- Ексклюзивний для *Windows,* що обмежує переносимість.
Unity	- Інтегроване середовище розробки з великою спільнотою та документацією.	- Може бути менше підходящим для великих ігор або складних симуляцій.
Unreal Engine	- Потужний інструмент для створення реалістичних сцен і ефектів.	- Вимагає більше ресурсів для розробки і використання.
CryEngine	- Високоякісна графіка та погодні ефекти.	- Вимагає докладної настройки та оптимізації для досягнення високої продуктивності.
Власний двигун	- Повний контроль над функціональністю та оптимізацією.	- Вимагає великих зусиль для розробки та підтримки.

OpenGL:

– Переваги:

– Відкритий стандарт, що робить його доступним для багатьох платформ.

– Забезпечує можливість прямого взаємодії з графічним апаратом комп'ютера.

– Недоліки:

– Низькорівневий *API,* вимагає більше коду для налаштування.

– Переносимість між різними платформами може бути проблемою.

DirectX:

– Переваги:

– Розроблений для *Windows,* що дозволяє використовувати всі графічні можливості платформи.

– Велика кількість бібліотек і інструментів для розробки ігор.

– Недоліки:

– Ексклюзивний для *Windows,* що обмежує переносимість на інші платформи.

– Потребує знань мови програмування *C++*.

Unity:

– Переваги:

– Інтегроване середовище розробки з великою спільнотою і підтримкою.

– Можливість створювати симулятори та ігри без значних знань програмування.

– Недоліки:

– Може бути менше підходящим для великих проектів з високими вимогами до продуктивності.

Unreal Engine:

– Переваги:

– Потужний інструмент для створення візуально захоплюючих сцен і ефектів.

– Велика кількість готових ресурсів і плагінів.

– Недоліки:

– Вимагає більше ресурсів для розробки і використання.

CryEngine:

– Переваги:

– Високоякісна графіка та погодні ефекти.

– Добра підтримка фізики.

– Недоліки:

– Вимагає докладної настройки та оптимізації для досягнення високої продуктивності.

Власний двигун:

– Переваги:

– Повний контроль над функціональністю та оптимізацією.

– Можливість розробляти унікальні рішення для конкретних завдань.

– Недоліки:

– Вимагає великих зусиль для розробки та підтримки.

– Може бути витратним з точки зору часу і ресурсів.

Моделювання графічних об'єктів є ключовою складовою візуалізації руху об'єктів в 3*D* симуляторах. Цей процес включає в себе створення та управління 3*D* моделями об'єктів, їх текстурами, анімацією та освітленням. Розглянемо детальніше підрозділ моделювання графічних об'єктів:

1. 3*D* Моделювання: Моделювання починається зі створення 3*D* об'єктів та їх геометричних форм. Це може бути виконано за допомогою спеціалізованих програм для моделювання, таких як *Blender, Maya,* 3*ds Max* або *Cinema* 4*D*. Розробники створюють меші (триангульні сітки), які описують форму об'єктів.

2. Текстури та Маппінг: Після створення мешів об'єктів, їм можуть бути надані текстури для надання реалістичного вигляду. Це включає в себе процес текстурного маппінгу, де текстури накладаються на меші так, щоб вони відображали колір, текстури та інші деталі об'єкта.

3. Анімація: Об'єкти можуть бути анімовані, щоб відтворити їх рух. Анімація може бути створена за допомогою кадрів (*keyframes*) або скриптів, що контролюють параметри об'єкта, такі як позиція, обертання, розмір тощо. Для складних анімацій можуть використовуватися скелетні анімації та морфінг.

4. Фізична Модель: Для реалістичного моделювання руху об'єктів в симуляторах часто використовують фізичні моделі. Це включає в себе моделювання фізичних властивостей об'єктів, таких як маса, інерція, опір повітря, гравітація і реакція на сили.

5. Освітлення: Освітлення графічних об'єктів важливо для створення реалістичних сцен. Графічні двигуни підтримують різні моделі освітлення, такі як фізично засноване освітлення (*PBR*), точкове освітлення, сонячне освітлення тощо.

6. Складові Руху: Для симуляції руху об'єктів додатково можуть бути враховані параметри, такі як швидкість, прискорення, обертання та траєкторія. Ці параметри обчислюються відповідно до фізичних законів.

3*D* моделювання є критично важливою частиною створення симуляторів, оскільки від цього процесу залежить вигляд та поведінка об'єктів у віртуальному просторі. Під час 3*D* моделювання для симуляторів, особливу увагу слід звертати наступним аспектам:

1. Геометрия об'єктів: Геометрична форма об'єкта визначає його зовнішній вигляд. Для реалістичної візуалізації руху об'єктів важливо створювати точні та докладні 3*D* моделі. Переконайтеся, що геометрія моделі відповідає реальному об'єкту, з урахуванням його розмірів і форми.

2. Текстури: Додавання текстур допомагає створити більш реалістичний вигляд об'єкта. Правильно вибрані текстури можуть відобразити деталі, колір та матеріал об'єкта. У симуляторах важливо використовувати текстури, які відображають стан об'єкта під час руху.

3. Анімація: Якщо об'єкти в симуляторі рухаються або взаємодіють між собою, анімація грає важливу роль. Для реалістичної відтворення руху об'єктів, створюйте анімації, які відповідають фізичним законам і показують поведінку об'єктів у різних сценаріях.

4. Фізична модель: У симуляціях руху, особливо в авіаційних, автомобільних або морських симуляторах, важливо враховувати фізичні властивості об'єктів. Це включає в себе масу, інерцію, опір повітря та інші параметри, які впливають на рух.

5. Деталізація: Деталізація моделей грає важливу роль у підвищенні реалістичності. Звертайте увагу на дрібні деталі, такі як текстурні деталі, бамп-мапи, нормальні картки та інші ефекти, які додають об'єктам глибину та реалізм.

6. Оптимізація: Оптимізація графічних моделей важлива для забезпечення плавної роботи симулятора, особливо в великих та складних проектах.

Використовуйте техніки, такі як рівні деталізації (*LOD*), щоб зменшити навантаження на обчислювальні ресурси без втрати якості.

7. Інтерактивність: Враховуйте можливість інтерактивності об'єктів у симуляторі. Наприклад, об'єкти можуть взаємодіяти з користувачем або відповідати на зовнішні впливи.

8. Спеціальні ефекти: Деякі симулятори можуть вимагати використання спеціальних ефектів, таких як часткова прозорість, вогнені ефекти, частки та інші для підвищення реалістичності та візуального ефекту.

Загалом, при створенні симуляторів руху важливо збалансувати деталізацію графічних об'єктів з продуктивністю та реалістичністю, щоб забезпечити іммерсивний досвід користувача.

Аналіз робіт [13] та [14] дозволив схематично представити цю модель для шкіл (рис. 1.1) та закладів вищої освіти (розширений варіант) (рис. 1.2).

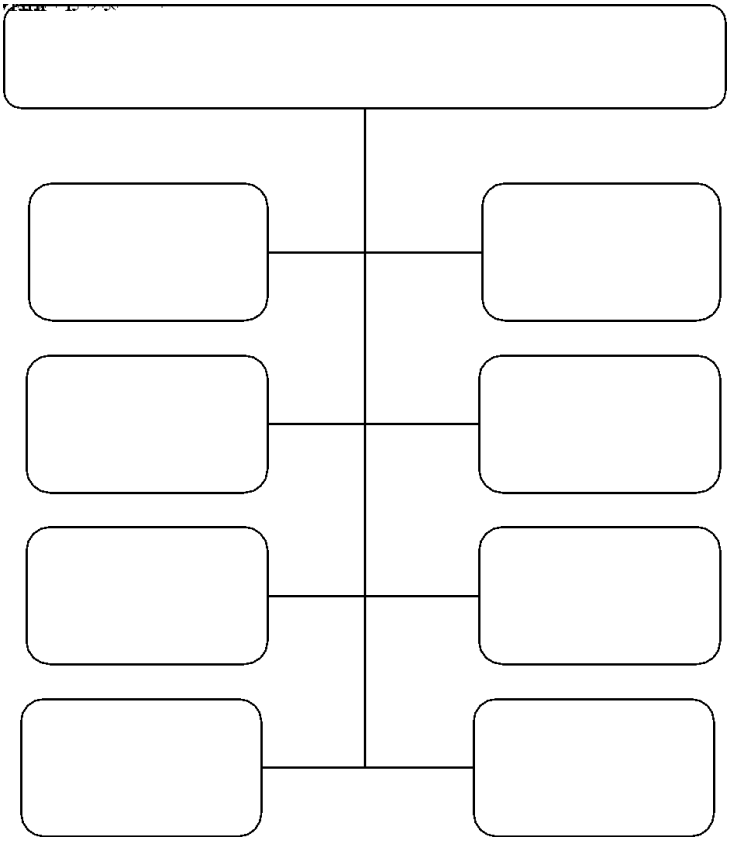

Рис. 1.1. Модель інтеграції очних і дистанційних форм навчання для шкіл

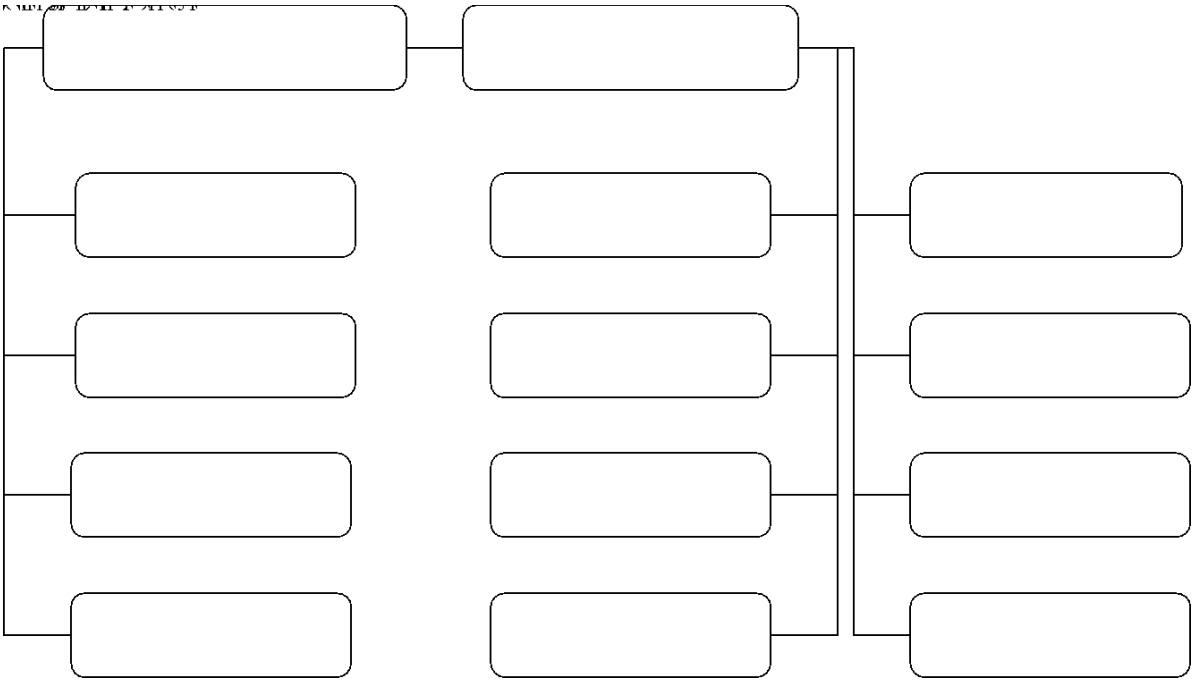

Рис. 1.2. Модель інтеграції очних і дистанційних форм навчання для закладів вищої освіти

У цьому випадку створюються спеціальні, автономні курси дистанційного навчання, тобто з окремих навчальних предметів, розділів або тем програми або віртуальні школи, кафедри, університети (рис. 1.3).

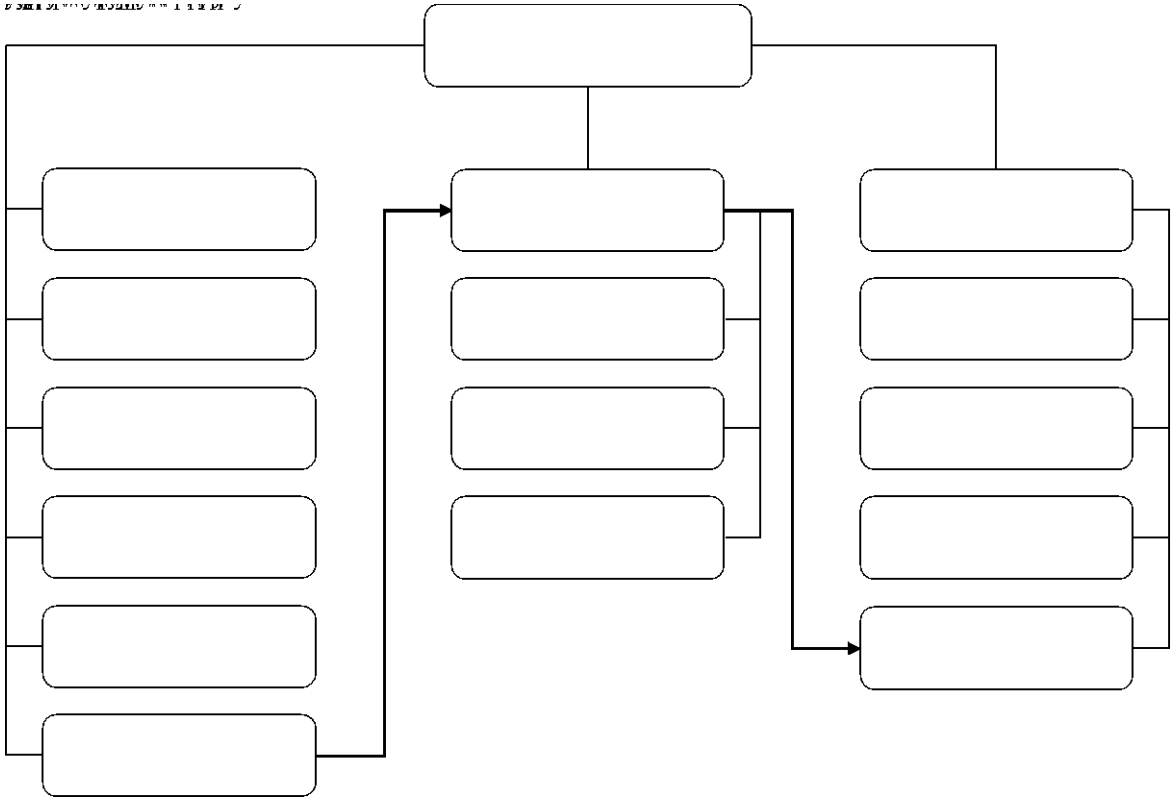

Рис. 1.3. Модель мережевого курсу у системі дистанційного навчання

Автономні курси більше призначені для оволодіння окремим навчальним предметом, поглиблення знань з цього предмету або навпаки, ліквідації прогалин в знаннях. Дана модель може бути представлена наступним чином. Модель інтерактивного телебачення має свої переваги, але вимагає серйозних витрат та технічних ресурсів для впровадження (рис. 1.4).

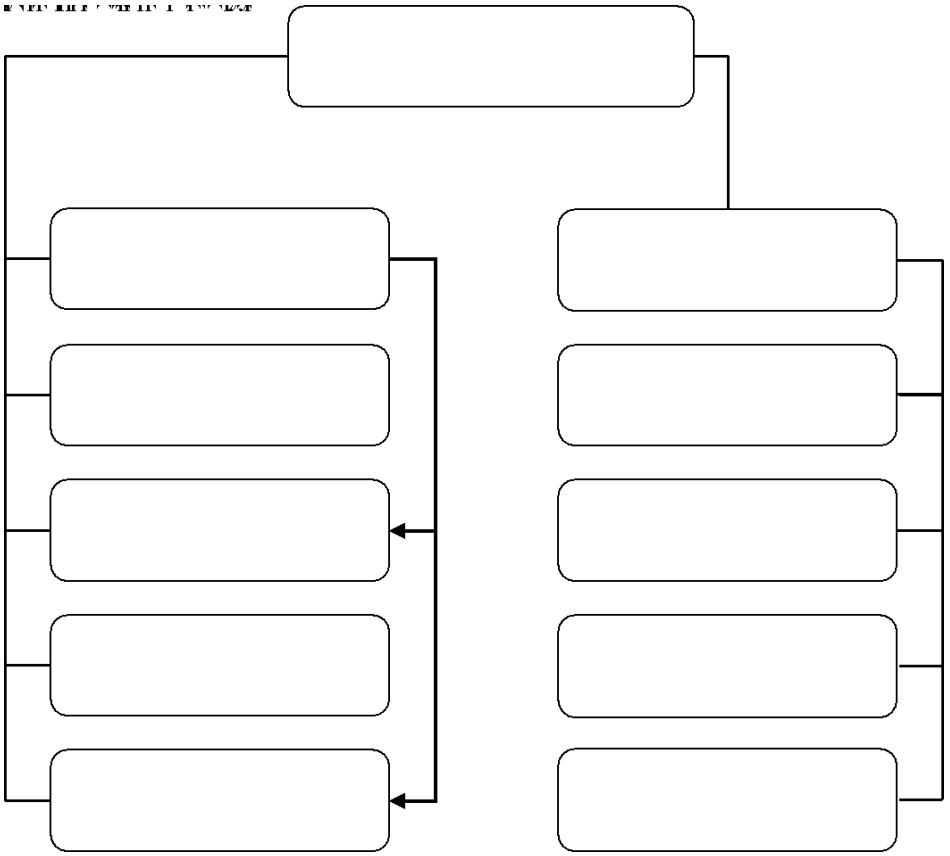

Рис. 1.4. Модель інтерактивного телебачення у системі дистанційного навчання

Підхід до організації процесу навчання в дистанційній освіті - це комплекс системних підходів, методів і стратегій, які використовуються для створення, управління і реалізації навчальних програм та курсів у віртуальних або дистанційних середовищах з метою забезпечення якісного навчання і навчального досвіду для студентів, викладачів і інших учасників освітнього процесу.

Організація процесу навчання в дистанційній освіті вимагає глибокого розуміння основних принципів, завдань, переваг і викликів цього типу освіти. До них відносяться:

1. Основні принципи дистанційної освіти:

• Доступність: Дистанційна освіта має бути доступною для широкого кола студентів, незалежно від їхнього місця проживання чи географічного розташування.

• Гнучкість: Система навчання повинна бути гнучкою, дозволяючи студентам обирати час і місце для вивчення матеріалів.

• Інтерактивність: Важливо створювати можливості для взаємодії між студентами та викладачами, навіть у віддаленому форматі.

• Якість: Дистанційне навчання повинно забезпечувати високу якість навчання та здобуття знань.

• Оцінка: Система оцінювання повинна бути об'єктивною та справедливою.

2. Технології і інфраструктура:

• Для успішної дистанційної освіти потрібна відповідна технічна інфраструктура. Це включає в себе комп'ютери, Інтернет-з'єднання, платформи для навчання, відеоконференції і інші технології.

3. Планування і дизайн курсу:

• Важливо створити добре організований навчальний план та курс, який враховує мети навчання, потреби студентів і методи навчання.

4. Активне взаємодія між учасниками:

• Взаємодія між студентами, викладачами та іншими учасниками навчання є важливим аспектом дистанційної освіти. Форуми, чати, відеоконференції та інші засоби спілкування сприяють обміну ідеями та досвідом.

5. Забезпечення мотивації та підтримки студентів:

• Дистанційне навчання може бути вимогливим та виснажливим. Важливо створити систему мотивації та підтримки для студентів, щоб вони залишалися мотивованими та успішними в навчанні.

6. Оцінка та звітування:

• Проведення оцінювання та надання звітів про успішність студентів є важливою частиною навчального процесу. Оцінка може бути здійснювана через тести, завдання, практичні роботи та інші методи.

7. Розвиток і вдосконалення:

• Дистанційна освіта постійно змінюється та розвивається разом із технологіями. Важливо вдосконалювати курси і методи навчання на основі отриманих результатів і зворотного зв'язку.

8. Співробітництво з іншими установами та організаціями:

• Установи і організації можуть співпрацювати для створення спільних навчальних програм та обміну ресурсами для підтримки дистанційної освіти.

Ці принципи і підходи до організації процесу навчання в дистанційній освіті є фундаментальними для забезпечення якісного навчання і розвитку цього сектору освіти. Вони допомагають вирішувати виклики та забезпечувати доступність та ефективність навчання у віддаленому форматі для всіх учасників освітнього процесу.

Ідея технологізації навчального процесу з'явилася на початку *XX* століття і виникала з віру в те, що тільки за допомогою нових технологій можна покращити ефективність освіти. Технологічний підхід до побудови навчання генетично пов'язаний з такими поняттями, як технічні засоби навчання (ТЗН), і тісно пов'язаний з розвитком технологій і їхнім впливом на освітній процес.

Технічні засоби навчання (ТЗН) включають в себе будь-які технології, прилади, системи або інструменти, що використовуються для полегшення або підтримки процесу навчання. Вони можуть бути фізичними, програмними, електронними або ж включати в себе комп'ютерне обладнання, мультимедійні засоби, інтерактивні платформи, відео- та аудіоінструменти, веб-додатки і багато інших.

