Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB University Enterprises Cooperation In Game Industry In Ukraine

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРНЕТИКИ ТА СИСТЕМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ЛЯШЕНКО О.М.

ЕЛЕКТРОННИЙ НАВЧАЛЬНИЙ ПОСІБНИК

«Розробка комп'ютерних ігор за допомогою Unity 3D»

Для підготовки студентів
спеціальності 121 «Інженерія програмного забезпечення»

ХЕРСОН-2018

Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB University Enterprises Cooperation In Game Industry In Ukraine

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРНЕТИКИ ТА СИСТЕМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ЕЛЕКТРОННИЙ НАВЧАЛЬНИЙ ПОСІБНИК

«Розробка комп'ютерних ігор за допомогою Unity 3D»

Для підготовки студентів
спеціальності 121 «Інженерія програмного забезпечення»

GAMEHUB: «СПІВРОБІТНИЦТВО МІЖ УНІВЕРСИТЕТАМИ ТА ПІДПРИЄМСТВАМИ В СФЕРІ ІГРОВОЇ ІНДУСТРІЇ В УКРАЇНІ»

GAMEHUB: «UNIVERSITY-ENTERPRISES COOPERATION IN GAME INDUSTRY IN UKRAINE»
561728-EPP-1-2015-1- ES-EPPKA2-CBHE-JP

The handbook were performed with support of the Erasmus+ Programme of the European Union (561728-EPP-1-2015-1- ES-EPPKA2-CBHE-JP). The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

ХЕРСОН-2018

Co-funded by the Erasmus+ Programme of the European Union

GAMEHUB University Enterprises Cooperation In Game Industry In Ukraine

УДК 004.4
Р 65

Розробка комп'ютерних ігор за допомогою Unity 3D: електронний навчальний посібник для підготовки студентів спеціальності 121 «Інженерія програмного забезпечення» / Укладач: О.М. Ляшенко. – Херсон: видавництво ФОП Вишемирський В.С., 2018. – 220 с.

ISBN 978-617-7573-90-5 (електронне видання)

Укладачі:
Ляшенко О.М., к.т.н., доцент кафедри Програмних засобів і технологій.

Рецензенти:

Шарко О.В., д.т.н, професор кафедри транспортних технологій Херсонської державної морської академії.

Львов М.С., д.ф-м.н, професор, завідувач кафедри інформатики, програмної інженерії та економічної кібернетики Херсонського державного університету

Розглянуто на засіданні кафедри Програмних засобів і технологій,
Протокол №11 від 23 травня 2017 року.

Розглянуто на засіданні Вченої ради Херсонського національного технічного університету,
Протокол №9 від 2 червня 2017 року.

УДК 004.4

ISBN 978-617-7573-90-5

ЗАГАЛЬНИЙ ВСТУП

Електронний навчальний посібник складений відповідно до програми навчальної дисципліни «Технології розробки крос-платформних комп'ютерних ігор», яка входить до дисциплін підготовки студентів спеціальності 121 «Інженерія програмного забезпечення за спеціалізацією «Розробка комп'ютерних ігор», яка впроваджена в рамках виконання міжнародного проекту Еразмус+ 561728-EPP-1-2015-1- ES-EPPKA2-CBHE-JP «GameHub: Співробітництво між університетами та підприємствами в сфері ігрової індустрії в Україні».

Основною метою навчального посібника є узагальнення й систематизація теоретичних надбань в галузі розробки крос-платформних комп'ютерних ігор та створення ігрового контенту за допомогою системи Unity 3D.

Навчальний посібник містить довідник дисципліни, лекційний матеріал, методичні вказівки до виконання кожної теми, питання для самоконтролю, задачі для самостійного розв'язку, критерії оцінювання практичних завдань та список рекомендованої літератури.

Перший розділ навчального посібника містить довідник дисципліни.

Другий розділ навчального посібника має теоретичну спрямованість і передбачає ознайомлення читачів з сучасними технологіями розробки крос-платформних комп'ютерних ігор, а саме: технологією анімації та створення ігрового контенту, технологією фізичного моделювання об'єктів, технологією людино-комп'ютерної взаємодії, ігровими мобільними технологіями, технологією розробки ігрових додатків за допомогою сучасних крос-платформних мов програмування, які використовуються на рівнях компіляції та виконання.

Змістовність, логічність і доступність викладу теоретичного матеріалу робить можливим його використання студентами, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» при підготовці до практичних занять, написанні курсових робіт, а також кваліфікаційних робіт рівнів «бакалавр» і «магістр».

Третій розділ навчального посібника має практичну спрямованість і передбачає набуття вмінь:

– використання сучасного інструментарію для роботи з графікою при проектуванні ігрового 3D-контенту за допомогою Unity 3D;

– використання відповідних компонентів Unity 3D для роботи з фізикою 3D- об'єктів;

– використання можливостей інтегрованого середовища розробки MonoDevelop, яке поєднує в собі функції текстового редактора з додатковими можливостями для налагодження і виконання завдань з управління ігровими проектами;

– використання відповідних компонентів Unity 3D для анімації об’єктів;

– використання відповідних аудіо-компонентів при проектуванні ігрових додатків;

– використання відповідних інструментів Unity 3D для роботи з системою подій Unity 3D;

– використання відповідних інструментів Unity 3D для створення крос-платформних комп’ютерних ігор (Windows, Android).

У третьому розділі розглядаються також види практичних завдань, для перевірки набутих знань, умінь та навичок.

У кінці кожного розділу навчального модуля наводяться контрольні питання та задачі для самостійного вивчення матеріалу.

Навчальний посібник забезпечує реальну можливість формування у студентів абстрактного мислення, розвитку здібностей до генерування нових ідей (креативність), самостійного прийняття обґрунтованих рішень щодо створення і супроводження крос-платформного програмного забезпечення.

Використаний під час підготовки навчального посібника методологічний прийом, заснований на проблемно-орієнтованому навчанні, дає змогу формувати креативне мислення та когнітивні здібності у студентів через вирішення проблемно-орієнтованих задач в області розробки комп’ютерних ігор.

Використання елементів проблемно-орієнтованого навчання дозволяє ефективно розподілити навчальний матеріал на певні порції, що відповідають окремим елементам засвоєння, та за рахунок наявності в кожній такій порції проблемного завдання, яке студент має виконати, забезпечує індивідуалізацію навчання з належним зворотним зв’язком та самоконтролем у виконанні кожного завдання.

Такий методологічний прийом, безумовно, буде сприяти підвищенню не лише ефективності підготовки фахівців з розробки крос-платформного програмного забезпечення, а й їх мотивацію щодо отримання нових теоретичних та практичних знань.

ЗМІСТ
ЗАГАЛЬНИЙ ВСТУП
ДОВІДНИК НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

ЛЕКЦІЇ ТА МЕТОДИКА ЇХ ПРОВЕДЕННЯ

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОНИХ РОБІТ

Вступ

Предметом навчальної дисципліни є сучасні методи та технології розробки крос-платформних комп'ютерних ігор, технології крос-платформного програмування, технології людино-комп'ютерної взаємодії, технології віртуальної та доповненої реальності.

Мета дисципліни

Метою викладання навчальної дисципліни «Технології розробки крос-платформних комп'ютерних ігор» є формування у студентів цілісної системи знань щодо технологій створення комп'ютерних ігрових додатків із застосуванням сучасних крос-платформних мов програмування.

Очікувані результати

Основними завданнями вивчення дисципліни «Технології розробки крос-платформних комп'ютерних ігор» є набуття знань про сучасні методи та технології розробки крос-платформних комп'ютерних ігор, а саме: технології анімації та створення ігрового контенту, технології фізичного моделювання об'єктів, технології людино-комп'ютерної взаємодії, графічні технології комп'ютерних ігор і віртуальної реальності, ігрові мобільні технології, сучасні крос-платформні мови програмування, які використовуються на рівнях компіляції, виконання та крос-платформні інтерпретатори, а також формування у студентів професійних умінь і навичок щодо розробки крос-платформних комп'ютерних ігор та створення ігрового контенту за допомогою Unity 3D.

Представлений матеріал зорієнтовано на студентів вищих навчальних закладів IT – спеціальностей, які вивчають такі мови програмування як Java, C# та JavaScript.

Зміст навчальної дисципліни передбачає отримання студентами системи знань та формування системи професійних умінь і навичок щодо розробки комп'ютерних ігор та створення ігрового контенту за допомогою Unity 3D.

Першу половину навчальної дисципліни присвячено знайомству з Unity 3D, а саме базовою системою компонентів, базовими 3D-моделями і текстурами.

Другу половину навчальної дисципліни присвячено розробці власних 3D- моделей та їх розгортанню на платформах Windows та Android.

Наприкінці вивчення навчальної дисципліни студенти матимуть змогу створювати власні тривимірні моделі, додавати інтерактивні пристрої та елементи до ігрового додатку, звукове супроводження та розгортати ігрові додатки на сучасних платформах.

1 Опис навчальної дисципліни

Галузь знань: 12 «Інформаційні технології».
Спеціальність: 121 «Інженерія програмного забезпечення»
Рівень підготовки: перший (бакалаврський) рівень вищої освіти
Назва дисципліни: «Технології розробки крос-платформних комп'ютерних ігор»
Семестри: 5,6
Кількість кредитних одиниць: дисципліна - 7,0
Орієнтовна кількість часів: дисципліна - 210
Викладач: к.т.н., доцент Ляшенко О.М.

2 Перелік компетентностей та результати навчання*

Загальні (універсальні) компетентності

ЗК-1 Здатність до абстрактного мислення, аналізу та синтезу.
ЗК-2 Здатність застосовувати знання в практичних ситуаціях.
ЗК-3 Здатність вчитися й оволодівати сучасними знаннями, здійснювати пошук, оброблення й аналіз інформації з різних джерел.
ЗК-4 Здатність генерувати нові ідеї (креативність), працювати в команді, бути критичним і самокритичним, розробляти проекти, приймати обґрунтовані рішення.
ЗК-5 Здатність оцінювати та забезпечувати якість виконуваних робіт.
ЗК-6 Здатність застосовувати математичний апарат, а також теоретичні, методичні й алгоритмічні основи інформаційних технологій під час вирішення прикладних і наукових завдань в області інформаційних систем і технологій.