Технічні засоби навчання можуть використовуватися для різних цілей і завдань в освітньому процесі, включаючи:

1. Полегшення доступу до інформації: Інтернет і електронні бібліотеки дозволяють студентам отримувати доступ до широкого обсягу навчальних матеріалів і ресурсів, незалежно від їхнього місця проживання.

2. Збільшення інтерактивності: Використання відеоконференцій, віртуальних класів і форумів сприяє активній взаємодії між студентами та викладачами, що поліпшує якість навчання.

3. Самостійне навчання: Електронні курси, навчальні платформи та інтерактивні навчальні матеріали дозволяють студентам вчитися самостійно та власним темпом.

4. Збільшення мотивації: Графіка, аудіо та відео можуть бути використані для створення цікавих та залучаючих навчальних матеріалів.

5. Оцінка та звітування: Електронні системи оцінювання та звітування спрощують процес оцінки студентів і надають змогу відстежувати їхній прогрес.

6. Гнучкість: За допомогою віртуальних платформ та інших інструментів можливо організовувати навчання в будь-який час та в будь-якому місці.

7. Співпраця та обмін досвідом: Інтернет дозволяє студентам та викладачам співпрацювати та обмінюватися досвідом з колегами з усього світу.

8. Збереження даних та ресурсів: Електронні системи зберігання даних дозволяють зберігати та архівувати навчальні матеріали та ресурси.

Технологічний підхід до навчання надає можливість розвивати нові методи навчання, залучаючи сучасні інструменти та ресурси. Він дозволяє створити більш доступні, ефективні та інтерактивні навчальні середовища для студентів у всьому світі.

Саме програмоване незалежне навчання - це підхід до освіти, який вважається однією з перспектив сучасної освіти. Цей підхід базується на ідеї, що студенти можуть самостійно вчитися та контролювати свій власний процес навчання, використовуючи різноманітні ресурси та інструменти, які доступні завдяки сучасним технологіям. Давайте розглянемо цей підхід більш детально.

Програмоване навчання - це метод навчання, який передбачає структурований підхід до вивчення матеріалу, де студенти працюють над завданнями та матеріалами відповідно до заздалегідь розробленого плану або програми. Основна ідея полягає в тому, щоб надати студентам можливість навчатися у своєму власному темпі та вирішувати, коли і як вони готові переходити до наступних розділів або завдань.

Основні характеристики програмованого навчання:

1. Самостійність студентів: Студенти мають можливість вибирати теми та завдання, які їх цікавлять, і працювати над ними у відповідності до свого графіку. Вони можуть обирати шлях навчання, який відповідає їхнім потребам та інтересам.

2. Самоконтроль: Програмоване навчання включає в себе механізми самоконтролю, де студенти оцінюють свій власний прогрес і роблять корекції у своїй роботі відповідно до отриманих результатів.

3. Індивідуалізація: Кожен студент може працювати на своєму рівні та відповідно до свого стилю навчання. Програмоване навчання дозволяє враховувати індивідуальні потреби кожного студента.

4. Використання різних ресурсів: Студенти можуть використовувати різні навчальні ресурси, такі як підручники, відеоуроки, веб-сайти, електронні курси тощо, для отримання інформації та вивчення матеріалу.

5. Адаптивність: Програмоване навчання може бути адаптоване до різних видів навчальних програм і цілей. Воно може бути використане як у формальних, так і в неформальних системах навчання.

6. Використання технологій: Сучасні технології, такі як Інтернет, мультимедіа, електронні платформи та інші інструменти, роблять програмоване навчання більш доступним і ефективним.

Саме цей режим навчання дозволяє студентам розвивати навички самостійності, саморегуляції та самоконтролю, які є важливими в сучасному світі. Студенти вчаться бути відповідальними за своє власне навчання та навчаються шукати і використовувати різні ресурси для досягнення своїх цілей.

Програмоване навчання також може бути ефективним інструментом для розвитку критичного мислення, аналітичних навичок та творчого мислення студентів. Воно дозволяє студентам досліджувати теми та працювати над проектами відповідно до своїх інтересів.

Загалом, програмоване навчання відкриває перед сучасною освітою нові можливості та дозволяє студентам більш ефективно вивчати матеріал, адаптувати навчання до своїх потреб та розвивати ключові навички, необхідні в сучасному світі.

Графічні двигуни (або рушії) є основними компонентами для відображення руху об'єктів в 3*D* симуляторах. Вони відповідають за обчислення та рендеринг 3*D* сцен, включаючи об'єкти, освітлення, тіні та текстури. Розглянемо детальніше графічні двигуни та їх роль у візуалізації руху:

1. *OpenGL*: *OpenGL* є відкритим стандартом для програмування графіки. Він надає низькорівневі функції для відображення 3*D* об'єктів та взаємодії з графічним апаратом комп'ютера. *OpenGL* широко використовується у відомих графічних двигунах та симуляторах для створення реалістичних візуальних ефектів.

2. *DirectX*: *DirectX* - це набір *API* для розробки ігор та програм з графікою на платформах *Windows*. Він включає в себе *DirectX* 11 та *DirectX* 12, які забезпечують високу продуктивність та підтримку сучасних графічних можливостей. *DirectX* використовується у багатьох відомих іграх та симуляторах.

3. *Unity*: *Unity* - це інтегроване середовище розробки для створення ігор та симуляторів. Він включає в себе власний графічний двигун, який дозволяє легко створювати 3*D* сцени та анімацію. *Unity* широко використовується для розробки симуляторів руху, включаючи автомобільні та літаки симулятори.

4. *Unreal Engine*: *Unreal Engine* - це інший популярний інтегрований двигун для розробки ігор та симуляцій. Він володіє потужними графічними можливостями, включаючи високоякісний рендеринг, фізично засноване освітлення та велику кількість готових ресурсів для створення реалістичних сцен.

5. *CryEngine*: *CryEngine* від *Crytek* є ще одним графічним двигуном, який використовується для розробки ігор та симуляторів. Він славиться своєю високоякісною графікою та ефективністю, а також можливістю створення різноманітних погодних ефектів та ландшафтів.

6. *Custom Engines*: Деякі симулятори руху розробляють власні графічні двигуни, настроюючи їх для конкретних потреб симуляції. Це дозволяє досягти максимальної продуктивності та контролю над відображенням.

Графічні двигуни допомагають відтворити рух об'єктів у 3*D* симуляторах, забезпечуючи високоякісну та реалістичну графіку. Вони також підтримують інші

функції, такі як обробка взаємодії з користувачем, анімація, фізика та оптимізація, що дозволяє створювати іммерсивні симуляції руху для різних цілей, від навчання до розваги.

1.4. Аналіз використання тренажерних систем в професійній освіті

Професійна освіта є важливою складовою підготовки фахівців у різних сферах індустрії та послуг. З часом змінюються вимоги до кваліфікації робочої сили, і сучасні тренажерні системи стають необхідним інструментом для навчання та підвищення кваліфікації майбутніх та досвідчених спеціалістів.

Роль тренажерів у професійній освіті:

1. Практичне навчання: Тренажери дозволяють студентам та спеціалістам набувати практичні навички та досвід без реального ризику. Наприклад, медичні тренажери дозволяють медичним студентам виконувати операції безпечно та безпечно, навіть до того, як вони працюють на пацієнтах.
2. Відтворення реалістичних умов: Тренажери можуть моделювати реальні умови та сценарії, з якими фахівці можуть зіткнутися у своїй роботі. Наприклад, авіаційні тренажери відтворюють польоти в різних погодних умовах та аварійних ситуаціях.
3. Підвищення ефективності навчання: Використання тренажерів може сприяти швидшому та більш ефективному навчанню. Студенти можуть вчитися на власному темпі, відтворюючи ситуації, поки вони не набудуть необхідні навички.
4. Можливість відстеження прогресу: Тренажери часто обладнані системами відстеження та аналізу, що дозволяє вчителям та інструкторам оцінювати прогрес студентів та спеціалістів. Це дає можливість вчасно коригувати навчальні плани та програми.

Види тренажерних систем в професійній освіті:

1. Медичні тренажери: Вони використовуються для навчання медичних студентів та лікарів різних спеціальностей. Медичні тренажери можуть відтворювати процедури, від огляду пацієнта до хірургічних операцій.

2. Авіаційні та автомобільні тренажери: Вони дозволяють пілотам та водіям навчатися та вдосконалювати свої навички в умовах, що імітують реальну роботу. Це сприяє безпеці та вмінню ефективно працювати в умовах ризику.

3. Тренажери для військових та правоохоронців: Вони моделюють бойові ситуації та дозволяють тренувати військовослужбовців та правоохоронців у реальних умовах. Це сприяє підвищенню їхньої готовності та безпеці.

4. Тренажери для інженерів та технічних спеціалістів: Вони використовуються для навчання та тестування інженерів та технічних фахівців. Це дозволяє їм вивчати роботу обладнання та систем без реального втручання.

Переваги використання тренажерів у професійній освіті:

1. Зменшення ризику: Навчання на тренажерах дозволяє фахівцям набувати досвід та вдосконалювати навички, не ризикуючи здоров'ям та життям.

2. Ефективність: Тренажери дозволяють збільшити швидкість та якість навчання. Студенти та спеціалісти можуть більше часу приділяти практичній роботі та вдосконаленню навичок.

3. Економія ресурсів: Використання тренажерів дозволяє зменшити витрати на матеріали та реальні об'єкти для навчання. Це може бути особливо важливим у сферах, де реальність дорога чи небезпечна.

Виклики та перспективи:

Незважаючи на переваги, існують певні виклики у використанні тренажерних систем у професійній освіті. Один із них - висока вартість розробки та підтримки таких систем. Додатково, для успішного використання тренажерів необхідно мати якісний викладацький персонал та інфраструктуру.

За останні роки тренажери стали більш доступними завдяки розвитку технологій віртуальної реальності (*VR*) та доповненої реальності (*AR*). Це відкриває нові перспективи для інтерактивного та захоплюючого навчання.

У майбутньому тренажерні системи можуть стати ще більш інтегрованими в професійну освіту, забезпечуючи студентам та фахівцям доступ до реалістичних сценаріїв та навчальних програм, які відповідають сучасним вимогам ринку праці.

Використання тренажерних систем у професійній освіті є важливою складовою навчального процесу. Вони дозволяють набувати навичок та досвід, зменшуючи ризики та збільшуючи ефективність навчання. Однак їх успішне впровадження вимагає великих витрат та відповідної підготовки викладацького складу. З розвитком технологій віртуальної реальності вони стають більш доступними та інтерактивними, що розширює можливості професійної освіти.

1.5. Принципи використання і розробки віртуальних тренажерів

Віртуальні тренажери є потужними інструментами для навчання та підвищення кваліфікації в різних сферах, від медицини та авіації до військової та інженерної підготовки. Ці інтерактивні системи відкривають можливості для практичного навчання в умовах, що імітують реальність, і дозволяють користувачам розвивати навички та вирішувати завдання без реального ризику. У цьому розділі розглянемо основні принципи використання та розробки віртуальних тренажерів.

Принципи використання віртуальних тренажерів:

1. Симуляція реальності: Одним із ключових принципів використання віртуальних тренажерів є можливість створювати віртуальні середовища, що максимально наближені до реальності. Це включає в себе реалістичну графіку, звукові ефекти, фізичну моделювання та відтворення реальних сценаріїв.

2. Інтерактивність: Віртуальні тренажери повинні бути інтерактивними, тобто користувачі мають можливість взаємодіяти з віртуальним середовищем та об'єктами в ньому. Це дозволяє їм виконувати практичні завдання, вирішувати проблеми та вдосконалювати навички.

3. Модульність: Віртуальні тренажери повинні бути побудовані на модульній архітектурі, що дозволяє додавати нові функціональність та сценарії без значних змін у загальній системі. Це спрощує розробку та оновлення тренажерів.

4. Відстеження прогресу: Важливим елементом є можливість відстеження прогресу користувачів. Віртуальні тренажери повинні збирати дані про

продуктивність та результати користувачів, щоб оцінити їхні досягнення та визначити області для покращення.

5. Адаптабельність: Тренажери повинні бути адаптованими до потреб користувачів. Це включає в себе можливість налаштовувати рівень складності, вибирати сценарії та завдання відповідно до вимог користувачів.

Принципи розробки віртуальних тренажерів:

1. Аналіз потреб: Першим етапом розробки віртуального тренажера є ретельний аналіз потреб користувачів. Визначення цільової аудиторії, їхніх завдань та очікуваних результатів є важливими для подальшої розробки.

2. Проектування віртуального середовища: Розробка віртуального середовища включає в себе створення 3*D* моделей, текстур, анімацій та звукового супроводу. Це важливий етап, який визначає візуальний та звуковий аспект тренажера.

3. Розробка фізичної моделі: Для тренажерів, пов'язаних з фізикою, необхідно розробити фізичну модель, яка відтворює реальні закони природи. Це дозволяє користувачам ефективно навчатися та вдосконалювати навички.

4. Програмування логіки та інтеракції: Розробка логіки тренажера передбачає програмування взаємодії об'єктів, сценаріїв та завдань. Це дозволяє користувачам взаємодіяти з віртуальним середовищем та виконувати завдання.

5. Тестування та вдосконалення: Після розробки тренажер потрібно тестувати на різних категоріях користувачів, виявляти помилки та недоліки та вносити відповідні покращення.

6. Підтримка та оновлення: Розвиток віртуальних тренажерів не закінчується після їхнього впровадження. Важливо забезпечувати підтримку користувачів та регулярно випускати оновлення з новими функціями та завданнями.

Використання віртуальних тренажерів в різних сферах:

1. Медицина: Віртуальні тренажери використовуються для медичної практики, навчання хірургічних навичок та діагностики.

2. Авіація: Пілоти навчаються використовувати віртуальні тренажери для тренування в умовах реального польоту та навчання управлінню літаком.

3. Військова підготовка: Військові користуються віртуальними тренажерами для вивчення тактики, стрільби та дій в екстремальних умовах.

4. Освіта та навчання: Віртуальні тренажери застосовуються в шкільному, вищому та професійному навчанні для набуття практичних навичок.

5. Інженерія та техніка: В розробці нових технологій та обладнання віртуальні тренажери допомагають інженерам та технічним спеціалістам тестувати та вдосконалювати свої розробки.

Виклики та перспективи:

1. Висока вартість розробки: Розробка віртуальних тренажерів може бути витратною та часомісткою задачею.

2. Необхідність підготовки користувачів: Користувачам може знадобитися певний час для засвоєння нового типу навчання.

3. Забезпечення підтримки: Підтримка та оновлення віртуальних тренажерів вимагають додаткових ресурсів та зусиль.

З розвитком технологій віртуальної та доповненої реальності, віртуальні тренажери стають більш доступними та інтерактивними. Вони мають великий потенціал у професійній освіті та підготовці, допомагаючи набувати навички та досвід у безпечному та ефективному середовищі. Розробка віртуальних тренажерів вимагає обгрунтованого аналізу потреб користувачів, реалістичних сценаріїв, а також підтримки та оновлень для забезпечення успішного впровадження та використання.

1.6. Висновки до розділу

Аналіз систем дистанційного навчання виявив певні тенденції та потреби на сучасному освітньому ринку. Зокрема, варто відзначити зростаючий попит на віртуальні тренажери або лабораторії, особливо в контексті професійної перепідготовки фахівців. Ця популярність визначається рядом чинників, які варто розглянути.

Перше, варто розглянути види віртуальних тренажерів та їх призначення. Вони можуть бути розділені на три основні групи: навчальні (допомагають у засвоєнні теоретичних знань), контролюючі (служать для оцінки рівня засвоєння матеріалу), та тренажери для розвитку практичних навичок і вмінь. Аналіз показав, що повноцінна віртуальна лабораторія повинна містити в собі тренажери всіх цих типів. Це означає, що студенти можуть отримувати комплексну освіту, яка охоплює як теоретичні знання, так і практичні навички, а також перевіряти свої досягнення.

З другого боку, важливо враховувати економічний аспект використання віртуальних тренажерів. Ефективне впровадження цих засобів в освітній процес може призвести до значних економічних вигід. Наприклад, менше потребується в матеріальних ресурсах, пов'язаних зі створенням та обслуговуванням реальних лабораторій або навчальних класів. Крім того, віртуальні тренажери можуть бути легко оновлюваними та адаптованими до сучасних потреб.

Проте, для успішного впровадження віртуальних лабораторій в освітній процес необхідний комплексний підхід. Це означає, що розробка та впровадження віртуальних 3*D*-тренажерів повинні бути здійснені в рамках загальної концепції, яка враховує потреби студентів, викладачів і адміністрації навчального закладу. До цього часу питання побудови віртуальних 3*D*-тренажерів практично не досліджувалося в роботах вітчизняних та закордонних дослідників. У більшості випадків існують окремі рішення, але відсутня загальна концепція реалізації та впровадження елементів 3*D*-візуалізації в освітній процес.

Таким чином, наукове дослідження, представлене в даній роботі, має на меті розкрити ключові аспекти створення та використання віртуальних лабораторій в освітньому процесі. Важливо створити зручні інструменти для налаштування зовнішнього оточення та забезпечити зручне середовище для користувачів та адміністраторів. Це є ключовим аспектом успішного впровадження віртуальних тренажерів у навчальний процес, який може значно полегшити навчання і підвищити якість освіти.

РОЗДІЛ 2
ПРИНЦИПИ РОЗРОБКИ ПРОЄКТУ НА *UNITY* 3*D*

2.1. Принципи побудови ігрових об'єктів і моделювання їх поведінки

2.1.1. Підмножина *DOTS .NET*

Entity Component System (*ECS*): *Entity Component System* - це парадигма розробки програмного забезпечення, яка використовується для побудови гнучких та швидких ігрових двигунів. *ECS* розриває традиційну модель об'єктно-орієнтованого програмування (ООП), замінюючи її складним набором сутностей (*entities*), компонентів (*components*) та систем (*systems*).

– Сутності (*Entities*): Сутності є контейнерами, які зберігають компоненти і визначають контекст для об'єкта в грі. Вони можуть бути чимось абстрактним, таким як "гравець" або "ворог", і містити компоненти, які надають їм функціональність.

– Компоненти (*Components*): Компоненти - це дані, які містяться в сутностях та визначають їхню поведінку. Вони є найменшими будівельними блоками в *ECS* і представляють собою прості структури даних. Наприклад, компонент "позиція" містить координати об'єкта.

– Системи (*Systems*): Системи обробляють компоненти та виконують над ними дії. Вони є функціональними блоками, які визначають логіку гри. Системи працюють над сутностями, які мають певні компоненти, і виконують обчислення на цих даних.

Data-Oriented Technology Stack (*DOTS*): *DOTS* - це технологічний стек від *Unity Technologies,* спрямований на оптимізацію та підвищення продуктивності ігрових двигунів. *DOTS* включає в себе наступні ключові компоненти:

– *Entity Component System* (*ECS*): *ECS* входить до складу *DOTS* і надає ігровим розробникам потужний інструмент для оптимізації гри за допомогою паралельного обчислення та керування пам'яттю. Це дозволяє створювати гігантські світи з безліччю сутностей та компонентів, не переживаючи про продуктивність.

– *Burst Compiler*: *Burst Compiler* - це компілятор, розроблений *Unity,* який оптимізує виконання *C#* коду на рівні машинного коду. Він дозволяє виконувати обчислення набагато швидше, що особливо важливо для ігор з великою кількістю обчислень.

– *Job System*: *Job System* дозволяє легко створювати паралельні завдання для обробки даних. Він автоматично розподіляє обчислення на різні потоки процесора, підвищуючи продуктивність гри.

– *Universal Render Pipeline* (*URP*): *URP* - це графічний рушій, який оптимізований для роботи з *ECS* і *DOTS*. Він дозволяє створювати візуально захопливі ігри з високою продуктивністю.

Вплив на ігрову розробку: Впровадження *ECS* та *DOTS* має значний вплив на ігрову розробку. Вони дозволяють створювати ігри з вражаючою графікою та високою продуктивністю на різних платформах. *ECS* спрощує управління грою, дозволяючи розробникам легко створювати та модифікувати компоненти та системи. *DOTS* надає доступ до максимальної продуктивності, забезпечуючи оптимізацію обчислень та графіки.

Підмножина *DOTS .NET* є потужним інструментом для ігрової розробки, який дозволяє створювати ігри з вражаючою графікою та високою продуктивністю. Вона впроваджує *ECS,* який спрощує управління грою та надає доступ до сучасних технологій оптимізації. *DOTS* робить ігрову розробку доступною та ефективною, допомагаючи розробникам створювати ігри нового покоління.

2.1.2. Конфігурації збірки

Конфігурації збірки в контексті розробки програмного забезпечення є критичним елементом, який визначає, як ваша програма буде зібрана, налаштована та запущена на цільовій платформі. Конфігурації збірки визначають параметри компіляції, завдання перед збіркою, налаштування середовища виконання і багато інших аспектів процесу розробки. У цій детальній теорії розглянемо концепції, засоби та практики, пов'язані з конфігураціями збірки.

Основні поняття:

1. Конфігурація збірки (*Build Configuration*): Це набір параметрів та налаштувань, які визначають, як із вихідних кодів буде зібрана програма. Кожна конфігурація може бути призначена для певної цільової платформи або середовища виконання.

2. Цільова платформа (*Target Platform*): Це обрана платформа, на якій програма буде виконуватися. Це може бути операційна система (*Windows, Linux, macOS*), апаратне забезпечення (*x*86, *ARM*), веб-браузер, мобільний пристрій або інше.

3. Компіляція (*Compilation*): Це процес перетворення вихідного коду програми, написаного на мові програмування, у машинний код або іншу форму, яку можна виконати.