Спеціальні (фахові) компетентності

ФК-1 Здатність опановувати сучасні технології математичного моделювання об'єктів, процесів і явищ, розробляти обчислювальні моделі та алгоритми чисельного розв'язання задач математичного моделювання з урахуванням похибок наближеного чисельного розв'язання професійних задач; здійснювати формалізований опис задач дослідження операцій в організаційно-технічних і соціально-економічних системах різного призначення, визначати їх оптимальні рішення, будувати моделі оптимального вибору управління з

* Освітньо-професійна програма першого (бакалаврського) рівня вищої освіти. Спеціальність 121 «Інженерія програмного забезпечення», спеціалізація «Програмна інженерія».

урахуванням змін параметрів економічної ситуації, оптимізувати процеси управління в системах різного призначення та рівня ієрархії.

ФК-2 Здатність реалізовувати багаторівневі обчислювальні моделі на основі архітектури клієнт-сервер (включаючи сховища, бази та банки даних і знань) для забезпечення обчислювальних потреб багатьох користувачів.

ФК-3 Здатність формулювати та забезпечувати вимоги щодо якості програмного забезпечення у відповідності з вимогами, технічним завданням та стандартами.

ФК-4 Здатність здійснювати аналіз і функціональне моделювання процесів, побудову та застосування функціональних моделей систем; застосування методів та інструментальних засобів для управління процесами життєвого циклу систем відповідно до вимог замовника.

ФК-5 Здатність опановувати та комплексно застосовувати базові загальні знання в області програмування (у тому числі, структурного, функціонального, логічного, об'єктно-орієнтованого, паралельного) та візуального проектування системного та прикладного програмного забезпечення; володіти алгоритмічним мисленням; проектувати та розробляти програмне забезпечення на основі інтеграції провідних сучасних технологій (із застосуванням відповідних моделей, методів та алгоритмів обчислень, структур даних); застосовувати об'єктно-орієнтований підхід під час проектування складних програмних систем методами програмної інженерії для реалізації програмного забезпечення з урахуванням вимог до його якості, надійності, виробничих характеристик.

ФК-6 Здатність опановувати та комплексно застосовувати базові знання в області принципів, методів і алгоритмів комп'ютерної графіки під час розробки графічних інтерфейсів взаємодії людини з комп'ютером.

ФК-7 Здатність здійснювати процес інтеграції системи, застосовувати стандарти і процедури управління змінами для підтримки цілісності загальної функціональності і надійності програмного забезпечення.

Програмні результати навчання

ПРН-1 Здатність аналізувати проблеми щодо створення програмного забезпечення.

ПРН-2 Здатність, аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки.

ПРН-3 Здатність використовувати знання щодо методів та засобів збору, формулювання та аналізу вимог до програмного забезпечення.

ПРН-4 Здатність застосовувати знання ефективних підходів щодо проектування програмного забезпечення.

ПРН-5 Здатність розуміти основні процеси, фази та ітерації життєвого циклу програмного забезпечення.

ПРН-6 Здатність розуміти і застосовувати знання сучасних підходів щодо оцінки та забезпечення якості програмного забезпечення.

ПРН-7 Здатність розуміти і застосовувати знання щодо відповідних математичних понять, методів доменного, системного і об'єктно-орієнтованого аналізів та математичного моделювання для розробки програмного забезпечення.

ПРН-8 Здатність застосовувати на практиці фундаментальні концепції і основні принципи функціонування мовних, інструментальних і обчислювальних засобів інженерії програмного забезпечення.

ПРН-9 Здатність застосовувати знання методів компонентної розробки програмного забезпечення, виділяючи інтерфейси і реалізації та взаємодію між модулями, підсистемами і компонентами.

Перелік компетентностей та результати навчання
Навчальна дисципліна «Технології розробки крос-платформних програмних додатків за допомогою Unity 3D»

Загальні (універсальні) компетентності	*ЗК-4, ЗК-5, ЗК-6*
Спеціальні (фахові) компетентності	*ФК-1, ФК-2, ФК-3, ФК-5, ФК-6*
Програмні результати навчання	*ПРН-4, ПРН-7, ПРН-8, ПРН-10*

3 Міждисциплінарні зв'язки

Для засвоєння матеріалу використовується такий перелік дисциплін, що забезпечують засвоєння навчального матеріалу дисципліни «Технології розробки крос-платформних комп'ютерних ігор»:

- Вища математика
- Основи програмування
- Основи програмної інженерії
- Алгоритми та структури даних
- Об'єктно-орієнтоване програмування
- Бази даних
- Комп'ютерна графіка
- Розробка мобільних додатків на платформі Android

4 Мета та передбачувані результати вивчення навчальної дисципліни

4.1 Мета навчальної дисципліни

Метою навчальної дисципліни є формування у студентів професійних умінь і навичок щодо розробки комп'ютерних ігор та створення ігрового контенту за допомогою Unity 3D. Студенти повинні ознайомитись з системою компонентів Unity 3D та методами розробки 3D - моделей та вміти використовувати їх при створенні комп'ютерних ігор незалежно від платформи.