4. Сценарії збирання (*Build Scripts*): Це скрипти або файли конфігурації, які вказують, як програма повинна бути зібрана. Вони можуть містити команди компіляції, завдання після збирання та інші налаштування.

Етапи налаштування конфігурацій збірки:

1. Вибір інтегрованого середовища розробки (*IDE*): Вибір *IDE* визначає, як будуть створюватися та налаштовуватися конфігурації збірки. Популярні *IDE* включають *Visual Studio, IntelliJ IDEA, Eclipse* і багато інших.

2. Створення проекту: Розробники створюють проект та вибирають цільову платформу. Проект може бути монолітним (всі частини разом) або розділеним на модулі.

3. Конфігурація збірки: Розробники визначають параметри збірки, такі як типи оптимізації, рівень логування, активування певних функцій (наприклад, відладка) та інші. Ці параметри можуть бути різними для різних конфігурацій.

4. Завдання збирання (*Build Tasks*): В конфігураціях збірки вказуються завдання, які повинні виконуватися перед або після збирання, такі як компіляція, копіювання файлів, стиснення або створення документації.

5. Послідовність збирання (*Build Pipeline*): Це послідовність завдань збирання, які виконуються в певному порядку. Зазвичай вони включають в себе компіляцію, лінкування, тестування та створення виконуваних файлів.

6. Цільовий вихід (*Build Output*): Результатом процесу збирання є один чи кілька виконуваних файлів, бібліотек або інших ресурсів, які готові до виконання на цільовій платформі.

Оптимізація та кросплатформеність:

– Оптимізація: Конфігурації збірки можуть бути налаштовані для оптимізації продуктивності, швидкості роботи або розміру виконуваних файлів. Це важливо для ігор, які повинні працювати на різних пристроях з обмеженими ресурсами.

– Кросплатформеність: Важливо, щоб конфігурації збірки були налаштовані для підтримки різних платформ. Зокрема, це стосується мобільних пристроїв, веб-додатків та різних операційних систем.

Системи керування конфігураціями:

Для кращого управління конфігураціями збірки використовуються системи керування конфігураціями, такі як *Git, SVN, Mercurial* і багато інших. Вони дозволяють відстежувати зміни в конфігураціях, зберігати історію змін та спільно працювати над проектом.

Конфігурації збірки є важливою складовою розробки програмного забезпечення. Вони дозволяють розробникам налаштовувати параметри компіляції, завдання збирання та інші аспекти процесу збирання програми. Правильна конфігурація забезпечує оптимізацію продуктивності та кросплатформену сумісність, що робить її важливою для розробки ігор та програмного забезпечення взагалі.

2.1.3. Час виконання та пакети

Ці концепції стосуються того, як оптимізувати та керувати процесом збирання та виконання програм для досягнення максимальної продуктивності та ефективності. У цій детальній теорії ми розглянемо поняття часу виконання, пакетів збірки та їх вплив на розробку програмного забезпечення.

Основні поняття:

1. Час виконання (*Build Time*): Це кількість часу, який потрібен для того, щоб зібрати (компілювати, лінкувати та ін.) вихідний код програми в виконуваний файл

або бібліотеку. Час виконання є критичним параметром для розробників, оскільки він впливає на продуктивність та швидкість розробки.

2. Пакети збірки (*Build Artifacts*): Це результати процесу збирання, такі як виконувані файли, бібліотеки, зображення, конфігураційні файли і будь-які інші ресурси, необхідні для виконання програми. Пакети збірки можуть бути однієї чи кількох версій і можуть бути розділені на різні пакети відповідно до конфігурацій збірки.

3. Залежності збірки (*Build Dependencies*): Це файли, модулі або ресурси, які потрібні для успішного завершення процесу збирання. Залежності збірки можуть включати в себе бібліотеки, заголовочні файли, файли конфігурації та інше.

Час виконання та його вплив:

– Важливість часу виконання: Час виконання є важливим параметром для розробників, оскільки тривалість процесу збирання може впливати на продуктивність. Довгий час збирання може уповільнити розробку та спричинити очікування.

– Зменшення часу виконання: Розробники використовують різні методи та техніки для зменшення часу виконання, такі як оптимізація компіляції, паралельна збірка, кешування та інші. Важливо збалансувати зусилля, спрямовані на зменшення часу виконання, із підтримкою необхідної функціональності.

– Складність проекту: Великі та складні проекти можуть мати довший час виконання, оскільки вони містять більше коду, залежностей і ресурсів. Розробники повинні розглядати можливості розбиття проекту на менші модулі, які можуть бути зібрані окремо.

Пакети збірки та їх використання:

– Спрощення розповсюдження: Пакети збірки роблять розповсюдження програмного забезпечення більш простим. Розробники можуть передавати іншим користувачам або екіпажам лише необхідні пакети для виконання програми.

– Забезпечення консистентності: Використання пакетів дозволяє забезпечити консистентність усіх складових програми на різних етапах розробки та виконання. Це важливо для підтримки працездатності та уникнення помилок.

Завдання збірки та їх планування:

– Автоматизація завдань збірки: Розробники використовують інструменти автоматизації, такі як *Make, CMake, Gradle* або інші, для автоматизації завдань збірки. Це дозволяє забезпечити репродуктивність та ефективність процесу.

– Планування завдань збірки: Планування та виконання завдань збірки важливо для забезпечення коректності та ефективності процесу. Розробники можуть встановлювати пріоритети для різних завдань збірки та виконувати їх у відповідному порядку.

Час виконання та пакети збірки є важливими аспектами розробки програмного забезпечення. Вони впливають на продуктивність, розповсюдження та консистентність програми. Розробники повинні ретельно планувати та оптимізувати процес збирання, щоб досягти максимальної ефективності та забезпечити якість свого програмного забезпечення.

2.2. Налаштування проєкту та пакету

Налаштування проєкту та пакету - це фундаментальний етап розробки програмного забезпечення, що передує власному процесу кодування. Налаштування дозволяє розробникам створити зручні та продуктивні умови для роботи над проєктом, а також забезпечити правильне виконання всіх необхідних операцій під час збирання, тестування та розгортання програми.

Основні аспекти налаштування проєкту та пакету:

1. Вибір інструментів розробки: Важливим аспектом є вибір інтегрованого середовища розробки (*IDE*) або текстового редактора, які найкраще відповідають потребам проєкту. Популярні *IDE* включають в себе *Visual Studio, IntelliJ IDEA, Eclipse* тощо.

2. Вибір мови програмування: Вибір мови програмування визначає, як буде реалізовано логіку програми. Це може бути мова *C++, Java, Python, JavaScript, C#* або інша, в залежності від специфіки проєкту.

3. Система контролю версій: Використання системи контролю версій, такої як *Git,* дозволяє відстежувати зміни у вихідному коді, спільно працювати над проєктом і забезпечує можливість відновлення попередніх версій коду у випадку помилок.

4. Конфігурація середовища розробки: Налаштування *IDE,* додавання плагінів, встановлення стандартів коду, кольорової палітри, шрифтів і гіткружок допомагають створити комфортне робоче середовище.

5. Налаштування залежностей: Для успішної розробки часто потрібно встановити та налаштувати різні бібліотеки, пакети та компоненти, які використовуються в проєкті. Використання менеджерів пакетів, таких як *npm* для *JavaScript* або *NuGet* для *C#,* дозволяє забезпечити потрібні залежності.

6. Конфігурація збирання та компіляції: Налаштування параметрів збирання і компіляції коду, включаючи вибір компілятора, визначення стандартів, встановлення шляхів до бібліотек та інше.

7. Параметри розгортання та тестування: Для розгортання програми на віддаленому сервері або хостингу, налаштовуються параметри, такі як адреси серверів баз даних, конфігурація серверів інтеграції, параметри безпеки тощо. Також визначаються налаштування тестування, включаючи автоматичні тести, інтеграційні тести та набори даних для тестів.

Завдання та кроки налаштування:

1. Створення нового проєкту: Розробники створюють новий проєкт та визначають його основні параметри, такі як назва, тип проєкту, мова програмування та шляхи до проєкту на файловій системі.

2. Налаштування інтегрованого середовища розробки: Вибране *IDE* налаштовується відповідно до потреб розробки. Це включає в себе встановлення необхідних плагінів, налаштування зовнішнього вигляду та інтерфейсу користувача.

3. Встановлення системи контролю версій: Розробники ініціюють систему контролю версій та створюють репозиторій для зберігання вихідного коду. Вони також додають файл *.gitignore* для визначення файлів, які не повинні включатися до версійного контролю.

4. Налаштування залежностей: Розробники встановлюють необхідні бібліотеки та пакети, які використовуються в проєкті. Це може включати встановлення пакетів через менеджери пакетів або налаштування файлу *package.json* (для *JavaScript* проектів) або *packages.config* (для *.NET* проектів).

5. Конфігурація збирання та компіляції: Розробники визначають параметри компіляції, такі як версія цільової платформи, типи бінарних файлів, шляхи до збірок тощо.

6. Параметри розгортання та тестування: Налаштовуються параметри, необхідні для розгортання програми на цільовому сервері або платформі. Це може включати встановлення змінних середовища, налаштування конфігураційних файлів, визначення параметрів баз даних, ключів безпеки і налаштування тестових середовищ.

Наприклад, у зразку *TinyPhysics* список збірок, на які посилаються, виглядає так (рис. 2.1).

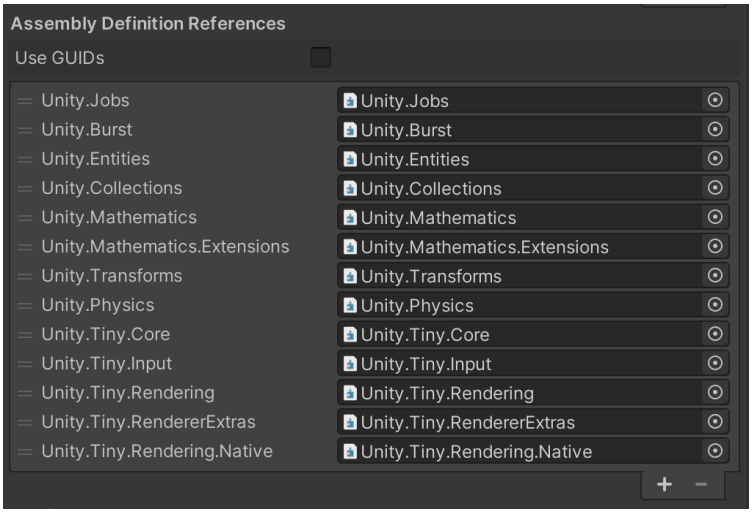

Рис. 2.1. Список збірок у *TinyPhysics*

Налаштування проєкту та пакету є важливим кроком у розробці програмного забезпечення, який впливає на продуктивність та ефективність роботи розробників, які роблять спільні зусилля для досягнення успіху проєкту. Грамотно налаштоване робоче середовище та правильно налаштовані параметри забезпечують більш якісну та ефективну розробку програмного забезпечення.

2.2.1. Налаштування сцени

Налаштування сцени є ключовим етапом при розробці інтерактивних 3*D* додатків, включаючи ігри, симулятори, віртуальну реальність і багато інших проектів. Сцена - це цифровий простір, в якому відбувається весь візуальний досвід користувача, і який включає в себе об'єкти, освітлення, камери та інші компоненти.

Основні аспекти налаштування сцени:

1. Створення сцени: Перший крок - створення нової сцени в інтегрованому середовищі розробки (*IDE*) або іншому спеціалізованому програмному засобі для роботи з 3*D* графікою, такому як *Unity*3*D* або *Unreal Engine*. Вибір інструменту залежить від характеру проєкту та розробників.

2. Додавання об'єктів: Далі необхідно додати об'єкти на сцену. Це можуть бути персонажі, предмети, ландшафти, будівлі тощо. Розробники можуть створювати власні 3*D* моделі або використовувати готові, наприклад, з онлайн-бібліотек або ринку активів.

3. Розташування об'єктів: Після додавання об'єктів їх потрібно розташувати на сцені так, щоб створити бажаний вигляд і розміщення. Це включає в себе переміщення, обертання і зміну масштабу об'єктів для досягнення потрібного композиційного рішення.

4. Освітлення: Освітлення грає важливу роль у створенні реалістичної сцени. Розробники налаштовують джерела світла, їхні властивості, такі як інтенсивність та кольори, і визначають, як вони впливають на об'єкти на сцені.

5. Камери: Встановлення камери дозволяє визначити точку спостереження користувача. Розробники можуть налаштовувати параметри камери, такі як поле зору, позиція, орієнтація та проекційна матриця для досягнення потрібної перспективи.

6. Колізії: Реалістичні інтеракції між об'єктами в сцені вимагають обробки колізій. Розробники встановлюють колайдери на об'єктах і визначають їх властивості, щоб об'єкти правильно взаємодіяли між собою.

7. Події та скрипти: Для надання об'єктам реактивності розробники додають скрипти та визначають події. Це може включати в себе обробку введення користувача, анімацію, штучний інтелект та багато іншого.

Завдання та кроки налаштування сцени:

1. Додавання сцени: В інтегрованому середовищі розробки створюється новий проєкт або відкривається існуючий. Після цього створюється нова сцена або використовується існуюча.

2. Додавання об'єктів: За допомогою інтерфейсу редактора розробники додають необхідні об'єкти на сцену. Це може бути здійснено за допомогою драг-анд-дроп або програмно шляхом створення об'єктів у віртуальному просторі.

3. Розташування об'єктів: Об'єкти розташовуються на сцені за допомогою переміщення, обертання та зміни масштабу в тривимірному просторі. Розробники використовують інтерфейс редактора для зручного розміщення.

4. Освітлення: Встановлення джерел світла, налаштування їх параметрів та розташування на сцені для досягнення певного освітлення та тіней.

5. Камери: Створення та налаштування камер для визначення перспективи і точки спостереження.

6. Колізії: Додавання колайдерів на об'єкти та налаштування їхніх властивостей для обробки колізій.

7. Скрипти та події: Розробка та прикріплення скриптів до об'єктів для надання їм функціональності. Визначення та обробка подій для реакції на взаємодію користувача або інших подій у сцені.

Налаштування сцени є важливим кроком при розробці 3*D* додатків і впливає на якість та реалістичність візуального досвіду користувачів. Коректно налаштована сцена дозволяє розробникам створити інтерактивні і відчутні віртуальні світи, які відповідають потребам проєкту. Налаштування об'єктів, освітлення, камер та інших компонентів дозволяє досягти бажаних результатів у створенні 3*D* додатків.

2.2.2. Багатопотокові / однопотокові варіанти

Багатопотокове програмування та однопотокове програмування є важливими концепціями в сучасному розробці програмного забезпечення. Вони визначають, як

програми обробляють багатозадачність та одночасність, і вибір між ними може суттєво впливати на продуктивність, надійність та вартість розробки програми. У цьому огляді ми детально розглянемо багатопотокове та однопотокове програмування, їх варіанти, переваги та недоліки.

Багатопотокове програмування

Багатопотокове програмування (*multithreading*) - це підхід до програмування, де програма розділяється на декілька незалежних потоків виконання. Кожен потік виконує певний набір інструкцій, і всі ці потоки працюють одночасно на багатозадачному пристрої або в середовищі операційної системи.

Переваги багатопотокового програмування:

1. Підвищення продуктивності: Багатопотокові програми можуть виконувати багато завдань паралельно, що призводить до підвищення продуктивності, особливо на багатоядерних процесорах.

2. Інтерактивність: Можливість реагувати на введення користувача або події під час виконання інших завдань.

3. Використання ресурсів: Можливість кращого використання ресурсів пристрою (наприклад, *CPU,* пам'ять) завдяки паралельному виконанню.

4. Спрощення коду: Деякі завдання, такі як обробка одночасних подій або асинхронна комунікація, можуть бути легше реалізовані за допомогою багатопотокового програмування.

Недоліки багатопотокового програмування:

1. Синхронізація: Потоки можуть конфліктувати між собою при доступі до спільних ресурсів, що може призвести до помилок інтерфейсу та навіть до "гонок" (*race conditions*).

2. Складність розробки: Багатопотокові програми можуть бути складнішими у розробці та тестуванні через складність синхронізації та взаємодії потоків.

3. Витрати на ресурси: Кожен потік вимагає додаткових системних ресурсів, таких як пам'ять та обробка, що може призвести до збільшення витрат на обладнання.

Однопотокове програмування

Однопотокове програмування - це підхід, де програма виконується в одному потоці виконання. Весь код обробляється послідовно, і немає паралельних виконань.

Переваги однопотокового програмування:

1. Спрощення синхронізації: Відсутність паралельних потоків уникає багатьох проблем синхронізації і конфліктів.

2. Стабільність: Однопотокові програми зазвичай менше схильні до виникнення помилок через однозначність виконання.

3. Легша розробка: Розробка однопотокових програм може бути більш простою і зрозумілою, особливо для менших проектів.

Недоліки однопотокового програмування:

1. Продуктивність: Однопотокові програми не використовують повністю потенціал багатоядерних процесорів, тому в деяких випадках може бути менш продуктивними.

2. Обробка подій: В однопотокових програмах обробка одночасних подій може призвести до блокування інтерфейсу користувача.

Вибір між багатопотоковим та однопотоковим програмуванням залежить від конкретних потреб проекту. Ось деякі вказівки для вибору:

1. Багатопотокове програмування: Використовуйте його, якщо у вас є завдання, які легко можуть бути розділені на паралельні потоки, і якщо ви хочете досягти підвищення продуктивності. Це особливо корисно для завдань обчислення або обробки даних.

2. Однопотокове програмування: Використовуйте його, якщо ваша програма має просту логіку виконання, і складність багатопотокового програмування перевищує можливі переваги. Це може бути відмінним вибором для інтерактивних програм та веб-застосунків.

Багатопотокове та однопотокове програмування - це два різних підходи до організації виконання програм. Кожен з них має свої переваги та недоліки, і вибір між ними повинен залежати від конкретних потреб вашого проекту. Важливо розуміти обидва підходи та вміти правильно вибирати між ними для досягнення оптимальних результатів у розробці програмного забезпечення.

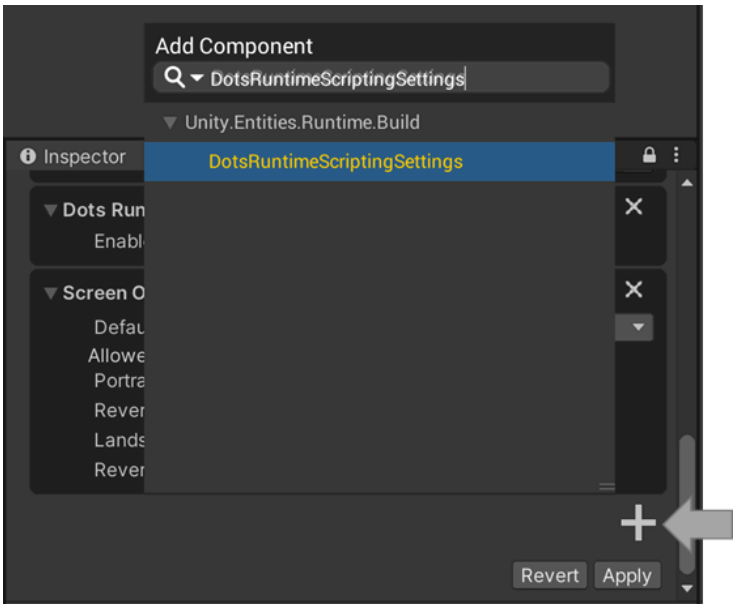

Рис. 2.2. Компонент *DotsRuntimeScriptingSettings*

2.3. Рішення *DOTS C#*

Unity, одне з найпопулярніших ігрових середовищ, завжди було привабливим для розробників завдяки своїй простоті та потужній графіці. Однак із зростанням амбіцій ігрових проєктів і вимог до реалістичності та продуктивності, виникали складні завдання для цієї платформи. *Unity* відповів на ці вимоги, представивши *DOTS* (*Data-Oriented Technology Stack*) та *C# Job System* - рішення, які дозволяють розробникам створювати більш швидкі та оптимізовані ігри.

Основні поняття

1. *DOTS* (*Data-Oriented Technology Stack*)

DOTS - це стек технологій від *Unity,* спрямований на оптимізацію продуктивності та роботи з великими обсягами даних. Основними компонентами *DOTS* є:

– *Entity Component System* (*ECS*): Новий підхід до створення об'єктів у грі, де дані і функції розділені та оптимізовані для роботи з багатьма об'єктами одночасно.

– *C# Job System*: Система, яка дозволяє виконувати обчислення в окремих потоках *CPU* для максимального використання багатоядерних процесорів.

– *Burst Compiler*: Компілятор, який генерує низькорівневий код для оптимізації виконання функцій на мові *C#*.

Ці компоненти разом утворюють стек *DOTS,* який дозволяє розробникам створювати високопродуктивні та швидкі ігри на *Unity*.

2. *C# Job System*

C# Job System - це один з ключових компонентів *DOTS*. Він дозволяє розробникам створювати паралельні обчислення у своєму коді, що забезпечує максимальне використання потужності багатоядерних процесорів.

Основні концепції *C# Job System*:

– *Job*: Задача, яка виконується в окремому потоці виконання. Вона має обмежену функціональність і виконується паралельно з іншими задачами.

– *JobHandle*: Об'єкт, який представляє виконання задачі. Він дозволяє синхронізувати виконання різних задач і контролювати їхню послідовність.

– *System*: Організація коду за допомогою *ECS* та *Job System*. Система обробляє певний аспект логіки гри і працює з компонентами та задачами.

Переваги *DOTS C#*

1. Підвищена продуктивність

DOTS C# дозволяє оптимізувати програми для роботи на багатоядерних процесорах. Завдяки *C# Job System* та *Burst Compiler,* ви можете виконувати обчислення швидше і ефективніше.

2. Робота з великими обсягами даних

DOTS дозволяє оптимізовано працювати з великими масивами даних. *ECS* дозволяє робити це швидко та ефективно.

3. Простота розробки

Хоча *DOTS* вимагає вивчення нових підходів до розробки, це може спростити розробку складних ігор. *ECS* дозволяє легко створювати та управляти об'єктами в грі.

4. Кросплатформенність

Продукти, створені з використанням *DOTS C#,* легко портуються на різні платформи, включаючи мобільні пристрої та консолі.

Недоліки *DOTS C#*

1. Вимоги до ресурсів

DOTS може бути вимогливим до ресурсів, особливо якщо ви працюєте з великими обсягами даних та складними обчисленнями.

2. Вивчення нових концепцій

Використання *DOTS* вимагає ознайомлення з новими концепціями, такими як *ECS* та *C# Job System,* що може бути викликаною для деяких розробників.

3. Обмежена підтримка платформ

На даний момент, не всі платформи підтримують *DOTS C#,* тому вам потрібно буде звертати увагу на це питання при розробці.

DOTS C# - це потужний інструмент для розробки високопродуктивних ігор та додатків у середовищі *Unity*. Він дозволяє розробникам ефективно працювати з великими обсягами даних та оптимізувати продуктивність своїх проєктів. Однак він також вимагає від розробників вивчення нових концепцій та може бути вимогливим до ресурсів. Правильно використовуючи *DOTS C#,* ви можете створити ігри, які вражатимуть ваших користувачів своєю швидкістю та реалістичністю.

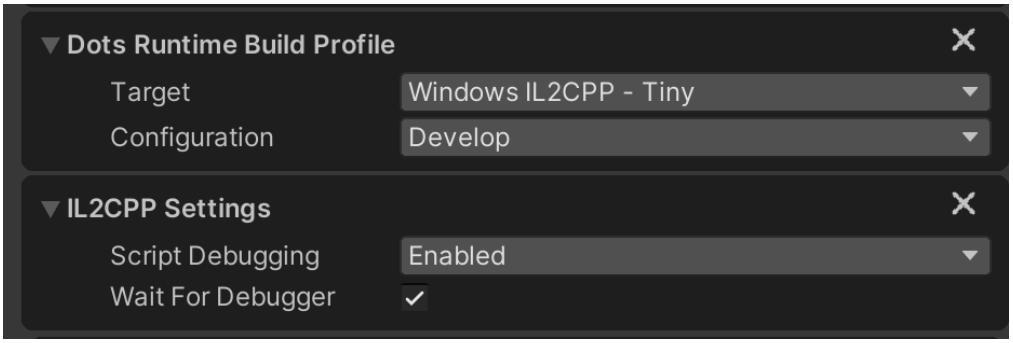

Рис. 2.3. Вікно налаштувань

2.4. Зображення характеристик і властивостей *GameObject*/*DOTS*

Unity - це потужний інструмент для створення ігор та інтерактивних додатків, і в центрі його функціоналу знаходяться *GameObject* і *DOTS* (*Data-Oriented Technology Stack*). В цьому розділі ми розглянемо основні характеристики та

властивості цих концепцій і як вони взаємодіють для створення ігрових середовищ та інтерактивних додатків.

1. *GameObject* в *Unity*

GameObject - це основний будівельний блок у грі або додатку, створюваний в середовищі *Unity*. Він представляє собою об'єкт, який може мати графічне відображення, компоненти та логіку, яка забезпечує його поведінку. Основні характеристики *GameObject* включають:

– Позиція та орієнтація: Координати в тривимірному просторі, орієнтація та масштаб, що визначають положення об'єкта в грі.

– Компоненти: *GameObject* може містити різні компоненти, такі як *MeshRenderer* для відображення графіки, *Collider* для взаємодії з іншими об'єктами, і власні скрипти, які додають логіку.

– Дерево об'єктів: *GameObject* може бути частиною ієрархії об'єктів, де деякі об'єкти можуть бути батьками, а інші - дітьми. Це важливо для управління та організації об'єктів на сцені.

– Теги та шари: Властивості, які дозволяють класифікувати і ідентифікувати *GameObject* для подальшого використання у логіці гри.

2. *DOTS* (*Data-Oriented Technology Stack*)

DOTS - це стек технологій, розроблений *Unity* для покращення продуктивності ігрової розробки. Він включає в себе *ECS* (*Entity Component System*), *C# Job System* та *Burst Compiler,* які дозволяють розробникам оптимізовувати ігровий код та керувати обчисленнями великими обсягами даних.

Характеристики *DOTS*

– *Entity Component System* (*ECS*): *ECS* використовується для створення об'єктів (*Entities*), які містять дані (*Components*) та системи (*Systems*), які обробляють ці дані. Ця архітектура дозволяє оптимізовано керувати об'єктами великими обсягами даних.

– *C# Job System*: Система, яка дозволяє виконувати обчислення в окремих потоках процесора для максимального використання багатоядерних систем.

– *Burst Compiler*: Компілятор, який генерує оптимізований машинний код з *C#* для виконання обчислень з високою швидкістю.

Взаємодія *GameObject* та *DOTS*

DOTS може взаємодіяти з *GameObject* в *Unity* наступним чином:

– Перетворення *GameObject* в *Entity*: Ви можете створити *Entity* в *DOTS*, яка представлятиме собою *GameObject* з *Unity*. Це дозволяє використовувати потужність *DOTS* для оптимізації обчислень.

– Системи *DOTS* та компоненти *Unity*: Ви можете використовувати системи *DOTS* для управління компонентами *GameObject*. Наприклад, ви можете використовувати *DOTS* для оптимізації фізики або штучного інтелекту в грі.

– Більша продуктивність: Використання *DOTS* для обчислень дозволяє покращити продуктивність великих ігор і додатків, які використовують багато *GameObject*.

GameObject та *DOTS* є важливими компонентами розробки в *Unity*. *GameObject* представляє базовий об'єкт у грі з усіма його характеристиками та властивостями. *DOTS* надає потужні технології для оптимізації продуктивності та керування обчисленнями. Взаємодія між ними дозволяє створювати ігри та додатки, які поєднують в собі високу продуктивність та гнучкість розробки.

Вимкнення камери під час виконання може призвести до збою (рис. 2.4).

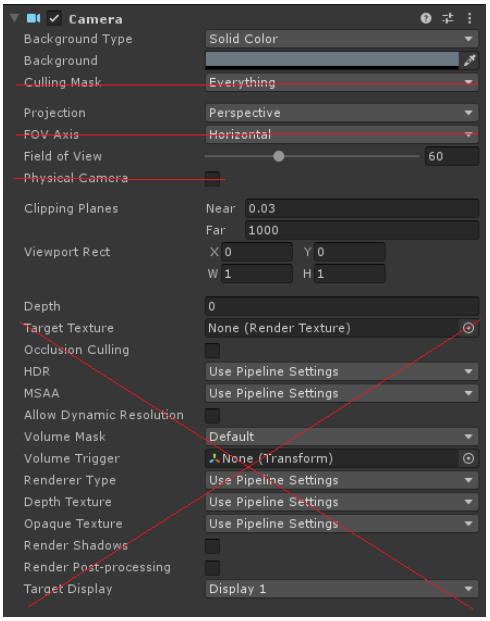

Рис. 2.4. Функції, що підтримуються камерою

Візуальне представлення карти властивостей в *ECS* для *UnityEngine.Sprite* зображено на рис. 2.5.

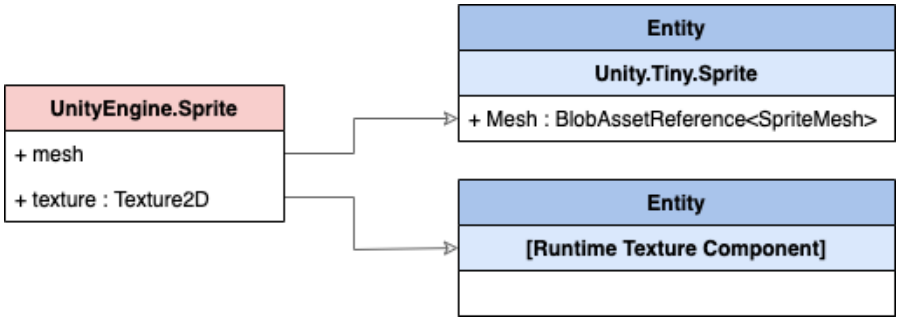

Рис. 2.5. Візуальне представлення карти властивостей в *ECS* для *UnityEngine.Sprite*

Ось візуальне представлення карти властивостей в *ECS* для *UnityEngine.SpriteRenderer* (рис. 2.6).

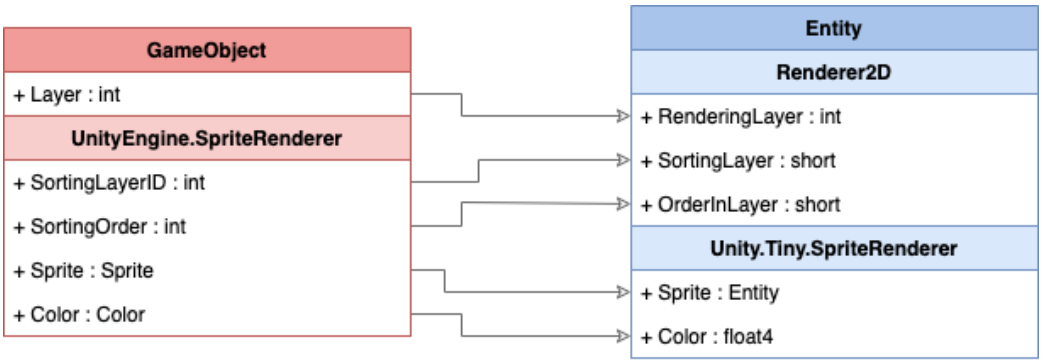

Рис. 2.6. Візуальне представлення карти властивостей в *ECS* для *UnityEngine.SpriteRenderer*

Представлення того, як різні 2*D*-об’єкти візуалізації та їх компоненти взаємодіють один з одним (рис. 2.7).

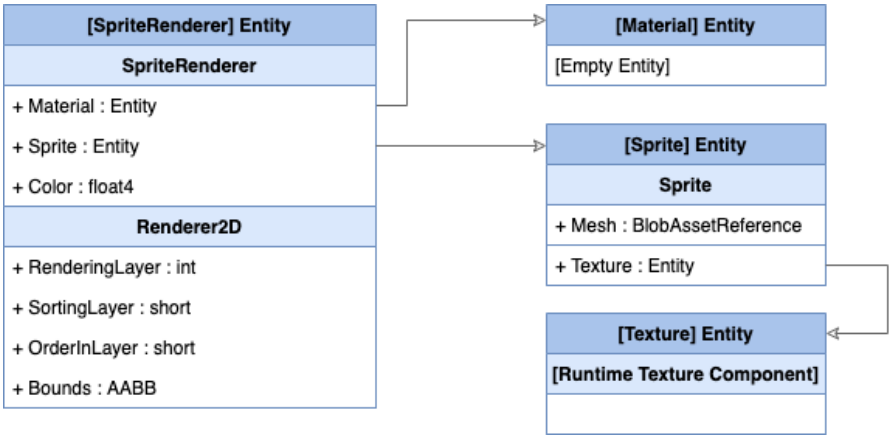

Рис. 2.7. 2*D*-об’єкти візуалізації та їх компоненти

2.5. Висновки до розділу

В даному розділі було представлено огляд інструментарію *DOTS,* який в *Unity* використовувався для реалізації руху об’єктів.

Аналіз показав, що властивості *Tiny particles* відображають підмножину властивостей *Shuriken.* Властивості *Shuriken* розбиті на групи, які називаються «модулями». У цьому розділі буде описано, які властивості в кожному модулі підтримуються в *Tiny.* Якщо модуль відсутній у списку, то в даний час жодні властивості цього модуля не підтримуються. Для непідтримуваних властивостей використовуються значення за замовчуванням, показані у вікні Інспектора, якщо не вказано інше.

.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ВІЗУАЛІЗАЦІЇ РУХУ ОБ'ЄКТІВ В 3*D* СИМУЛЯТОРАХ

3.1. Призначення та область застосування

Після завантаження та налаштування графічної частини проєкту, наступним важливим кроком є створення необхідних функціональних класів, які були описані у діаграмі класів (див. рис. 3.1). Кожен з цих класів відповідає за конкретну функціональність симулятора та має своє призначення. Розглянемо деякі з них більш докладно:

1. Класи для імітації руху об'єктів: Для кожного об'єкта сонячної системи (планети, сонця тощо) були створені окремі класи, які відповідають за імітацію їхнього руху. Ці класи включають в себе розрахунки шляху руху об'єкта відповідно до законів гравітації та дійсних параметрів планет. Це дозволяє симулювати рух об'єктів у тривимірному просторі точно та реалістично.

2. Моделі планет та об'єктів: Для кожної планети та об'єкта були створені 3*D* моделі, які реалістично відображають їхні розміри та форму. Ці моделі використовуються для візуалізації об'єктів на сцені симулятора.

3. Система відображення орбіт: Для відображення орбіт планет була створена система, яка динамічно розраховує траєкторії руху об'єктів та відображає їх на сцені. Це допомагає користувачам бачити, як планети рухаються вздовж своїх орбіт.

4. Інтерактивність та можливість взаємодії: Симулятор надає можливість взаємодії з об'єктами сонячної системи. Користувачі можуть вибирати планети, дізнаватися більше інформації про них та взаємодіяти з їхнім рухом. Для цього були створені відповідні класи та інтерфейси для обробки взаємодії.

Всі ці класи і компоненти разом створюють функціональну основу симулятора сонячної системи, яка дозволяє користувачам вивчати та аналізувати рух об'єктів сонячної системи в реальному часі з високою ступенем реалізму. Такий підхід

дозволяє створити інтерактивний та навчальний інструмент для дослідження космічних явищ та вивчення сонячної системи.

У процесі розробки проєкту використовуємо ігровий двигун *Unity* 2018 та відповідний фреймворк для C# на програмному середовищі *Visual Studio* 2017.

Фреймворк уже містить клас графічних об’єктів на сцені – *GameObject* та клас реалізації логіки – *MonoBehavior.* При чому зв’язки фреймворку дозволяють підключати різноманітні екземпляри класів логіки до екземпляру класу графічних об’єктів на сцені.

Відповідно до цього створюємо необхідний перелік графічних класів, які будуть наслідувати *GameObject* та перелік класів логіки, які наслідуватимуть *MonoBehavior* (табл. 3.1).

Таблиця.3.1

Таблиця необхідних класів

Графічні класи	Логічні класи
Сонце(*Sun*)	Обертання навколо своєї осі (*Rotation*)
Планета(*Planet*)	Обертання навколо космічного тіла (*AroundRotation*)
Камера(*Camera*)	Контроль камери (*CamController*)
Панель довідки(*Info*)	Менеджер довідки планети (*Selection*)

Маючи перелік необхідних класів та базовий опис їх призначення побудуємо діаграму класів з описом їх взаємодії (рис. 3.1).

Наступним кроком створюємо, та налаштовуємо новий 3*D* проєкт на *Unity.* Через графічний інтерфейс двигуна створюємо дочірні класи від *GameObject.* Додаємо до них необхідні моделі та розміщуємо у просторі кімнати програми (рис. 3.2).

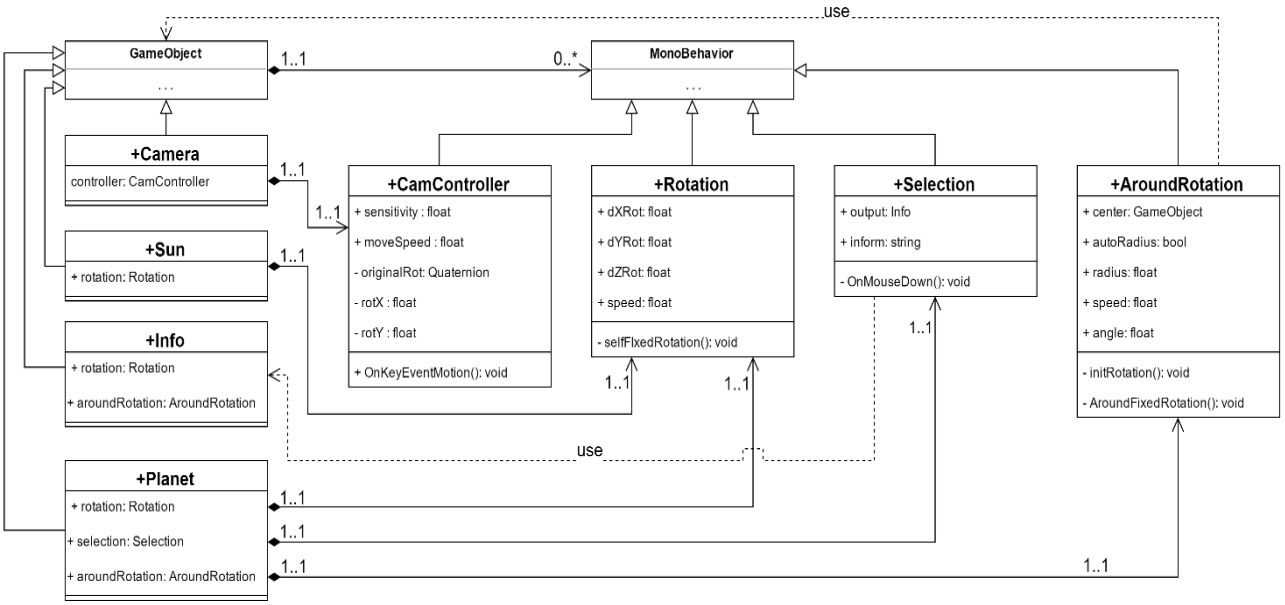

Рис. 3.1. Побудова діаграми класів проєкту

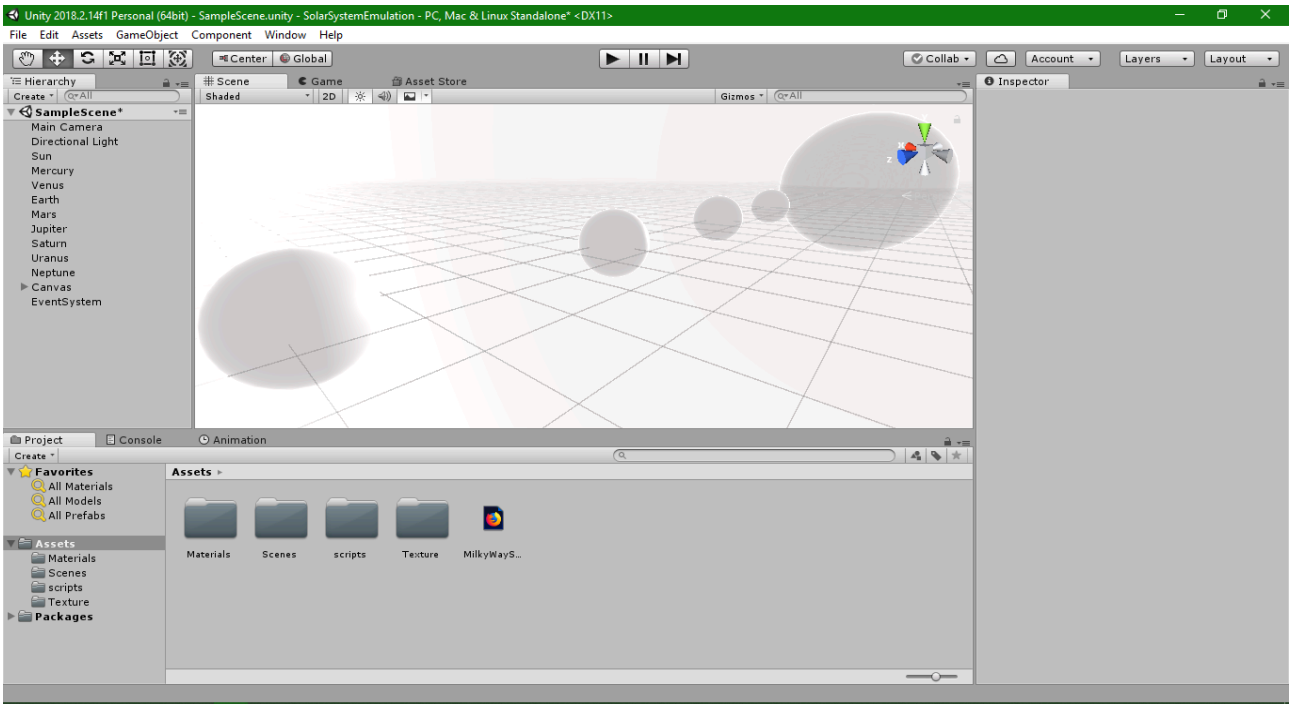

Рис. 3.2. Розміщення графічних об’єктів

У процесі розробки нашого проєкту ми використовуємо ігровий двигун *Unity* 2018, який є одним із найпопулярніших інструментів для розробки ігор та інтерактивних додатків. *Unity* надає нам потужний інструментарій для створення тривимірних і двовимірних середовищ, а також для реалізації логіки взаємодії об'єктів в цих середовищах. При цьому ми використовуємо мову програмування *C#*

для написання логіки нашого проєкту, і зв'язок між *Unity* та *C#* здійснюється через відповідний фреймворк та інтеграцію з програмним середовищем *Visual Studio* 2017.