4.2 Результати навчання

Знання та їх використання

У разі успішного оволодіння матеріалами навчальної дисципліни студент буде вміти використовувати сучасні методи та технології розробки комп'ютерних ігор та ігрового 3D-контенту за допомогою Unity 3D, а саме: методи роботи з графікою (освітлення, камери, матеріали, текстури, ландшафти, рендеринг); методи роботи з компонентами, які використовуються з 3D-фізикою; методи роботи в інтегрованому середовищі розробки MonoDevelop, яке поєднує в собі функції текстового редактора з додатковими можливостями для налагодження і виконання завдань з управління ігровими проектами; методи роботи з аудіо-компонентами (імпорт і налаштування звуку); методи анімації об'єктів (анімація спрайтами, анімація, заснована на фізиці, анімація за допомогою системи Mecanim); методи роботи з компонентами, які використовуються для створення крос-платформних комп'ютерних 3D-ігор (Windows, Android).

Дослідницькі навички

У разі успішного вивчення навчальної дисципліни студент буде вміти комплексно застосовувати технології та методи розробки крос-платформного програмного забезпечення при проектуванні комп'ютерних ігор; вміти комплексно застосовувати технології та методи геометричного моделювання об'єктів, які засновано на математичних методах аналітичної геометрії при розробці комп'ютерних ігор та ігрового контенту; вміти застосовувати сучасні крос-платформні мови програмування (C#, Java, JavaScript) при розробці ігрових додатків; вміти застосовувати методи анімації об'єктів при розробці ігрового контенту; вміти здійснювати апробацію отриманих результатів, приймаючи участь у відповідних формах організації наукових заходів (семінарах, конференціях, конгресах, симпозіумах тощо); вміти

впроваджувати отримані результати у практичну діяльність підприємств та організацій.

Спеціальні вміння

У разі успішного вивчення навчальної дисципліни студент буде вміти:

– використовувати відповідні інструменти для роботи з графікою при проектуванні ігрового 3D-контенту за допомогою Unity 3D;

– використовувати відповідні компоненти Unity 3D для роботи з фізикою 3D- об'єктів;

– використовувати можливості інтегрованого середовища розробки MonoDevelop, яке поєднує в собі функції текстового редактора з додатковими можливостями для налагодження і виконання завдань з управління ігровими проектами;

– використовувати відповідні компоненти Unity 3D для анімації об'єктів;

– використовувати відповідні аудіо-компоненти при проектуванні ігрових додатків;

– використовувати відповідні інструменти Unity 3D для роботи з системою подій Unity 3D;

– використовувати відповідні інструменти Unity 3D для створення крос-платформних комп'ютерних ігор (Windows, Android).

Соціальні вміння

У разі успішного вивчення навчальної дисципліни студент буде вміти працювати в складі професійної проектної команди, що виконує дослідження та розробку в сфері крос-платформного програмного забезпечення, усвідомлюючи призначені персональні обов'язки, заплановані витрати та конкретну відповідальність за результати роботи.

Особисті якості

У разі успішного вивчення навчальної дисципліни студент буде вміти:

– обробляти та систематизувати професійні знання щодо створення і супроводження крос-платформного програмного забезпечення;

– розробляти, реалізовувати і координувати процеси, фази та ітерації життєвого циклу крос-платформних програмних додатків;

– обґрунтовано обирати та освоювати інструментарій з розробки та супроводження крос-платформного програмного забезпечення;

– аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки.

5 Календарний план семестру і структура навчальної дисципліни

5.1 Структура навчальної дисципліни

Номер	Змістовний модуль	Тиждень вивчання
1	Технології розробки крос-платформного програмного забезпечення. Особливості розробки крос-платформних комп'ютерних ігор.	1-4
2	Крос-платформні мови програмування (C#, Java, JavaScript).	5-9
3	Розробка комп'ютерних ігор за допомогою Unity 3D.	10-16

5.2 Інформаційне наповнення змістових модулів навчальної дисципліни

Номер тижня	Зміст
1	Базові концепції крос-платформного програмування. Типи крос-платформності. Крос-патформність на рівні апаратної платформи. Крос-патформність на рівні ОС.
2	Крос-платформні середовища виконання. Крос-платформне програмне забезпечення. Мобільність програмного забезпечення.
3	Особливості розробки крос-платформних комп'ютерних ігор.
4	Об'єктно-орієнтована концепція крос-платформного програмування. Успадкування. Інкапсуляція. Поліморфізм. Платформа .NET.
5	Крос-платформні мови програмування.
6	Особливості розробки крос-платформних комп'ютерних ігор мовою C#.
7	Особливості розробки крос-платформних комп'ютерних ігор мовою Java.
8	Особливості розробки крос-платформних комп'ютерних ігор мовою JavaScript.
9	Огляд сучасних інструментальних засобів розробки ігрових додатків.
10	Основи роботи в Unity 3D. Інтерфейс програми: головне меню, огляд проекту, ієрархія, сцена, ігровий вид, інспектор. Робота з графікою (освітлення, камери, матеріали, текстури, ландшафти, рендеринг).
11	Робота з компонентами Unity 3D для роботи з фізикою 3D-об'єктів (Rigidbody, Colliders, Joints, Character Controller).

12	Розробка та використання скриптів в Unity 3D. Створення і знищення ігрових об'єктів (GameObjects). Управління ігровими об'єктами за допомогою відповідних компонентів. Робота в інтегрованому середовищі розробки MonoDevelop. Система подій EventSystem.
13	Робота з аудіо-компонентами Unity 3D. Імпорт і налаштування звуку.
14	Анімація об'єктів в Unity 3D. Анімація спрайтами (окремі спрайти, атлас спрайтів). Анімація, заснована на фізиці (використання фізичної системи Unity). Анімація за допомогою системи Mecanim.
15	Робота з інструментами Unity 3D, які використовуються для створення крос-платформних комп'ютерних 3D-ігор (Windows, Android).
16	Контрольна робота Презентація індивідуального завдання

6 Форми навчання

Навчальний процес здійснюється у таких формах: навчальні аудиторні заняття (лекції, лабораторні, консультації), виконання індивідуальних завдань, самостійна робота, практична підготовка, контрольні заходи.

Аудиторна робота включає 24 лекції та 30 лабораторних робіт.

Лекція - основна форма проведення навчальних занять, яка призначена для засвоєння теоретичного матеріалу.

Лабораторні заняття - форма навчальних занять, на яких студенти поглиблюють теоретичні знання з навчальної дисципліни та набувають практичних навичок роботи з Unity 3D за допомогою спеціалізованого ігрового обладнання.

Консультації - це навчальні заняття, на яких студент отримує відповіді викладача на конкретні запитання або пояснення певних теоретичних положень чи практичних аспектів роботи з Unity 3D.

Індивідуальні завдання передбачають проектування та розробку власного ігрового додатку та ігрового 3D-контенту в Unity 3D. Тема індивідуального завдання вибирається студентом самостійно. Індивідуальні завдання виконуються студентом самостійно з консультацією викладача.

Самостійна робота здійснюється у вільний від аудиторних навчальних занять час. Вона спрямована на оволодіння студентом навчальним матеріалом, практичними навичками, передбачає отримання нових знань та самостійне вирішення завдань.

7 Порядок проведення атестації

У навчальній дисципліні передбачено виконання лабораторних робіт, контрольної роботи та індивідуального завдання. Підсумкова рейтингова оцінка включає оцінку за виконання і захист лабораторних робіт, контрольної роботи та індивідуального завдання.

Курс оцінюється за 100 бальною шкалою. Максимальна сумарна оцінка за виконання лабораторних робіт в межах навчальної дисципліни складає 6 балів. У кожній роботі 30% нараховується за виконання практичної частини і її відповідність поставленій задачі і 70% – за захист роботи (аргументоване пояснення ходу роботи і відповіді на теоретичні питання).

Контрольна робота оцінюється у 35 балів. Робота проводиться у форматі тесту, який містить 25 теоретичних питань, які оцінюються по 1 балу кожне та 5 практичних питань, які оцінюються по 2 бали кожне (максимально 35 балів за виконання контрольної роботи).

Виконання та презентація фінального індивідуального завдання оцінюється у 29 балів.

Графік проведення поточного оцінювання

Номер тижня	Оцінювання
2	Оцінка виконання лабораторної роботи 1
4	Оцінка виконання лабораторної роботи 2
6	Оцінка виконання лабораторної роботи 3
8	Оцінка виконання лабораторної роботи 4
10	Оцінка виконання лабораторної роботи 5
12	Оцінка виконання лабораторної роботи 6
16	Оцінка виконання контрольної роботи Оцінка виконання індивідуального завдання

Подання звіту щодо виконання лабораторних робіт

Лабораторні роботи виконуються протягом аудиторного заняття. У кінці кожного заняття студент зобов'язаний продемонструвати виконане завдання.

У межах двох тижнів студент зобов'язаний самостійно оформити звіт до попередньої лабораторної роботи, де повинні бути вказані основні кроки, що були виконані у ході даної роботи.

Протягом аудиторного заняття, відведеного для виконання наступної лабораторної роботи студент має захистити попередню роботу, відповівши на

питання щодо теоретичного матеріалу та надавши пояснення про хід і результати лабораторної роботи.