Основними компонентами, які ми використовуємо при розробці в *Unity,* є такі:

1. *GameObject*: Клас *GameObject* є базовим класом для всіх графічних об'єктів на сцені. Цей клас дозволяє нам створювати, розміщувати та керувати об'єктами в грі. Від камери і персонажів до об'єктів оточення - всі вони є екземплярами класу *GameObject*. Ми можемо додавати до об'єктів компоненти, які відповідають за різні аспекти їх поведінки.

2. *MonoBehavior*: Клас *MonoBehavior* є базовим класом для створення логіки об'єктів в *Unity*. Цей клас включає різноманітні методи життєвого циклу, такі як *Awake, Start, Update, FixedUpdate* та інші. Ми можемо наслідувати цей клас у власних скриптах, щоб реалізувати логіку об'єктів. Наприклад, ми можемо створити скрипт для керування рухом персонажа, обробки взаємодії з іншими об'єктами тощо.

Зв'язок між графічними об'єктами на сцені та їх логікою встановлюється за допомогою компонентів. Кожен об'єкт може мати різні компоненти, які додають функціональність об'єкту. Наприклад, компонент *Rigidbody* може бути доданий до об'єкта для надання йому фізичних властивостей, таких як гравітація та реакція на зіткнення. Ми можемо створювати власні компоненти, які наслідують класи *Unity,* для реалізації специфічної логіки наших об'єктів.

Для забезпечення оптимальної організації роботи з різними класами об'єктів ми створюємо власний перелік графічних класів, які будуть наслідувати клас *GameObject*. Це дозволяє нам створювати власні типи об'єктів зі специфічними властивостями та методами.

Також ми створюємо перелік класів логіки, які наслідуватимуть клас *MonoBehavior*. Ці класи міститимуть реалізацію різних аспектів поведінки наших об'єктів. Наприклад, можливі класи для руху, взаємодії з користувачем, обчислення фізики тощо.

Усі ці компоненти та класи логіки об'єднуються разом для створення цілісного проєкту дистанційного навчання, де студенти можуть взаємодіяти з віртуальними об'єктами та засвоювати необхідні знання та навички. *Unity* та мова програмування

C# надають нам інструменти для створення реалістичних та взаємодіємих середовищ для навчання.

Маючи перелік необхідних класів та базовий опис їх призначення, ми переходимо до створення діаграми класів з описом їх взаємодії. Діаграма класів допоможе нам краще розібратися в структурі нашого проєкту та визначити залежності між класами.

Діаграма класів може включати такі основні елементи:

1. Класи: Кожен клас представляє собою окремий об'єкт або компонент нашого проєкту. У нашому випадку це можуть бути класи, які відповідають за різні аспекти навчального середовища, такі як об'єкти в кімнаті, користувачі, взаємодія з об'єктами тощо.

2. Атрибути класів: Ми можемо додавати атрибути до класів, щоб вказати їхні характеристики та властивості. Наприклад, клас "Користувач" може мати атрибути, такі як ім'я користувача, рівень доступу тощо.

3. Методи класів: Методи класів визначають функціональність цих класів. Наприклад, метод "рухати_об'єкт" може бути визначений у класі, який представляє об'єкти в кімнаті.

4. Залежності та асоціації: Діаграма може також показувати зв'язки між класами. Наприклад, клас "Користувач" може мати асоціацію з класом "Об'єкт", яка вказує на те, що користувач може взаємодіяти з об'єктами.

Після побудови діаграми класів ми переходимо до створення та налаштування нового 3*D* проєкту в середовищі *Unity*. *Unity* надає графічний інтерфейс, який спрощує створення та редагування об'єктів у віртуальному просторі.

1. Створення дочірніх класів від *GameObject*: Відповідно до структури нашого проєкту, ми створюємо нові класи, які наслідують клас *GameObject*. Ці класи представляють різні об'єкти в нашій віртуальній кімнаті навчання. Наприклад, це можуть бути столи, стільці, дошки, екран комп'ютера та інші об'єкти, які можуть бути присутні в реальному навчальному середовищі.

2. Додавання моделей та розміщення в просторі: Ми додаємо графічні моделі цих об'єктів та розміщуємо їх у віртуальному просторі нашого проєкту. Це

допомагає створити реалістичне середовище для навчання, де користувачі можуть взаємодіяти з об'єктами та вивчати їх властивості.

На цьому етапі ми готові до подальших кроків розробки нашого дистанційного навчального середовища, де студенти зможуть взаємодіяти з об'єктами та навчатися в інтерактивному середовищі.

Далі завантажуємо необхідні текстури знімків поверхні планет сонячної системи та Сонця. За допомогою графічного редактора накладаємо їх на уже розміщені моделі планет (рис. 3.3).

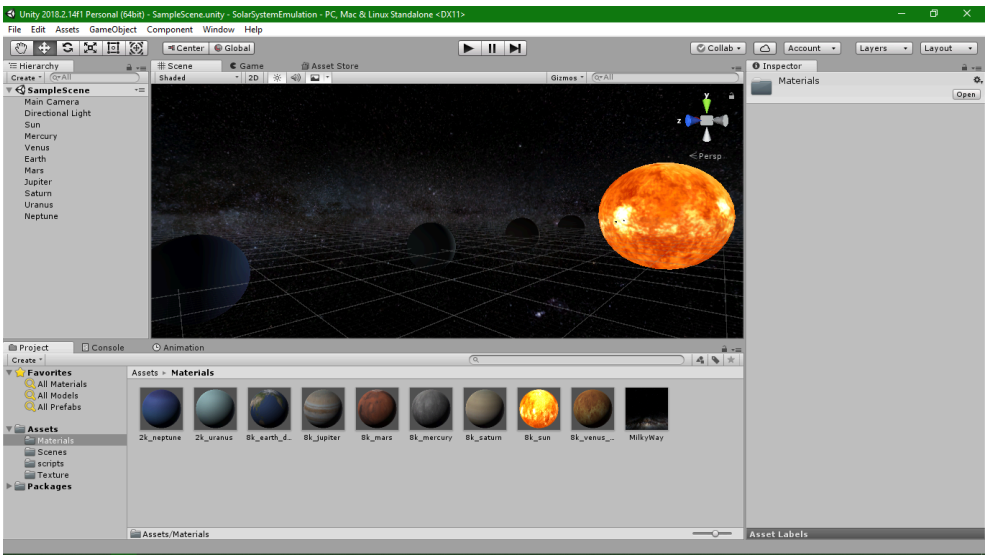

Рис. 3.3. Текстурування графічних об’єктів та оточення

Далі у нашому проєкті ми переходимо до завантаження необхідних текстур знімків поверхні планет сонячної системи та Сонця. Текстури грають важливу роль у створенні реалістичної графіки для об'єктів нашої симуляції. Ми використовуємо текстури для накладання зображень на поверхні об'єктів, щоб вони виглядали як планети та Сонце.

Детальний опис процесу завантаження та накладання текстур:

1. Завантаження текстур: Ми шукаємо відповідні текстури для кожної планети та Сонця. Ці текстури можуть бути взяті з відкритих джерел або створені самостійно. Текстури повинні містити зображення поверхні планети або Сонця в форматі, який підтримується *Unity*.

2. Графічний редактор: Ми використовуємо графічний редактор, такий як *Adobe Photoshop* або *GIMP,* для обробки текстур. Це може включати вирізання зображень, редагування освітлення та кольорів, а також розмір текстур. Графічний редактор дозволяє нам створити текстури, які відображають реалістичний вигляд планет та Сонця.

3. Накладання текстур: Після обробки текстур ми імпортуємо їх до *Unity* та накладаємо на моделі планет. *Unity* має вбудований інструмент для накладання текстур, що дозволяє вибирати, яка текстура буде відображатися на конкретних частинах моделі. Наприклад, ми можемо накласти текстуру Землі на сферу, яка її представляє, зробивши її вигляд реалістичним.

4. Налаштування параметрів текстур: Ми також можемо налаштовувати параметри текстур, такі як поверхневий блиск або прозорість, щоб досягти бажаного ефекту. Налаштування параметрів текстур дозволяє створити більш деталізований та реалістичний вигляд планет.

Після завершення цих кроків ми отримуємо об'єкти в нашому проєкті, які мають текстури, що відображають поверхні планет та Сонця. Це робить нашу симуляцію більш візуально привабливою та допомагає користувачам краще сприймати інформацію про об'єкти сонячної системи.

Наступним кроком створюємо камеру та додаємо до неї базовий користувацький інтерфейс програми (рис. 3.4).

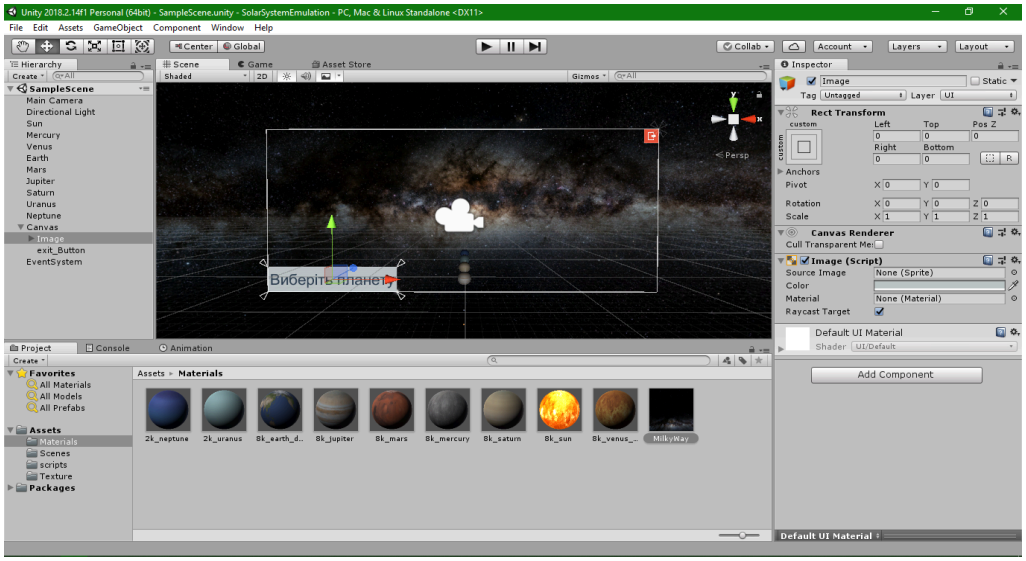

Рис. 3.4. Створення камери ка користувацького інтерфейсу

Наступним важливим етапом розробки нашого проєкту є створення камери та інтерфейсу користувача. Камера в 3*D*-симуляціях відповідає за те, як користувач сприймає світ навколо. Вона дозволяє визначити точку огляду та кут огляду об'єктів у сцені. Ось детальний опис цього етапу:

1. Створення камери: У середовищі *Unity* ми створюємо об'єкт камери. Камера може бути розміщена в будь-якому місці сцени, і це визначає точку огляду для користувача. Важливо вибрати таке розташування та кут огляду, яке найкраще підходить для симуляції сонячної системи.

2. Параметри камери: *Unity* дозволяє налаштовувати різні параметри камери, такі як величина об'єктиву (фокусна відстань), поле зору, плоскість обрію, режими відображення (перспективний або ортографічний) та багато інших. Налаштування цих параметрів дозволяє досягти бажаного ефекту перегляду сцени.

3. Додавання інтерфейсу користувача: Для спрощення взаємодії користувача з симуляцією додаємо інтерфейс користувача. Це може включати в себе кнопки, панелі, текстові поля, вікна та інші елементи управління. На цьому етапі ми також розміщуємо на інтерфейсі інформацію про об'єкти сонячної системи, таку як назви планет, їх характеристики, інструкції для користувача тощо.

4. Логіка камери: Додаємо логіку камери, яка дозволяє користувачеві керувати її положенням та орієнтацією. Наприклад, можливість переміщення камери, обертання та масштабування для більшого зручності спостереження.

5. Логіка інтерфейсу користувача: Програмуємо логіку інтерфейсу користувача, яка реагує на дії користувача. Наприклад, при натисканні на кнопку може відбуватися зміна режиму відображення, відкриття вікон з додатковою інформацією, взаємодія з об'єктами тощо.

6. Тестування та налаштування: Після створення камери та інтерфейсу проводимо тестування, щоб переконатися, що користувач може комфортно користуватися симуляцією та взаємодіяти з інтерфейсом. В разі необхідності вносимо налаштування для поліпшення користувацького досвіду.

Налаштування камери та створення інтерфейсу дозволяють користувачам ефективно взаємодіяти з нашою 3*D*-симуляцією сонячної системи, надаючи їм зручні засоби спостереження та отримання інформації.

Після завантаження та налаштування усієї графічної частини проєкту переходимо до створення необхідних функціональних класів, описаних у діаграмі (рис. 3.1).

Для реалізації обертання планет навколо своєї осі у вашому проєкті було створено клас під назвою "*Rotation*". Цей клас відповідає за керування обертанням об'єктів, таких як планети та Сонце, навколо їхніх власних осей. Давайте детально розглянемо цей процес та його реалізацію:

Створення класу "*Rotation*"

1. Конструктор класу: Першим кроком при створенні класу "*Rotation*" було визначення його конструктора. Конструктор встановлює початкові значення для параметрів обертання ініціалізує об'єкт класу.

2. Змінні для збереження параметрів обертання: Клас "*Rotation*" містить змінні для збереження параметрів обертання, таких як кут обертання і швидкість обертання.

Реалізація обертання

Для обчислення руху об'єкта навколо своєї осі використовується тригонометрична формула обертання. Основні компоненти цієї формули включають:

– Початковий кут (*angle*): Це кут, під яким об'єкт розташований в початковий момент часу. Визначається при ініціалізації об'єкта.

– Швидкість обертання (*angularSpeed*): Швидкість обертання об'єкта, виражена в градусах (або радіанах) на одиницю часу. Це може бути константна швидкість або змінна в залежності від потреби.

– Час (*time*): Час, який пройшов з моменту старту обертання. Це обчислюється у кожному кадрі гри.

– Кут обертання (*rotationAngle*): Це обчислювана величина, яка визначає новий кут обертання об'єкта у кожному кадрі. Вона розраховується як добуток швидкості обертання на час, який пройшов з початку обертання.

– Оновлення кута обертання: Після розрахунку кута обертання в кожному кадрі, новий кут додається до початкового кута, оновлюючи таким чином позицію об'єкта на сцені.

Для того щоб об'єкт обертався навколо своєї осі, екземпляри класу "*Rotation*" додавалися до відповідних об'єктів, таких як планети та Сонце, на сцені. Кожен об'єкт має свій власний екземпляр "*Rotation*", який відповідає за його рух.

Клас "*Rotation*" та відповідний алгоритм обертання дозволяють реалізувати рух об'єктів навколо своєї осі у вашому проєкті. Ця функціональність надає реалістичний вигляд планетам і Сонцю, і додає іммерсивність до симулятора сонячної системи. Реалізація такого обертання допомагає створити візуально привабливий та навчальний досвід для користувачів.Далі для реалізації обертання планет навколо сонця створимо клас *AroundRotation* та додаймо екземпляри цього класу до планет на сцені. Реалізуємо змінні та формулу обертання навколо об’єкта у просторі.

Наведений нижче метод створений для циклічного обертання навколо заданого об’єкта та опрацьовується кожен фрейм програми.

Реалізація обертання та переміщення камери у просторі у залежності від дій користувача за допомогою класу "*CamController*":

Для створення можливості обертання та переміщення камери у просторі на основі взаємодії з користувачем ви використовували клас під назвою "*CamController*". Давайте розглянемо, як цей клас був реалізований та які функції він виконує:

1. Створення класу "*CamController*": Почнемо зі створення самого класу "*CamController*". Цей клас відповідає за управління камерою на сцені.

2. Додавання екземпляра "*CamController*" до основної камери: Щоб камера реагувала на дії користувача, був створений екземпляр класу "*CamController*", який був доданий до основної камери сцени. Це дозволяє цьому об'єкту контролювати положення та орієнтацію камери.

3. Перевірка на натискання клавіш та рух миші: В класі "*CamController*" був розширений метод "*Update*", який викликається кожен кадр гри. У цьому методі виконується перевірка на натискання клавіш та рух миші користувачем.

4. Обмеження активного простору кімнати: Для того щоб обмежити область, в якій може переміщатися камера, в класі "*CamController*" було додано обмеження на положення камери у просторі. Це може включати в себе обмеження по координатах *X, Y* та *Z,* щоб запобігти виходу за межі кімнати або сцени.

5. Реакція на дії користувача у вигляді обертання та руху камери: Після перевірки наявності дій користувача, таких як натискання клавіш або рух миші, в класі "*CamController*" була реалізована логіка для зміни орієнтації та положення камери відповідно до цих дій. Це може включати в себе обертання та зміщення камери у відповідь на дії користувача.

Клас "*CamController*" виконує важливу функцію у вашому проєкті, дозволяючи користувачеві взаємодіяти з камерою для обертання та переміщення в тривимірному просторі сцени. Це надає користувачам можливість досліджувати сонячну систему з різних ракурсів і реалістично відчувати об'єкти на сцені.

Реалізація вибору планети та виводу її назви на графічному інтерфейсі з використанням класу "*Selection*":

Для можливості вибору планети та виводу її назви на графічному інтерфейсі був створений клас під назвою "*Selection*". Давайте розглянемо, як цей клас був реалізований та які функції він виконує:

1. Створення класу "*Selection*": Почнемо зі створення самого класу "*Selection*". Цей клас відповідає за обробку вибору планети користувачем і вивід її назви на графічний інтерфейс.

2. Додавання екземплярів "*Selection*" до інформативних об'єктів (планет): Щоб можна було визначити, яка саме планета була вибрана користувачем, екземпляри класу "*Selection*" були додані до всіх інформативних об'єктів на сцені (планет). Це дозволяє кожній планеті реагувати на дії користувача окремо.

3. Реалізація змінних та передача їх на графічний інтерфейс: Клас "*Selection*" містить змінні, які зберігають інформацію про вибрану планету, таку як назву чи

інші характеристики. При виборі планети ці змінні оновлюються, і ця інформація передається на графічний інтерфейс, щоб вивести її на екрані користувача.

Клас "*Selection*" відіграє важливу роль у вашому проєкті, дозволяючи користувачам вибирати планети та переглядати інформацію про них на графічному інтерфейсі. Це додає інтерактивність та можливість отримувати детальну інформацію про обрані об'єкти сонячної системи, покращуючи досвід користувача.Залишилося створити реакцію на вище наведені події у вигляді обертання та руху камери.

rotX = *rotX* % 360;

rotY = *rotY* % 360;

rotX = *Mathf.Clamp*(*rotX,* -360*,* 360);

rotY = *Mathf.Clamp*(*rotY,* -60*,* 60);

Quaternion qX = *Quaternion.AngleAxis*(*rotX, Vector*3.*up*);

Quaternion qY = *Quaternion.AngleAxis*(*rotY, Vector*3.*left*);

transform.localRotation = *originalRot* * *qX* * *qY*;

Після цього реалізуємо вибір планети та вивід її назви на графічному інтерфейсі. Для цього створимо клас *Selection* та додаймо екземпляри цього класу до усіх інформативних об’єктів (планет) на сцені. Реалізуємо змінні та їх передачу на графічний інтерфейс:

Реалізація підсвічування вибраної планети на сцені:

Для того, щоб вибрана планета на сцені підсвічувалася і користувач міг легко визначити обрану планету, потрібно розширити метод *OnMouseDown*. Нижче наведено кроки, які необхідно виконати для досягнення цієї функціональності:

1. Перевірка наявності попередньо обраної планети: Перш за все, вам потрібно перевірити, чи була вже обрана планета на сцені. Це допоможе уникнути ситуації, коли одночасно обрані дві чи більше планети.

2. Зняття попереднього підсвічування: Якщо на сцені вже є обрана планета, то потрібно зняти підсвічування з попередньої планети, щоб виділити лише нову обрану планету.

3. Підсвічування обраної планети: Після цього потрібно налаштувати візуальний ефект (наприклад, зміну кольору чи відсвічування) для виділення обраної планети. Це може включати зміну матеріалу, колір рамки або інший ефект, який дозволяє виділити планету.

4. Оновлення змінних в класі "*Selection*": Після того, як користувач обрав планету, ви повинні оновити змінні в класі "*Selection*", щоб вони відображали інформацію про нову обрану планету.

5. Відображення інформації на графічному інтерфейсі: Нарешті, змінену інформацію про обрану планету слід відобразити на графічному інтерфейсі, наприклад, вивести назву планети на екран користувача або в іншому вигляді.

Після налаштування взаємодії створених екземплярів класів логіки коригуємо змінні для отримання максимально реалістичної фізичної моделі поведінки (рис. 3.5).

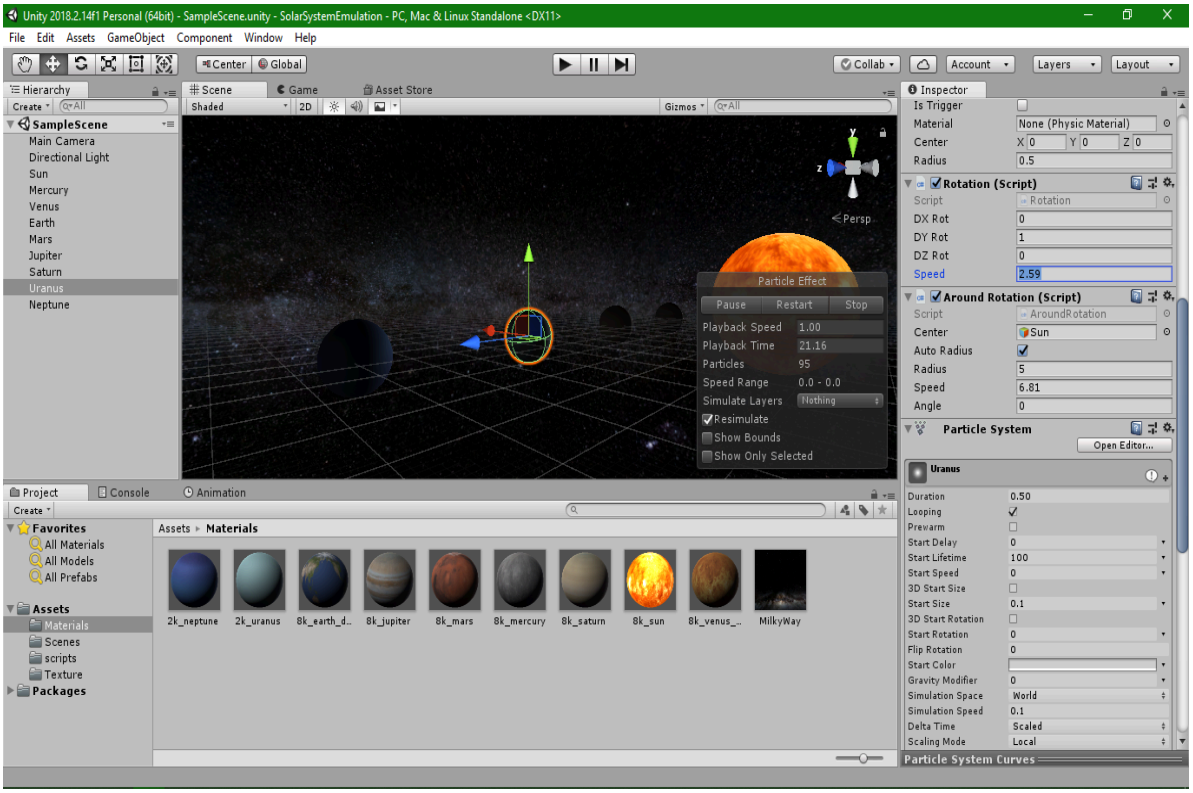

Рис. 3.5. Налаштування фізичних характеристик логіки поведінки об’єктів

Після створення та налаштування класів для імітації руху об'єктів у тривимірному просторі важливо коригувати змінні та параметри, щоб досягнути

максимально реалістичної фізичної моделі поведінки. Це може включати в себе наступні кроки:

1. Фізичні параметри об'єктів: Для кожного об'єкта, такого як планета чи Сонце, важливо визначити фізичні параметри, такі як маса, радіус, гравітаційна константа тощо. Ці параметри впливають на рух об'єктів відповідно до законів гравітації та інших фізичних законів.

2. Модель гравітації: Для реалістичного моделювання гравітації об'єктів використовуються формули та алгоритми, які враховують масу та відстань між об'єктами. Це може включати в себе закон Ньютона про всесвітальну гравітацію та інші фізичні закони.

3. Рух та траєкторія: Важливо налаштувати параметри руху об'єктів, такі як швидкість та напрямок. Реалістична траєкторія об'єкта повинна враховувати гравітацію та інші сили, що впливають на нього.

4. Взаємодія об'єктів: Якщо на сцені присутні різні об'єкти, такі як планети чи інші об'єкти сонячної системи, то важливо визначити їхню взаємодію. Наприклад, об'єкти можуть взаємодіяти гравітаційно, створюючи зміни у русі один одного.

5. Оптимізація та реалізм: На останньому етапі важливо оптимізувати фізичну модель так, щоб вона була реалістичною, але при цьому не надто важкою для обчислень. Реалістичність та продуктивність повинні бути збалансовані.

Коригування змінних для отримання максимально реалістичної фізичної моделі поведінки є важливим етапом у розробці 3*D* симуляторів. Це допомагає забезпечити, що об'єкти ведуть себе відповідно до фізичних законів і надають користувачам реалістичний досвід взаємодії з ними.

Основний алгоритм роботи програми з візуалізації руху об’єктівможна представити у вигляді схеми алгоритму (рис. 3.6).

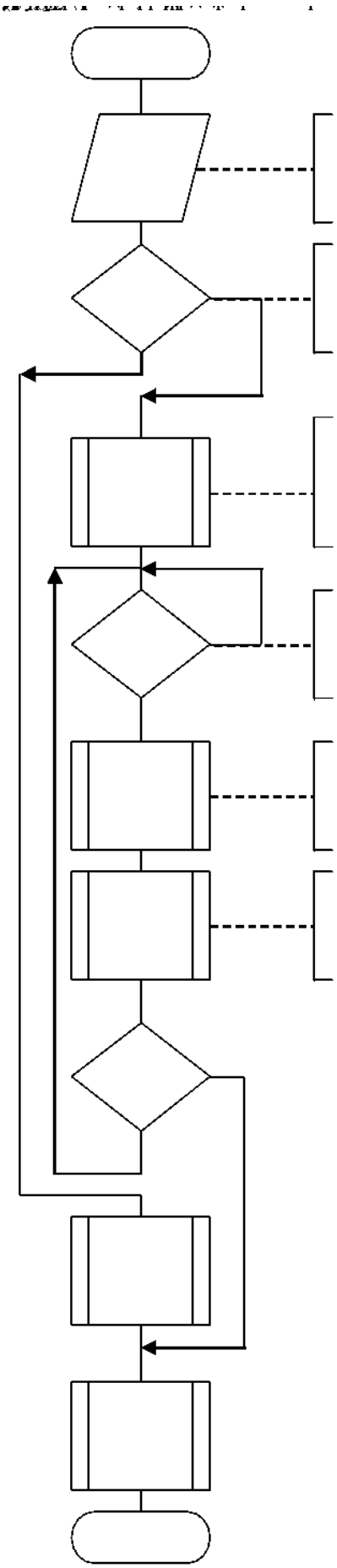

Рис. 3.6. Схема алгоритму руху об’єктів

3.2. Керівнитво користувача

Структура та розмір програмного пакету:

Програмний пакет, який реалізує симулятор сонячної системи в середовищі *Unity* 2018 та відповідному фреймворку для C# в програмному середовищі *Visual Studio* 2017, має наступну структуру та параметри:

1. Формат файлів: Весь програмний пакет поставляється у вигляді архіву з розширенням *.rar* (*Roshal Archive*). Цей формат архіву дозволяє компактно об'єднати різні файли та теки для зручного розповсюдження та зберігання.

2. Розмір архіву: Розмір архіву програмного пакету становить 48 мегабайтів (МБ). Це є розміром файлу, який необхідно завантажити для отримання доступу до програми.

3. Розмір розпакованого стану: Після розпакування архіву в розпакованому стані, програма займає більше місця на диску. Розмір розпакованого програмного пакету становить 176 мегабайтів (МБ). Це означає, що користувач повинен мати на диску цю кількість вільного місця для встановлення та використання програми.

4. Операційна система: Програмний пакет створено для середовища операційної системи *Windows*. Це означає, що він призначений для використання на комп'ютерах, які працюють під управлінням операційних систем *Windows,* таких як *Windows* 7*, Windows* 8*, Windows* 10 тощо.

Зазначений програмний пакет є компактним, але вимагає наявності вільного місця на диску для розпакування та використання. Він спеціально розроблений для операційних систем *Windows* та містить усі необхідні файли та ресурси для функціонування симулятора сонячної системи в середовищі *Unity* 2018.

Програмні й апаратні вимоги:

– операційна система *Windows* 7 / 8 / 8.1 / 10;

– графічний драйвер *DirectX* 11 і вище;

– 2 *Gb* оперативної пам’яті комп’ютера;

– *Microsoft .NET Framework* 3.7 і вище.

Для використання програми необхідно розпакувати застосунок з архіву «*SolarSystemEmulation.rar*» до зручної вам папки. При умові належного програмного та апаратного забезпечення запускаємо виконуваний файл *SolarSystemEmulation.exe* для відкриття меню підготовки програми (рис. 3.7).

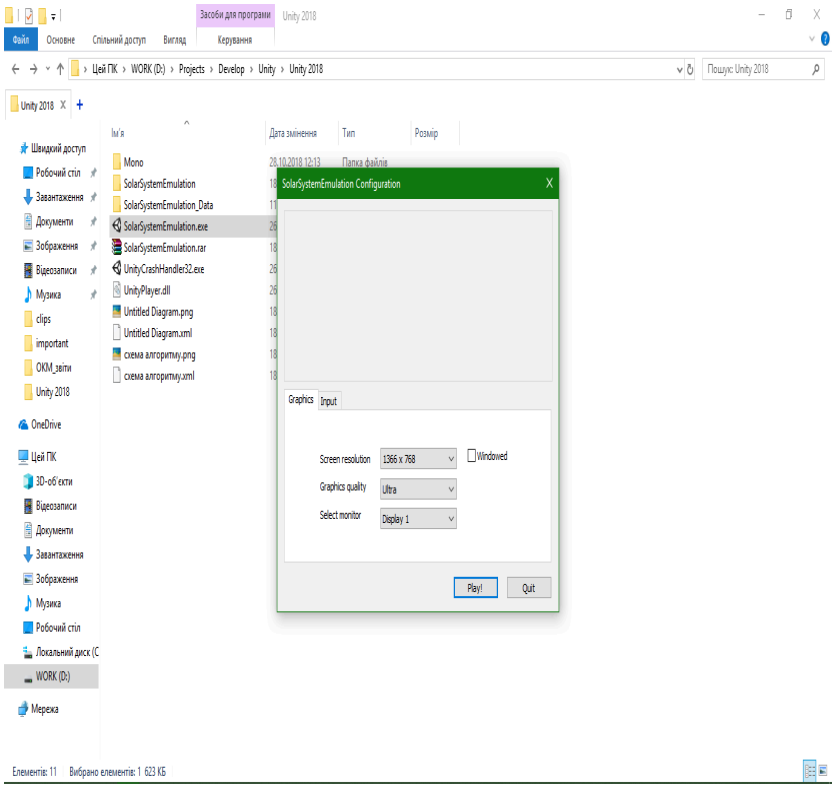

Рис. 3.7. Меню підготовки програми

Для успішного використання програми "Симулятор сонячної системи" необхідно дотримуватися наступних кроків:

1. Завантаження програмного пакету:

– Отримайте програмний пакет у вигляді архіву з назвою "*SolarSystemEmulation.rar*". Цей архів містить всі необхідні файли та ресурси для програми.

2. Розпакування програмного пакету:

– Створіть нову теку або виберіть існуючу, в яку ви бажаєте розпакувати програму.

– Відкрийте архів "*SolarSystemEmulation.rar*" за допомогою програми архівації, такої як *WinRAR* або 7-*Zip*.

– Розпакуйте вміст архіву до обраної теки. Після цього у вас повинно з'явитися кілька файлів та папок, які містять програму.

3. Запуск програми:

– Перейдіть до теки, в яку ви розпакували програму "Симулятор сонячної системи".

– Знайдіть виконуваний файл з назвою "*SolarSystemEmulation.exe*". Цей файл призначений для запуску програми.

4. Меню підготовки програми:

– Після запуску виконуваного файлу "*SolarSystemEmulation.exe*" відкриється меню підготовки програми (рис. 3.7).

– У цьому меню ви зможете вибрати параметри запуску програми, налаштувати графічні налаштування, визначити параметри симуляції сонячної системи тощо.

5. Запуск симуляції:

– Після виконання налаштувань у меню підготовки програми, натисніть відповідну кнопку або опцію для запуску симуляції сонячної системи.

– Ви побачите відкритий симулятор, де ви зможете спостерігати за об'єктами сонячної системи та взаємодіяти з ними.

6. Використання програми:

– Після запуску симулятора ви зможете спостерігати за рухом планет, їхньою візуалізацією та вивчати різні параметри сонячної системи в реальному часі.

– Інтерактивний інтерфейс дозволить вам взаємодіяти з об'єктами сонячної системи та отримувати інформацію про них.

7. Завершення роботи:

– Після закінчення використання програми закрийте її за допомогою відповідного функціоналу програми або виходу з неї.

Це загальний опис процедури завантаження, розпакування та використання програми "Симулятор сонячної системи". За допомогою цієї програми ви зможете вивчати рух об'єктів сонячної системи та багато іншого.

У цьому меню можна вибрати бажане розширення екрану в списку «*Screen resolution*», рівень деталізації графіки в списку «*Graphics quality*», один з підключених моніторів для виведення програми з списку «*Select monitor*» та режим роботи програми (віконний чи повноекранний) активуючи поле «*Windowed*». Всі ці налаштування створені для більшої адаптації по потужності персонального комп'ютера та комфортної роботи з програмою. У випадку зависання достатньо понизити ці характеристики, щоб відновити праце-здатність.

По завершенню налаштування необхідно натиснути на кнопку «*Play*!» розміщену в нижньому лівому куті. Застосувавши зміни відкриється вікно самої емуляції (рис. 3.8).

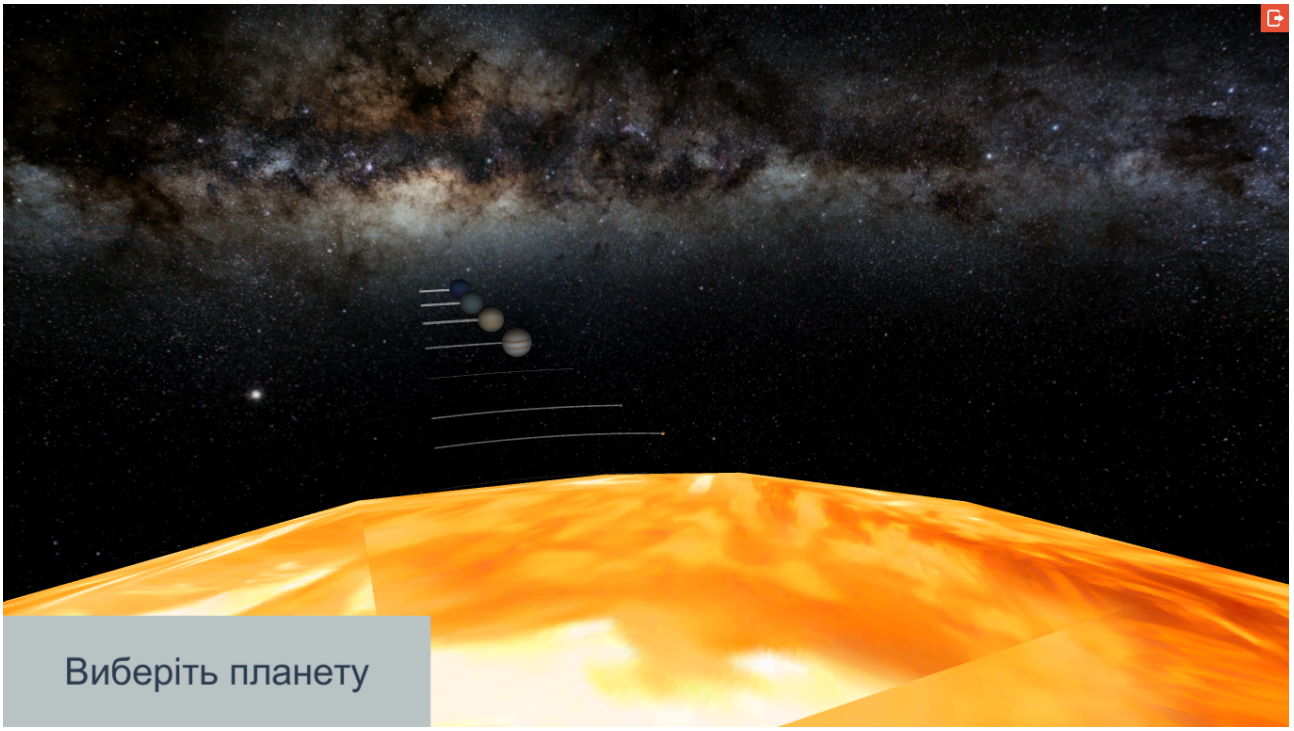

Рис. 3.8. Вікно програми емуляції сонячної системи після запуску

У меню підготовки програми "Симулятор сонячної системи" вам надається можливість налаштувати різні параметри для оптимізації роботи програми та забезпечення комфортного користувацького досвіду. Давайте розглянемо кожен із параметрів та їх призначення:

1. *Screen resolution* (Розширення екрану): В цьому параметрі можна вибрати бажане розширення екрану для відображення програми. Вибір розширення залежить від вашого монітора та особистих вподобань.

2. *Graphics quality* (Рівень деталізації графіки): Цей параметр дозволяє налаштувати рівень деталізації графіки у програмі. Ви можете вибрати відповідний рівень якості графіки в залежності від потужності вашого комп'ютера. Вищий рівень деталізації може вимагати більше обчислювальних ресурсів.

3. *Select monitor* (Вибір монітора): Якщо у вас підключено декілька моніторів, ви можете вибрати, на якому саме моніторі ви бажаєте відображати програму. Це корисно, коли ви працюєте з багатьма моніторами.

4. *Windowed* (Віконний режим): Цей параметр дозволяє вибрати режим роботи програми – віконний або повноекранний. Якщо ви активуєте цей параметр (поставите позначку), програма буде працювати у віконному режимі, і ви зможете зручно перемикатися між програмою та іншими додатками.

Після налаштування цих параметрів, вам потрібно натиснути на кнопку "*Play*!" (Грати!), яка розташована в нижньому лівому куті меню. Після цього програма запуститься з вибраними налаштуваннями. Вікно самої емуляції з'явиться на вашому екрані, і ви зможете почати спостерігати за рухом об'єктів сонячної системи та взаємодіяти з ними.

Ці параметри створені для того, щоб забезпечити комфортну роботу з програмою на різних комп'ютерах і з різними налаштуваннями. Якщо програма запускається повільно або виникають зависання, ви можете спробувати знизити рівень деталізації графіки або змінити інші налаштування, щоб покращити продуктивність.

Після короткого очікування завантаження, ви побачите емуляцію сонячної системи та користувацький інтерфейс програми. Це вікно стане вам доступним для взаємодії.

Для того, щоб керувати камерою та спостерігати за об'єктами сонячної системи, ви можете використовувати інтерактивний вказівник "Миша" або сенсорну панель мультимедійного пристрою (якщо ви використовуєте пристрій з сенсорним екраном).

1. Поворот камери за допомогою миші: Камера постійно буде рухатися, слідкуючи за курсором миші. Це дозволить вам обертати камеру в потрібному напрямку для детального огляду об'єктів сонячної системи.

2. Поворот камери за допомогою клавіші "Миша": Якщо у вас немає можливості використовувати мишу або сенсорний екран, ви можете використовувати клавіші стрілочок або літери на клавіатурі для переміщення позиції камери у просторі. Використовуйте клавіші "↑" (уверх), "↓" (униз), "←" (вліво) та "→" (вправо) для обертання камери у відповідних напрямках. Використання літер "*W*", "*S*", "*A*" та "*D*" дасть аналогічний результат.

Ці способи управління камерою дозволять вам вільно переміщатися та спостерігати за рухом об'єктів сонячної системи у програмі. Таким чином, ви зможете досліджувати цікавий та реалістичний симулятор сонячної системи в реальному часі.Для вибору планети та отримання інформації про неї необхідно навести на вибране небесне тіло курсор миші та натиснути ліву кнопку. У інтерфейсі з’явиться інформація про вибране космічне тіло, а воно саме буде виділено зеленим світінням (рис. 3.9).