Оцінювання лабораторних робіт:

− за кожну лабораторну роботу студент отримує 6 балів (максимальна сумарна оцінка за усі лабораторні роботи в межах навчальної дисципліни складає 36 балів);

− якщо робота виконана не самостійно, то знімається 50% від максимальної кількості балів;

− якщо в програмі не витримано основні правила створення програмних продуктів (наприклад, модульність, дружній інтерфейс, наявність коментарів) знімається 5%.

За кожен тиждень запізнення захисту лабораторної роботи (комп'ютерного практикуму) нараховується штрафні – 0,2 бали (максимально 5 днів).

Методи оцінки змістовних модулів навчальної дисципліни

Кількість балів в загальній оцінці змістовних модулів відповідає наступному:

Виконання лабораторної роботи 1	максимально 6 балів.
Виконання лабораторної роботи 2	максимально 6 балів.
Виконання лабораторної роботи 3	максимально 6 балів.
Виконання лабораторної роботи 4	максимально 6 балів.
Виконання лабораторної роботи 5	максимально 6 балів.
Виконання лабораторної роботи 6	максимально 6 балів.
Виконання контрольної роботи	максимально 35 балів.
Виконання індивідуального завдання	максимально 29 балів.

Усі набрані бали підсумовуються (максимально 100 балів), штрафні бали за запізнення в представленні звіту з лабораторної роботи віднімаються.

Метод оцінки дисципліни в цілому

Оцінки студентів за результатами вивчення змістовних модулів 1 – 3 підсумовуються. Таким чином розраховується сумарна оцінка студента в балах за дисципліною.

Сумарна оцінка в балах переводиться за нижченаведеною шкалою оцінювання в національну та ECTS – оцінку.

Таблиця 1.1 - Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	**Оцінка ECTS**	**Оцінка за національною шкалою**	
		для екзамену, курсового проекту (роботи), практики	**для заліку**
90 – 100	**A**	відмінно	зараховано
82-89	**B**	добре	
74-81	**C**		
64-73	**D**	задовільно	
60-63	**E**		
35-59	**FX**	незадовільно з можливістю повторного складання	не зараховано з можливістю повторного складання
0-34	**F**	незадовільно з обов'язковим повторним вивченням дисципліни	не зараховано з обов'язковим повторним вивченням дисципліни

8 Зворотній зв'язок

Про результати захисту лабораторних робіт студенти дізнаються під час заняття.

Про результати контрольної роботи студенти дізнаються протягом одного тижня.

Консультації для студентів проводяться викладачем впродовж семестру два рази на тиждень.

Інформація щодо оцінки за дисципліною в цілому надається студентам на 16 тижні навчання.

Контактні дані для on-line допомоги та консультування:

Викладачі: к.т.н., доцент Ляшенко О.М., e-mail: olenakntu@gmail.com

9 Викладацький склад та допоміжні джерела

Обов'язки викладачів

Основні обов'язки викладачів навчальної дисципліни полягають у проведенні лекцій і лабораторних занять згідно навчальної програми та проведенні контролю якості отриманих знань, умінь і навичок.

Обов'язки координатора дисципліни

Головні обов'язки координаторів навчальної дисципліни полягають у розробці та внесенні змін до змістовних модулів у відповідності з поточними потребами, навчальними планами тощо; координації і управлінні професорсько-викладацьким складом; координації проведення заліків.

Обов'язки допоміжного персоналу

Допоміжний персонал здійснює підготовку комп'ютерної техніки та спеціалізованого ігрового обладнання до виконання лабораторних робіт студентами та надає технічну підтримку студентам під час виконання лабораторних робіт.

10 Навчальна програма і матеріали

10.1 Тема 1: Основи роботи в Unity 3D.

Анотація

Лекція знайомить з основами роботи в Unity 3D - системою розробки крос-платформних 2D- і 3D-ігор та інтерактивного контенту.

Мета лекції

Ознайомити студентів з перевагами та недоліками розробки ігрових додатків та ігрового контенту за допомогою Unity 3D; з новим функціоналом та відповідним інструментарієм системи Unity 3D, який допоможе розробникам успішно створювати анімаційні та ігрові сцени; сферами застосування комп'ютерних ігор, розроблених за допомогою Unity 3D.

Очікувані результати

Студент оволодіє поняттями: комп'ютерна крос-платформна 2D- і 3D-гра, інтерактивний контент, ігрова сцена. Буде знати основні переваги та недоліки розробки ігрових додатків та ігрового контенту за допомогою Unity 3D, функціонал та відповідний інструментарій системи Unity 3D для створення анімаційних та ігрових сцен, сфери застосування комп'ютерних ігор, розроблених за допомогою Unity 3D.

Контрольні запитання

- Дайте визначення поняттям «комп'ютерна крос-платформна 3D-гра», «інтерактивний контент», «ігрова сцена».
- Назвіть основні переваги розробки ігрових додатків та ігрового контенту за допомогою Unity 3D.
- Який інструментарій Unity 3D використовується для створення анімаційних та ігрових сцен?
- У чому полягають відмінності крос-платформної розробки комп'ютерних ігор?
- Назвіть сфери застосування комп'ютерних ігор, розроблених за допомогою Unity 3D.