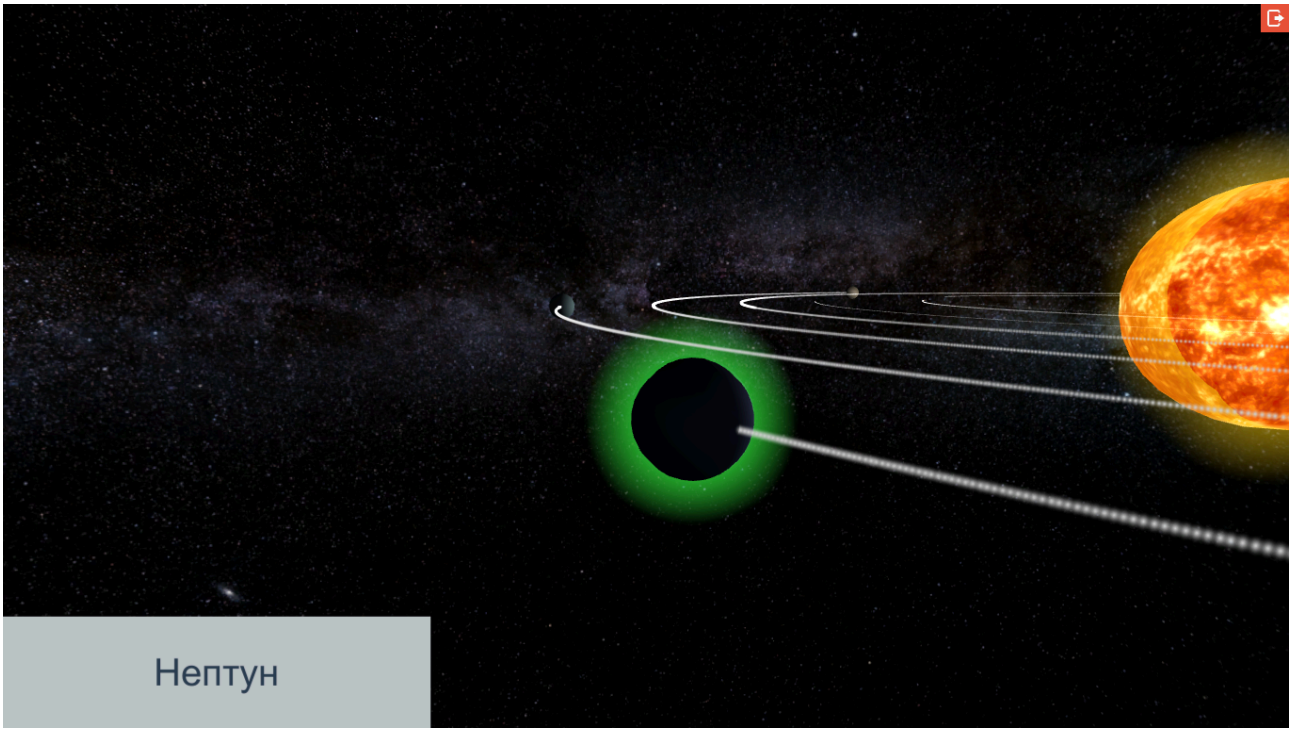

Рис. 3.9. Реакція програми при виборі космічного тіла

Для того, щоб вибрати планету та отримати інформацію про неї у програмі, вам необхідно виконати наступні дії:

1. Наведення курсору на небесне тіло: Проведіть курсор миші (або використовуйте сенсорний екран) над бажаною планетою або іншим об'єктом сонячної системи на екрані. Курсор миші повинен бути розташованим над цим об'єктом.

2. Натисніть ліву кнопку миші: Після наведення курсору на планету або об'єкт сонячної системи, натисніть ліву кнопку миші (ліва кнопка миші - це зазвичай основна кнопка миші). Після цього в інтерфейсі з'явиться інформація про вибране космічне тіло.

3. Інформація та виділення об'єкта: Після натискання на об'єкт сонячної системи, ви побачите відповідну інформацію про це тіло на інтерфейсі програми. Крім того, об'єкт буде виділений зеленим світінням на сцені, що допоможе вам легко визначити вибраний об'єкт серед інших.

Ця функція дозволяє вам отримувати інформацію про будь-яку планету чи об'єкт сонячної системи, з якою ви бажаєте поближче ознайомитися, і вивчати деталі про нього в інтерфейсі програми.

Можливості *Unity*3*D* відкривають широкий спектр інноваційних можливостей для візуалізації руху об'єктів у 3*D* симуляторі сонячної системи. Нижче розглянуті деякі вражаючі функції, які можна реалізувати за допомогою цієї платформи:

1. Спостереження за тілами сонячної системи: Завдяки *Unity*3*D* можна створити деталізовані 3*D*-моделі планет, місяців, астероїдів та інших об'єктів сонячної системи. Ці моделі можуть бути вірними репліками реальних об'єктів, які відображають їхні розміри, форму та текстури. Користувачеві дозволяється спостерігати за цими об'єктами у великій деталізації, що надає реалістичного відчуття дослідження сонячної системи.

2. Реалістична анімація руху об'єктів: *Unity*3*D* надає потужний інструментарій для створення анімацій руху планет і інших об'єктів. Можливо реалізувати точне моделювання орбітального руху, обертання навколо власної осі та інші аспекти руху,

відповідно до законів гравітації та фізики. Це дозволяє користувачам бачити, як планети обертаються навколо Сонця, місяці навколо планет, і так далі.

3. Інтерактивність і управління: *Unity*3*D* дозволяє створювати інтерактивні симулятори, де користувачі можуть керувати об'єктами та взаємодіяти з ними. Наприклад, вони можуть змінювати швидкість обертання планети, зміщувати камеру для докладного вивчення певних областей або навіть моделювати зіткнення астероїдів.

4. Графічні ефекти та освітлення: *Unity*3*D* має потужний графічний рушій, який дозволяє застосовувати різноманітні спеціальні ефекти, такі як відблиски світла, тіні та обробка текстур. Це допомагає підвищити реалістичність симуляції і надає об'єктам сонячної системи більш природний вигляд.

5. Можливість розширеного вивчення: за допомогою *Unity*3*D* можливо створити можливості для розширеного вивчення сонячної системи. Наприклад, можна додати інтерактивні підказки, навігаційні маркери та інші допоміжні засоби, які сприяють зрозумінню та освоєнню інформації.

6. Можливості віртуальної реальності (*VR*): *Unity*3*D* підтримує віртуальну реальність, що дозволяє користувачам досліджувати сонячну систему в *VR*-середовищі. Це створює надзвичайно іммерсивний досвід і робить навчання більш захоплюючим.

7. Підтримка різних платформ: *Unity*3*D* дозволяє створювати програми для різних платформ, включаючи *Windows*, *macOS*, *Android* і *iOS*. Це дозволяє забезпечити доступність симулятору для широкого кола користувачів.

8. Можливості розвитку і розширення: *Unity*3*D* надає можливості для розвитку та розширення симулятора. Розробники можуть додавати нові об'єкти, покращувати графіку, вдосконалювати фізичні моделі та розширювати функціональність, щоб відповідати зростаючим вимогам користувачів.

Загалом, *Unity*3*D* відкриває безмежні можливості для створення захоплюючих та освітніх 3*D*-симуляторів сонячної системи, які дозволяють користувачам досліджувати космос у віртуальному середовищі.

Також можна оцінювати та порівнювати дійсні швидкості руху планет, а також їх кут та швидкість обертання навколо своєї осі (рис.3.10).

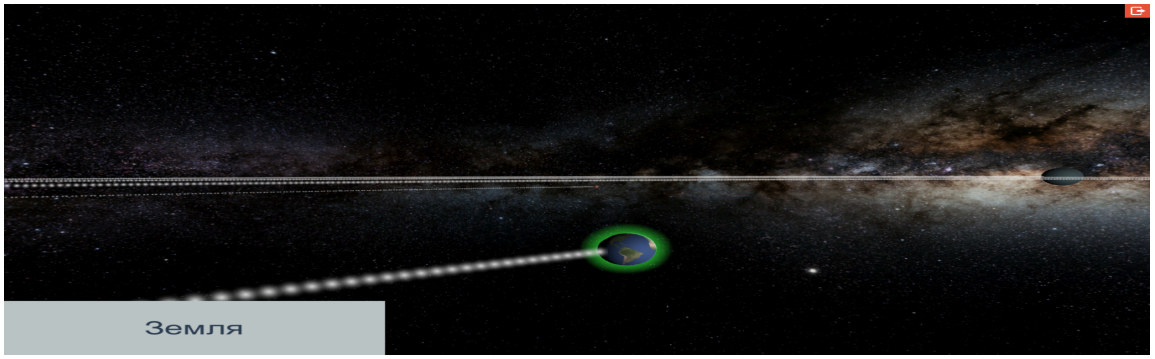

Рис. 3.10. Кут та обертання Землі навколо своєї осі

Можливості *Unity*3*D* дозволяють реалізувати вражаючі функції для візуалізації руху об'єктів у 3*D* симуляторі сонячної системи:

1. Спостереження за тілами сонячної системи: Завдяки *Unity*3*D* можна створити деталізовані 3*D* моделі планет, місяців, астероїдів і інших об'єктів сонячної системи. Користувач може спостерігати за цими об'єктами у великій деталізації.

2. Оцінка розмірів об'єктів: *Unity*3*D* дозволяє відтворити масштабні співвідношення між об'єктами сонячної системи, що дозволяє користувачеві отримувати реалістичне уявлення про їхні розміри та відстані між ними.

3. Відображення орбіт планет: За допомогою *Unity*3*D* можна створити орбіти планет та інших об'єктів, і користувач може спостерігати, як вони рухаються вздовж своїх орбіт, отримуючи уявлення про їхні шляхи навколо Сонця.

4. Швидкість руху планет: *Unity*3*D* дозволяє налаштовувати швидкості руху планет та інших об'єктів, щоб користувач міг спостерігати реалістичний рух об'єктів сонячної системи відповідно до їхніх реальних значень.

5. Кут та швидкість обертання: Використовуючи *Unity*3*D*, можна налаштовувати кут та швидкість обертання планет навколо їхніх осей. Користувач може спостерігати, як планети обертаються, і порівнювати це з реальними даними.

Ці можливості роблять візуалізацію сонячної системи у 3*D* симуляторі за допомогою *Unity*3*D* дуже привабливою для навчальних та наукових цілей, а також для розваг і досліджень.

Функції та можливості середовища *Unity*3*D* для візуалізації об'єктів сонячної системи:

1. Створення 3*D*-моделей: *Unity*3*D* дозволяє створювати 3*D*-моделі планет, супутників, зірок, астероїдів та інших об'єктів сонячної системи.

2. Анімація руху: Можливість анімувати рух об'єктів, включаючи обертання планет навколо своєї осі та обертання навколо сонця.

3. Фізичне моделювання: *Unity*3*D* має фізичні двигуни, які дозволяють моделювати гравітаційні сили, взаємодію об'єктів та їхні траєкторії.

4. Управління камерою: Ви можете контролювати камеру для перегляду об'єктів сонячної системи з різних точок зору.

5. Відображення орбіт: Можливість відображати орбіти об'єктів для ілюстрації їхнього руху.

6. Текстури і матеріали: Використовуйте тексти і матеріали для створення реалістичного вигляду планет, місяців та інших об'єктів.

7. Інтерактивність: Створюйте інтерактивні сценарії, де користувачі можуть взаємодіяти з об'єктами сонячної системи, натискаючи на них або отримуючи додаткову інформацію.

8. Симуляція подорожей: Розробляйте симуляції подорожей через сонячну систему, дозволяючи користувачам обирати маршрути та спостерігати за астрономічними явищами.

9. Зображення великих відстаней: *Unity*3*D* відображає великі відстані в космосі та масштаби космічних об'єктів.

10. Інтеграція з додатковими ресурсами: Імпортуйте дані з астрономічних джерел для точності інформаційних симуляцій.

Ці функції дозволяють створювати навчальні додатки та симуляції для вивчення сонячної системи та астрономічних процесів.

Розглянемо компоненти *Unity*3*D,* які можна використовувати для реалізації різних функцій візуалізації об'єктів сонячної системи:

1. Створення 3*D*-моделей:

– *Mesh Renderer*: Дозволяє відображати 3*D*-моделі, задаючи їм матеріали і текстури.

– 3*D*-моделі: Можна створювати або імпортувати 3*D*-моделі планет, супутників і інших об'єктів.

2. Анімація руху:

– *Animator*: Дозволяє створювати анімації руху, такі як обертання планет навколо своєї осі або рух навколо сонця.

– *Rigidbody*: Використовується для симуляції фізики руху об'єктів в просторі.

3. Фізичне моделювання:

– *Physics*: Включає фізичні компоненти для моделювання гравітаційних сил, зіткнень і траєкторій руху.

4. Управління камерою:

– *Camera*: Компонент для керування камерою, визначення її позиції і орієнтації.

– *Cinemachine*: Додатковий плагін для створення складних камерних рухів і переходів.

5. Відображення орбіт:

– *Line Renderer*: Дозволяє малювати лінії, які можуть використовуватися для відображення орбіт.

6. Текстури і матеріали:

– *Material*: Визначає зовнішній вигляд об'єкта, включаючи кольори, текстури і властивості відображення.

– *Texture*: Використовується для нанесення текстур на 3*D*-моделі.

7. Інтерактивність:

– *Collider*: Дозволяє об'єктам реагувати на взаємодію з іншими об'єктами.

– *Scripts* (*C#* або *JavaScript*): Для написання скриптів для обробки подій та взаємодії користувача.

8. Симуляція подорожей:

– *UI Elements*: Дозволяє створювати елементи інтерфейсу, такі як кнопки або меню, для керування симуляцією.

9. Зображення великих відстаней:

– *Skybox*: Використовується для створення космічного фону та відображення великих відстаней.

10. Інтеграція з додатковими ресурсами:

– *Asset Importer*: Дозволяє імпортувати дані та моделі з зовнішніх джерел.

Цей підхід до візуалізації 3*D* об'єктів в симуляторах має ряд суттєвих переваг, які варто виділити:

1. Зручність розробки: Використання готового ігрового двигуна, такого як *Unity,* спрощує процес розробки і дозволяє зосередитися на важливих аспектах візуалізації та функціональності симулятора. Розробникам не потрібно вигадувати "велосипед" і можуть використовувати вже існуючі ресурси та інструменти.

2. Широкі можливості візуалізації: *Unity* має потужну систему графіки, яка дозволяє створювати реалістичні та деталізовані 3*D* моделі, додавати різні текстури, ефекти освітлення та тіней, що робить візуалізацію об'єктів більш живою та привабливою.

3. Інтерактивність: Використання *Unity* дозволяє створювати інтерактивні симулятори, де користувачі можуть взаємодіяти з об'єктами сцени, змінювати параметри руху та спостерігати за наслідками своїх дій. Це робить процес навчання або дослідження більш ефективним та цікавим.

4. Можливості навчання: Створені симулятори на основі *Unity* можуть використовуватися для навчальних цілей у вищих навчальних закладах, школах та навчальних центрах. Вони надають змогу студентам та учням вивчати складні наукові концепції шляхом візуального моделювання.

5. Масштабованість: *Unity* дозволяє легко розширювати та модернізувати симулятори в майбутньому. Нові функції та об'єкти можуть бути додані без значних труднощів, що робить систему дійсно масштабованою.

6. Підтримка спільноти: *Unity* має велику та активну спільноту розробників. Це означає, що завжди є можливість знайти відповіді на питання або отримати поради щодо розробки.

У кваліфікаційній роботі було зроблено значний крок у напрямку створення сучасних симуляторів з використанням ігрового двигуна *Unity* 2018 та відповідного фреймворку на мові програмування *C#*. Для візуалізації 3*D* об'єктів та імітації руху об'єктів були реалізовані нові підходи, які спрямовані на поліпшення функціональності та ефективності симулятора.

1. Реалістична візуалізація об'єктів: Однією з ключових особливостей є реалістична візуалізація об'єктів сонячної системи. Були створені 3*D* моделі планет, які відображають їх дійсний розмір та форму. Кожна планета має власну текстуру та ефекти освітлення, що дозволяє спостерігачам спостерігати за ними в реальному часі та оцінювати їх реалістичність.

2. Відображення орбіт та руху об'єктів: Було реалізовано систему відображення орбіт планет та їх руху в навколосонячному просторі. Користувачі можуть слідкувати за дійсним відображенням орбіт, оцінювати їхні траєкторії та пройдений шлях. Цей підхід надає можливість кращого розуміння космічних рухів та їх властивостей.

3. Розрахунок дійсних параметрів планет: В симуляторі було включено можливість оцінювання та порівняння дійсних швидкостей руху планет, а також їх кутів та швидкостей обертання навколо власної осі. Це дозволяє користувачам навчатися та досліджувати природні явища в сонячній системі.

4. Інтерактивний дослід: Поєднання можливостей *Unity* і *C#* дозволило створити інтерактивний симулятор, де користувачі можуть взаємодіяти з планетами та спостерігати за їх рухом. Це створює можливості для навчання та дослідження, де користувачі можуть вивчати природні процеси в сонячній системі за допомогою власних дій.

5. Можливість розширення: Розроблений симулятор має потенціал для подальшого розширення та модернізації. Нові об'єкти, ефекти та функції можуть бути легко додані до симулятора без великих труднощів завдяки гнучкості *Unity*.

Отже, розроблений симулятор відзначається своєю реалістичною візуалізацією, можливістю вивчати та аналізувати рух об'єктів сонячної системи в реальному часі, інтерактивністю та можливістю подальшого розширення. Цей підхід

відкриває нові можливості для освіти, дослідження та розвитку сучасних симуляторів.

У процесі розробки проекту "Симулятор сонячної системи" відзначається використання ігрового двигуна *Unity* 2018 та спеціалізованого фреймворку для мови програмування *C#*, що дозволило досягти високої функціональності та реалістичності симулятора.

Фреймворк *Unity* вже містив деякі важливі компоненти для реалізації інтерактивних сцен, такі як клас *GameObject* для представлення графічних об'єктів на сцені та клас *MonoBehavior* для реалізації їхньої логіки. Однак для досягнення головної мети проекту, а саме імітації реалістичного руху об'єктів в тривимірному просторі сонячної системи, було необхідно розробити додаткові класи.

Основними компонентами, які були реалізовані під час розробки, є:

1. Класи для імітації руху об'єктів: Для кожного об'єкта (планети, сонця тощо) були створені окремі класи, які визначали їхні параметри та рух в просторі. Ці класи включали в себе розрахунки шляху руху об'єкта відповідно до законів гравітації та дійсних параметрів планет. Вони також забезпечували можливість контролю за обертанням об'єктів навколо власної осі.

2. Моделі планет та об'єктів: Для кожної планети була створена 3*D* модель, яка реалістично відображала її розмір та форму. Моделі були обладнані текстурами та ефектами освітлення для досягнення максимальної реалістичності.

3. Система відображення орбіт: Для відображення орбіт планет була створена система, яка динамічно розраховувала шлях руху об'єктів та показувала їхні траєкторії на сцені. Це додавало велику педагогічну цінність симулятору.

4. Інтерактивність та можливість взаємодії: Розроблено інтерактивний інтерфейс, який дозволяє користувачам взаємодіяти з об'єктами сонячної системи. Вони можуть вибирати планети, збільшувати та зменшувати масштаб, а також вивчати параметри руху та обертання.

Цей підхід до розробки симулятора сонячної системи дозволив досягти високого ступеня реалістичності та інтерактивності, що робить його ефективним інструментом для навчання та дослідження астрономії. Крім того, розроблені

компоненти можуть бути використані в подальших проектах для створення подібних симуляторів та навчальних програм.

Окрім розробленого симулятора сонячної системи, існує широкий спектр сучасних методів і засобів 3*D*-візуалізації, які можна використовувати при впровадженні систем дистанційного навчання. Ці методи та засоби сприяють покращенню процесу навчання, підвищенню зрозуміння складних концепцій і полегшенню доступу до освіти. Ось кілька з них:

1. Використання віртуальної реальності (*VR*) та розширеної реальності (*AR*): *VR* та *AR* надають можливість навчання в іммерсивному середовищі, де студенти можуть взаємодіяти з об'єктами та сценами в тривимірному просторі. Це особливо корисно для навчання, пов'язаного з просторовими концепціями, наприклад, геометрією або архітектурою.

2. Інтерактивні веб-середовища: Веб-додатки та платформи для віддаленого навчання стають все більш інтерактивними та візуально привабливими. Їхні можливості включають інтерактивні діаграми, анімацію та 3*D*-моделі, що полегшує зрозуміння складних тем.

3. Системи імітації: Подібно до симулятора сонячної системи, системи імітації дозволяють створювати інтерактивні симуляції різних явищ та процесів. Це може бути корисним для навчання фізики, хімії, біології та інших наук.

4. Графічні двигуни та бібліотеки: Використання сучасних графічних двигунів і бібліотек дозволяє створювати реалістичні 3*D*-середовища для навчання. *Unity, Unreal Engine* та *Blender* - це приклади інструментів, які можна використовувати для створення 3*D*-моделей та сцен для освітніх цілей.

5. Інтерактивні тестові завдання: За допомогою 3*D*-візуалізації можна створювати інтерактивні тестові завдання, де студенти повинні взаємодіяти з об'єктами або розв'язувати завдання в тривимірному середовищі.

6. Засоби анімації та відображення даних: 3*D*-графіка може бути використана для анімації наукових явищ та відображення даних у більш доступному та зрозумілому вигляді.

7. Колаборативні середовища: Платформи для дистанційного навчання можуть включати функції спільної роботи над 3*D*-сценами, де студенти можуть спільно вивчати та розробляти проекти.

Застосування цих методів та засобів може покращити результативність систем дистанційного навчання, роблячи процес навчання більш інтерактивним і зрозумілим для студентів. Такі підходи сприяють розвитку більш доступної та ефективної освіти, дозволяючи студентам легше освоювати складні концепції та набувати практичних навичок.

Завдяки можливостям *Unity*3*D*, користувачі мають змогу спостерігати за тілами сонячної системи та оцінювати їх реальний розмір та розташування відносно один одного у віртуальному середовищі. Ця можливість надає величезні переваги для освіти та наукового дослідження. Давайте докладніше розглянемо, які конкретні можливості доступні:

1. Оцінка розміру тіл сонячної системи: *Unity*3*D* дозволяє створювати деталізовані 3*D*-моделі планет, місяців, астероїдів і інших об'єктів сонячної системи. Користувачі можуть буквально «підійти» до цих об'єктів у віртуальному середовищі і оцінити їхній розмір в порівнянні один з одним. Наприклад, порівняти розмір Землі з розміром Юпітера, що дозволяє краще розуміти масштаби в сонячній системі.

2. Відображення реальних орбіт планет: Віртуальний симулятор може точно відображати орбіти планет навколо Сонця. Це дозволяє користувачам спостерігати, як планети рухаються по своїх орбітах та пройдений ними шлях. Ця функція надає можливість відслідковувати дійсний рух планет у віртуальному середовищі.