10.2 Лабораторна робота № 1. Режими роботи та інтерфейс Unity 3D. Робота з графікою.

Анотація

Лабораторна робота орієнтована на ознайомлення студентів з інтерфейсом системи Unity 3D (головне меню, огляд проекту, ієрархія, сцена, ігровий вид, інспектор) та методами роботи з графікою (освітлення, камери, матеріали, текстури, ландшафти, рендеринг).

Мета лабораторної роботи

Освоїти основні прийоми роботи з 2D- і 3D-режимами системи Unity 3D, інтерфейсом системи та методами роботи з графікою. Вивчити можливості роботи з освітленням в Unity 3D (Directional Light, Point Light, Spot lights, Area Light). Вивчити можливості роботи з камерою в Unity 3D (Perspective and orthographic cameras). Освоїти основні прийоми роботи з матеріалами та текстурами в Unity 3D. Вивчити можливості створення та редагування ландшафтів (Terrain). Освоїти основні прийоми рендерингу в Unity 3D.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде знати та вміти застосовувати основні прийоми роботи з 2D- і 3D-режимами системи Unity 3D, вміти застосовувати основні графічні компоненти системи для роботи з освітленням та камерою, застосовувати основні прийоми роботи з матеріалами та текстурами при розробці ігрових сцен, створювати та редагувати ландшафти, вміти застосовувати прийоми рендерингу при розробці ігрових додатків.

Контрольні запитання

- Назвіть базові графічні компоненти системи Unity 3D для роботи з освітленням та камерою.
- Перерахуйте основні елементи інтерфейсу системи Unity 3D.
- Назвіть базові компоненти системи Unity 3D для створення та редагування ландшафтів.
- Дайте визначення понятю «рендеринг».

10.3 Тема 2. Фізика 3D - об'єктів.

Анотація

Лекція знайомить з основними компонентами, які використовуються для роботи з фізикою 3D - об'єктів.

Мета лекції

Ознайомити студентів з основними компонентами Unity 3D для роботи з фізикою 3D - об'єктів. Ознайомити студентів з властивостями компонентів Rigidbodies, Colliders, Joints, Character Controllers та можливостями, які вони надають розробникам ігрового контенту.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде знати та вміти застосовувати основні компоненти та їх властивості для роботи з фізикою 3D - об'єктів при розробці ігрових додатків.

Контрольні запитання

- Назвіть базові компоненти системи Unity 3D для роботи з фізикою 3D - об'єктів.
- Назвіть основні властивості компоненту Rigidbodies.
- Назвіть основні властивості компоненту Colliders.
- Для чого в системи Unity 3D використовуються Joint – компоненти?
- Назвіть опції Joint - компонента, які можна включити для різних ефектів.
- Для чого в системи Unity 3D використовуються контролери персонажа (Character Controllers)?
- Назвіть основні властивості компоненту Character Controllers.
- Які можливості надають розробникам ігрового контенту компоненти Box Collider, Capsule Collider, Character Joints, Configurable Joint, Fixed Joints, Sphere Collider, Spring Joint?

10.4 Лабораторна робота № 2. Компоненти Unity 3D для роботи з фізикою 3D- об'єктів.

Анотація

Лабораторна робота орієнтована на оволодіння студентами навичок роботи з основними компоненти, які використовуються для роботи з фізикою 3D - об'єктів.

Мета лабораторної роботи

Створення 3D - об'єктів та використання компонентів Rigidbodies, Colliders, Joints, Character Controllers для роботи з фізикою 3D - об'єктів.

Робота з властивістю «Is Kinematic» компонента Rigidbody, яка надає можливість виключення об'єкта з-під контролю фізичного движка, і дозволяє переміщати його кінематично за допомогою відповідного скрипта.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде вміти використовувати компоненти Rigidbodies, Colliders, Joints, Character Controllers та їх властивості для роботи з фізикою 3D - об'єктів, використовувати властивість «Is Kinematic» компонента Rigidbody для виключення об'єкта з-під контролю фізичного движка, і кінематичного переміщення об'єкта за допомогою відповідного скрипта.

Контрольні запитання

- Назвіть етапи створення 3D - об'єктів в системі Unity 3D.
- Для чого у компонента Rigidbody використовується властивість «Is Kinematic»?
- Назвіть тип нефізичного руху, який здійснюється за допомогою коду.
- Який компонент дозволяє здійснювати обертання 3D - об'єкта навколо заданої точки та осі?

10.5 Тема 3: Розробка та використання скриптів в Unity 3D. Середовище розробки MonoDevelop. Система подій (EventSystem).

Анотація

Лекція знайомить з технологією розробки скриптів в системі Unity 3D, методами створення і знищення ігрових об'єктів, управління ігровими об'єктами за допомогою відповідних компонентів, методами роботи в інтегрованому середовищі розробки MonoDevelop, яке поєднує в собі функції текстового редактора з додатковими можливостями для налагодження і виконання завдань з управління ігровими проектами, системою подій «EventSystem» Unity 3D.