3. Оцінка швидкості руху планет: *Unity*3*D* дозволяє відобразити дійсні швидкості руху планет. Користувачі можуть спостерігати, як швидко рухається планета по своїй орбіті і порівнювати ці швидкості між різними планетами. Це сприяє кращому розумінню законів руху в космосі.

4. Оцінка кута та швидкості обертання планет: Користувачі також можуть оцінювати кут та швидкість обертання планет навколо своєї осі. Наприклад, вони можуть спостерігати, як швидко Земля обертається навколо своєї вісі та порівнювати це з обертанням інших планет, які можуть обертатися повільніше або швидше.

Загалом, *Unity3D* дозволяє створити візуально вражаючу та інтерактивну симуляцію сонячної системи, яка дозволяє користувачам не лише вивчати основи астрономії, але й глибше розуміти рух об'єктів у космосі в реальному часі. Це корисний інструмент для освіти та досліджень у галузі астрономії.

Після натискання на кнопку виходу процес емуляції сонячної системи буде завершено, і програма повністю вимкнеться. Важливо зауважити, що розташування об'єктів у симуляції не запам'ятовується між сеансами використання програми. Тобто, при наступному запуску програми емуляція сонячної системи розпочнеться знову з того ж самого моменту, що відповідає такому явищу, як "Парад планет".

Для повторного використання програми просто запустіть виконуваний файл і пройдіть всі необхідні кроки експлуатації з початку. Таким чином, ви можете насолоджуватися емуляцією сонячної системи та досліджувати її різні аспекти при кожному запуску програми.

По завершенню використання програми емуляції достатньо навести курсор на червону кнопку виходу та натиснути ліву кнопку миші (рис. 3.11).

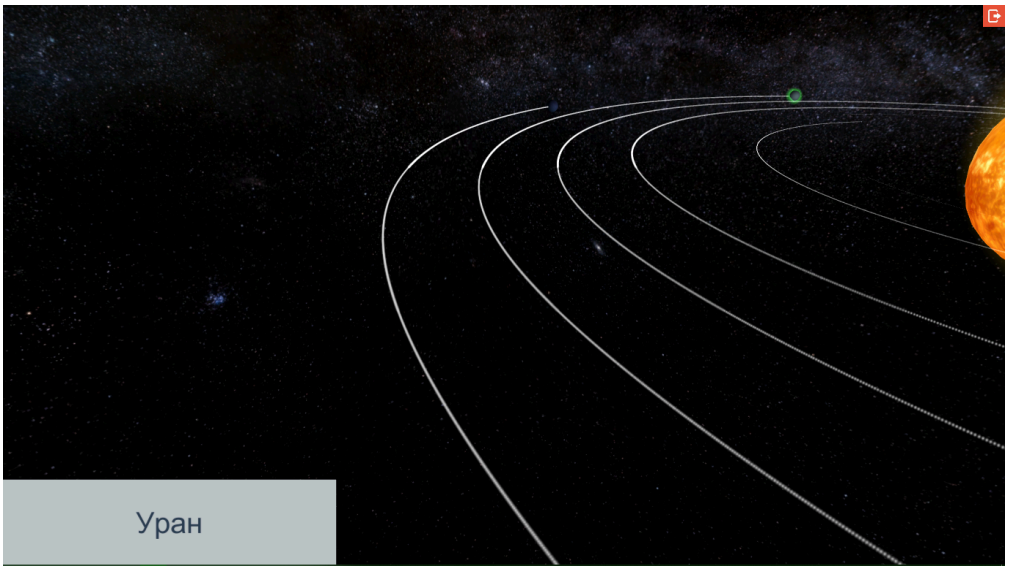

Рис. 3.11. Реакція програми при виборі космічного тіла

Після натискання процес емуляції завершиться і програма буде повністю вимкнена. Розміщення об’єктів не запам’ятовується, тому при повторному запуску програмна емуляція почнеться знову з моменту такого явища як «Парад планет». Для

повторного використання достатньо запустити виконуваний файл та пройти всі кроки експлуатації від початку.

Розроблений симулятор сонячної системи відзначається низкою важливих особливостей та переваг, які роблять його потужним і корисним інструментом для освіти, дослідження та розвитку:

1. Реалістична візуалізація: Симулятор використовує потужність графічного ігрового двигуна *Unity*3*D* для створення віртуального представлення сонячної системи. Велика увага приділяється деталізації та реалізму 3*D*-моделей планет, місяців та інших об'єктів. Користувач може переглядати ці об'єкти в різних масштабах та вивчати їх детальні характеристики.

2. Вивчення та аналіз руху об'єктів в реальному часі: Симулятор надає можливість спостерігати за рухом планет і інших об'єктів сонячної системи в реальному часі. Користувач може вивчати їхні орбіти, швидкості, траєкторії та інші параметри. Це дозволяє здійснювати освітні та наукові дослідження в сфері астрономії та астрофізики.

3. Інтерактивність: Програма надає інтерактивний інтерфейс, який дозволяє користувачам взаємодіяти з об'єктами сонячної системи. Ви можете вибирати планети для отримання інформації, обертати та переміщувати камеру, щоб отримати кращий огляд.

4. Можливість подальшого розширення: Симулятор розроблено з урахуванням можливості подальшого розширення та додавання нових функцій. Це означає, що програму можна покращувати та доповнювати з часом, враховуючи потреби користувачів та нові відкриття в науці.

Усі ці аспекти дозволяють використовувати цей симулятор для освіти, наукових досліджень та розвитку сучасних симуляцій сонячної системи, допомагаючи користувачам краще розуміти та досліджувати навколишній космічний простір.

3.3. Висновки до розділу

Під час розробки проєкту надзвичайно важливою була використана технологія ігрового двигуна *Unity* 2018, разом із відповідним фреймворком для мови програмування *C#* в програмному середовищі *Visual Studio* 2017. Даний фреймворк вже включав у себе два важливих класи: *GameObject* для реалізації графічних об'єктів на сцені та *MonoBehavior* для реалізації логіки програми.

Однак, для досягнення головної мети проєкту, якою була імітація реалістичного руху об'єктів в тривимірному просторі сонячної системи, довелося розробити додаткові класи та компоненти. Основні компоненти, створені під час розробки, включали в себе:

1. Класи для імітації руху об'єктів: Для кожного об'єкта сонячної системи (планети, сонця тощо) були розроблені окремі класи, що визначали їхні параметри та рух в тривимірному просторі. Ці класи включали в себе розрахунки шляху руху об'єкта відповідно до законів гравітації та дійсних параметрів планет.

2. Моделі планет та об'єктів: Для кожної планети була створена відповідна 3*D*-модель, яка реалістично відображала її розмір та форму. Ці моделі додавали візуальну реалістичність симуляції.

3. Система відображення орбіт: Для відображення орбіт планет була розроблена система, яка динамічно розраховувала траєкторії руху об'єктів та відображала їх на сцені. Це допомагало користувачам бачити траєкторії об'єктів та розуміти їхні рухи.

4. Інтерактивність та можливість взаємодії: В рамках проєкту був розроблений інтерактивний інтерфейс, що дозволяв користувачам активно взаємодіяти з об'єктами сонячної системи. Це включало в себе можливість взаємодії з планетами, отримання інформації про них, зміну параметрів симуляції тощо.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмний модуль в 3*D* симуляторі з елементами імітації руху об'єктів, який має важливе практичне та освітнє застосування. Цей модуль дозволить користувачам не лише візуалізувати об'єкти сонячної системи в тривимірному просторі, але й отримати глибоке розуміння їх руху, фізичних характеристик та взаємодії.

Застосування цього модуля може бути корисним у різних областях, включаючи освіту, науку, а також для популяризації астрономії серед широкого загалу. Освітні заклади можуть використовувати цей модуль для навчання студентів астрономії та фізики сонячної системи. Він дозволить студентам вивчати рух планет, їх орбіти, швидкості і обертання, що сприятиме глибшому розумінню фундаментальних астрофізичних процесів.

Поза освітою, цей модуль може бути використаний для проведення публічних астрономічних заходів, створення віртуальних астрономічних обсерваторій або ігрових додатків з астрономічною тематикою. Він може послужити інструментом для вивчення та наукових досліджень, пов'язаних з сонячною системою.

Основні переваги розробленого модуля включають:

– візуалізацію об'єктів сонячної системи в реальному масштабі та об'ємному вигляді;

– можливість налаштовувати параметри руху об'єктів, щоб досліджувати різні сценарії;

– інтерактивність, яка дозволяє користувачам взаємодіяти з об'єктами та спостерігати їх рух з різних точок огляду;

– можливість використовувати модуль в навчальних програмах та наукових дослідженнях.

Розроблений модуль є важливим кроком у розширенні можливостей візуалізації сонячної системи та сприятиме підвищенню інтересу до вивчення астрономії та природничих наук серед широкого загалу.

Розроблений симулятор сонячної системи відзначається кількома ключовими характеристиками, які варто врахувати при його оцінці та використанні:

1. Реалістична візуалізація: Однією з основних переваг цього симулятора є віртуальна реалістична візуалізація сонячної системи. Завдяки детально відтвореним 3*D* моделям планет, сонця та інших об'єктів, користувачі можуть бачити сонячну систему так, ніби вони спостерігають її в реальному світі.

2. Аналіз руху об'єктів в реальному часі: Симулятор надає можливість користувачам вивчати рух об'єктів сонячної системи в реальному часі. Це означає, що користувачі можуть спостерігати за тим, як планети обертаються навколо сонця, як вони рухаються вздовж своїх орбіт, і як вони взаємодіють одна з одною через гравітаційні сили.

3. Інтерактивність: Симулятор надає можливість взаємодії з планетами та іншими об'єктами сонячної системи. Користувачі можуть взаємодіяти з об'єктами, вибирати їх, дізнаватися більше інформації про них та впливати на їхні рухи.

4. Можливість розширення: Симулятор розроблено з урахуванням можливості подальшого розширення. Це означає, що в майбутньому можна додавати нові об'єкти, розширювати функціональність та вдосконалювати візуальну частину без необхідності переробки всього проєкту з нуля.

5. Педагогічний та дослідницький потенціал: Даний симулятор відкриває нові можливості для освіти та досліджень. Він може бути використаний в навчальних закладах для вивчення астрономії та фізики. Крім того, він може стати інструментом для дослідження різних аспектів руху об'єктів у сонячній системі та розвитку сучасних симуляторів космічних подій.

Усі ці характеристики роблять розроблений симулятор сонячної системи потужним інструментом для освіти, дослідження та віртуального вивчення космічних явищ.

Під час розробки проєкту надзвичайно важливою була використана технологія ігрового двигуна *Unity* 2018, разом із відповідним фреймворком для мови програмування *C#* в програмному середовищі *Visual Studio* 2017. Даний фреймворк

вже включав у себе два важливих класи: *GameObject* для реалізації графічних об'єктів на сцені та *MonoBehavior* для реалізації логіки програми.

Однак, для досягнення головної мети проєкту, якою була імітація реалістичного руху об'єктів в тривимірному просторі сонячної системи, довелося розробити додаткові класи та компоненти. Основні компоненти, створені під час розробки, включали в себе:

1. Класи для імітації руху об'єктів: Для кожного об'єкта сонячної системи (планети, сонця тощо) були розроблені окремі класи, що визначали їхні параметри та рух в тривимірному просторі. Ці класи включали в себе розрахунки шляху руху об'єкта відповідно до законів гравітації та дійсних параметрів планет.

2. Моделі планет та об'єктів: Для кожної планети була створена відповідна 3*D*-модель, яка реалістично відображала її розмір та форму. Ці моделі додавали візуальну реалістичність симуляції.

3. Система відображення орбіт: Для відображення орбіт планет була розроблена система, яка динамічно розраховувала траєкторії руху об'єктів та відображала їх на сцені. Це допомагало користувачам бачити траєкторії об'єктів та розуміти їхні рухи.

4. Інтерактивність та можливість взаємодії: В рамках проєкту був розроблений інтерактивний інтерфейс, що дозволяв користувачам активно взаємодіяти з об'єктами сонячної системи. Це включало в себе можливість взаємодії з планетами, отримання інформації про них, зміну параметрів симуляції тощо.

Загалом, використання ігрового двигуна *Unity* 2018 та створення додаткових класів та компонентів дало можливість досягти високого ступеня реалізму та імітації реального руху об'єктів сонячної системи в тривимірному просторі. Це стало основою для успішної реалізації цього проєкту та створення візуально захоплюючої симуляції сонячної системи.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

2. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – 39 с.

3. *Anderson, J., & McCormick, R.* (2005). *Ten pedagogic principles for e-learning. Insight – Thematic Dossiers – Ten Pedagogic Principles for E-learning. Retrieved from https://oeb.global/oeb-insights/wp-content/uploads/*2011/09/10-*Principles-for-Successful-E-learning.pdf.*

4. *Cui, X., Shi, H. A*-based Pathfinding in Modern Computer Games*/ *X. Cui, H. Shi* // *IJCSNS International Journal of Computer Science and Network Security-Vol.*11 *No.*1, *January* 2011.

5. *Il'kaev R.I., Seleznev V.E., Aleshin V.V., Klishin G.S. Numerical simulation of gas pipeline networks*: *theory, computational implementation, and industrial applications*. *Moscow*: *KomKniga,* 2005. 720 *p.*

6. *Lumsdaine, A. A., Glaser, R. Teaching Machines and Programmed Learning*: *A Source Book* / *Dept. of Audio-Visual Instruction, National Education Association,* 1961. – 724 *p.*

7. *Mika, M., Charla, C. Simple, Cheap Pathfinding* / *M. Mika, C. Charla* // *AI Game Programming Wisdom*-2002.

8. *O'Neill, J.* C. *Efficient Navigation Mesh Implementation* / *J. C. O'Neill* // *Journal of Game Development-Vol.* 1 *No.* 1, 2004.-71-90 *pp.*

9. *Operator Training Simulation Global Market Research Study. Market Analysis and Forecast through* 2017. *ARC Advisory Group*. 2012

10. *Ostapenko V.O.* 3*D visualization in learning systems* / *Artamonov Y.B., Ostapenko V.O.* // Матеріали *X*I*V* міжнар. наук.-техн. конф. "Авіа-2019" (23-24 квітня

2019). – К.: НАУ, 2019. – електронний збірник. Постійне посилання: *http*://*conference.nau.edu.ua*/*index.php*/*AVIA*/*AVIA*2019/*paper*/*view*/6230/4724.

11. *Yap, P. Grid-Based Path-Finding* / *P.Yap* // *AI* ‘02 *Proceedings of the* 15*th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*-2002. – 44-55 *pp*

12. *Zhang, G. Precise algorithm to generate random sequential addition of hard hyperspheres at saturation* / *G. Zhang, S. Torquato* // *Physical review, E* 88. 2013. *pp*.053312-1-9.

13. Аксенов М.В. Технологія розробки експертно-навчальних систем, спрямованих на навчання точним дисциплінам / Дисертація на здобуття наукового ступеня кандидата технічних наук. К.: КНУ, 2004.

14. Баляєва С.А., Углова А.Н. Проектування моделі навчання на основі системно-діяльнісного підходу // Системний аналіз у проєктуванні та управлінні: Тези *VII* Міжнародної науково-практичної конференції. К: Видавництво Знання. – 2003.

15. Бєлов В.В. Комп'ютерна реалізація рішення науково-технічних і освітніх завдань: навчальний посібник / В.В. Бєлов, І.В. Образцов, В.К. Іванов, Є.Н. Коноплев // Л: ЛДУ, 2015. 108 с.

16. Бєлов М.А. Принципи проєктування віртуальної комп'ютерної лабораторії на основі технології хмарних обчислень / М.А. Бєлов, О.Є. Антипов // Збірник праць міжнародної конференції "Сучасні проблеми і шляхи їх вирішення у науці, транспорті, виробництві та освіті – 2010". Одеса: УКРНІІМФ, 2010.

17. Вінокурова, С.Є. Модифікація методу навігаційного графа для пошуку шляху в тривимірному просторі // Програмні системи і обчислювальні методи. 2014. № 1. С. 109-124.

18. Гапанюк Ю.Є. Модель опису навчального середовища автоматизованого навчального курсу; *http*://*iu*5.*bmstu.com.ua*.

19. Горбаченко І.М. Методи моделювання процесу навчання та розробка інтерактивних навчальних курсів / Дисертація на здобуття наукового ступеня кандидата технічних наук. К.: КНУ, 2021.

20. Гриншкун А.В. Комп'ютерні ігри в навчанні школярів. // Вісник КНУ. Серія інформатика і інформатизація освіти. / К.: КНУ, – 2018, №4 (14). С. 46-47.

21. Дворецький С.І., Муромцев Ю.Л., Погонін В.А. Моделювання систем. – К.: Видавничий центр "Академія", 2009. – 320 с.

22. Дозорцев В.М. Комп'ютерні тренажери для навчання операторів технологічних процесів. К. Синергія. 2019. – 372 с.

23. Дозорцев В.М. та ін. Комп'ютерний тренінг операторів: нескінченна актуальність, нові можливості, людський фактор // Автоматизація в промисловості. 2015. № 7. С. 8-20.

24. Дозорцев В.М. Комп'ютерні тренажери для навчання операторів технологічних процесів. - М. : СИНТЕГ, 2009. - 372 с.

25. Дозорцев В.М. Світовий ринок комп'ютерних тренажерів для навчання операторів: тенденції, виклики, прогнози // Автоматизація в промисловості. 2016. № 2. – С. 47-50

26. Євгенев Г.Б. Інтеграція прикладних систем на основі баз знань // Програмні продукти і системи. Додаток до Міжнародного журналу "Проблеми теорії і практики управління". 2005. № 3.

27. Краснянський М.Н., Карпушкін С.В., Дедов Д.Л. Системний підхід до проєктування автоматизованої інформаційної системи навчання студентів і тренування операторів хіміко-технологічних систем // Вісник ЛГТУ. - 2019. - №4. - С. 926-935.

28. Нечипорук О.П., Мікульський В.В. Автоматизація тестування при візуалізації руху об'єктів в 3*d* симуляторах. – *System analysis and intelligent systems for management*: *proceedings of the XVII International Scientific and Practical Conference* (*Ankara, Turkey, May* 02-05, 2023). *Ankara,* 2023. *P*. 458-466.

29. Мікульський В.В. Формалізація даних про об'єкти в 3*d* симуляторах // Тези доповідей міжн. наук.-техн. конф. “Інтелектуальні технології лінгвістичного аналізу” (м. Київ, 24-25 жовтня 2023 р.) – К.: НАУ, 2023. – С. 67.

30. Мікульський В.В. Тестування зіткнень між об'єктами в 3*D* симуляторі // Тези доповідей наук.-практ. конф. “Сучасні тенденції розвитку системного програмування” (23-24 листопада 2023 р.). – К.: НАУ, 2023. – С. 43-44.

Лістинг окремих класів програмного коду

```
void Update(){
if ( Input.GetKey( KeyCode.W ) || Input.GetKey( KeyCode.UpArrow ) )
transform.position += transform.forward * moveSpeed * Time.deltaTime;
if ( Input.GetKey( KeyCode.S ) || Input.GetKey( KeyCode.DownArrow ) )
transform.position -= transform.forward * moveSpeed * Time.deltaTime;
if ( Input.GetKey( KeyCode.D ) || Input.GetKey( KeyCode.RightArrow ) )
transform.position += transform.right * moveSpeed * Time.deltaTime;
if ( Input.GetKey( KeyCode.A ) || Input.GetKey( KeyCode.LeftArrow ) )
transform.position -= transform.right * moveSpeed * Time.deltaTime;
Vector3 limit = transform.position;
limit.x = Mathf.Clamp( limit.x, -5000, 5000 );
limit.y = Mathf.Clamp( limit.y, -1000F, 1500F );
limit.z = Mathf.Clamp( limit.z, -5000, 5000 );
transform.position = limit;
rotX += Input.GetAxis( "Mouse X" ) * sensitivity;
rotY += Input.GetAxis( "Mouse Y" ) * sensitivity;
}
private void FixedUpdate(){
Vector3 target = center.transform.position;
Vector3 point = transform.position - target;
Vector3 newPoint = new Vector3();
newPoint.x = ( float ) ( ( point.x ) * Math.Cos( angle ) + ( point.z )
* Math.Sin( angle ) );
newPoint.y = transform.position.y;
newPoint.z = ( float )( ( point.z ) * Math.Cos( angle ) - ( point.x )
* Math.Sin( angle ) );
point = newPoint;
```

```
        newPoint.x = target.x + radius * point.x / point.magnitude;
        newPoint.y = transform.position.y;
        newPoint.z = target.z + radius * point.z / point.magnitude;
        transform.position = newPoint;
        }
        public class Rotation : MonoBehaviour {
        public float dXRot;
        public float dYRot;
        public float dZRot;
        public float speed;
        private void FixedUpdate(){
        transform.Rotate( new Vector3( dXRot, dYRot, dZRot ) * Time.fixedDeltaTime
* speed );
        }
        }
        void Start () {
        if ( autoRadius )
        radius = ( transform.position - center.transform.position ).magnitude;
        if (radius <= 0)
        Debug.Log( "radius is incorrect! ( " + radius + " <= 0 )" );
        If (angle != 0F)
        angle = ( float ) ( Time.fixedDeltaTime * angle * Math.PI/180.0);
        else
        angle = ( float ) Math.Acos( 1.0 - Time.fixedDeltaTime * speed / Math.Pow(
radius, 2 ) );
    }
```