Мета лекції

Ознайомити студентів з методами створення, управління і знищення ігрових об'єктів, методами роботи в інтегрованому середовищі розробки MonoDevelop, способами відправки подій до об'єктів в ігровому додатку – системою подій «EventSystem».

Очікувані результати

Сформувати у студентів знання щодо основних методів створення і знищення ігрових об'єктів в системі Unity 3D, основних компонентів, які використовуються для управління ігровими об'єктами, основними принципами роботи в інтегрованому середовищі розробки MonoDevelop, основних методів роботи з системою подій «EventSystem».

Контрольні запитання

- Які мови програмування підтримує система Unity 3D?
- Яким чином здійснюється управління ігровими об'єктами (GameObjects) в системі Unity 3D?
- Назвіть переваги використання інтегрованого середовища розробки MonoDevelop для налагодження і виконання завдань з управління ігровими проектами.
- Назвіть основні методи роботи з системою подій «EventSystem» в Unity 3D.

10.6 Лабораторна робота №3.
Робота в інтегрованому середовищі розробки MonoDevelop.

Анотація

Лабораторна робота орієнтована на оволодіння студентами навичок роботи в інтегрованому середовищі розробки MonoDevelop.

Мета лабораторної роботи

Знайомство з інтегрованим середовищем розробки MonoDevelop, налаштування MonoDevelop, редагування вихідного коду в MonoDevelop.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде вміти створювати, редагувати скриптові файли з використанням середовища розробки MonoDevelop.

Контрольні запитання

- Назвіть основні етапи процесу установки середовища розробки MonoDevelop.
- Назвіть основні етапи налаштування інтегрованого середовища розробки MonoDevelop в системі Unity 3D.
- Назвіть основні функції середовища розробки MonoDevelop.
- Як здійснюється синхронізація Unity проекту з проектом MonoDevelop?
- Для чого в MonoDevelop використовується шкала Breakpoint Bar?

10.7 Тема 4: Аудіо-компоненти Unity 3D. Імпорт і налаштування звуку.

Анотація

Лекція знайомить з аудіо-компонентами Unity 3D, можливостями імпорту і відтворення звукових ефектів, методами активації звукових ефектів за допомогою програмного коду, методами роботи з AudioManager в Unity 3D.

Мета лекції

Ознайомити студентів з системою аудіо-компонентів Unity 3D. Розглянути можливості імпорту і відтворення звукових ефектів в Unity 3D, методи активації звукових ефектів за допомогою програмного коду, методи налаштування та роботи з AudioManager в Unity 3D.

Очікувані результати

Сформувати у студентів знання щодо реалізації звукового супроводження ігрових додатків за допомогою використання аудіо-компонентів Unity 3D та методів активації звукових ефектів за допомогою програмного коду.

Контрольні запитання

- Для чого в Unity 3D використовуються компоненти AudioClip, AudioSource, AudioListener?
- Яким чином в Unity 3D здійснюється активації звукових ефектів?
- Назвіть компоненти Unity 3D для роботи зі звуком.
- Для чого в Unity 3D використовується AudioManager?
- Назвіть основні етапи налаштування центрального диспетчера управління звуком в Unity 3D.

10.8 Лабораторна робота № 4. Імпорт аудіокліпів в ігрові додатки. Робота з диспетчером AudioManager в Unity 3D.

Анотація

Лабораторна робота орієнтована на оволодіння студентами методами роботи зі звуком за допомогою диспетчера AudioManager в Unity 3D.

Мета лабораторної роботи

Сформувати у студентів знання щодо налаштування центрального диспетчера управління звуком AudioManager в Unity 3D, методів розробки інтерфейсів користувача для налаштування звукових ефектів та регулювання гучності звуку, методів імпорту аудіо-кліпів в ігрові додатки.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде вміти створювати ігрові додатки зі звуковим супроводженням за допомогою диспетчера AudioManager.

Контрольні запитання

- Назвіть основні функції центрального диспетчера управління звуком AudioManager.
- Назвіть основні методи розробки інтерфейсів користувача для налаштування звукових ефектів та регулювання гучності звуку.
- Назвіть основні етапи імпорту аудіо-кліпів в ігрові додатки.
- В чому полягає відмінність 2D і 3D звуку?
- За допомогою якого компонента можна легко регулювати на глобальному рівні гучність звукових ефектів?
- Для чого в Unity 3D використовуються файли формату WAV?

10.9 Тема 5: Анімація об'єктів в Unity 3D.

Анотація

Лекція знайомить з основними поняттями анімації та типами анімації об'єктів в Unity 3D.

Мета лекції

Ознайомити студентів з основними типами анімації об'єктів в Unity 3D. Розглянути методи анімації твердого тіла, анімації на основі скелета, анімації спрайтами (окремі спрайти, атлас спрайтів), анімації, що заснована на фізиці (використання фізичної системи Unity), відеоанімації (відтворення відеофайлів у вигляді анімованих текстур), анімації частинками, анімації за допомогою системи Mecanim, анімації персонажів.

Очікувані результати

Сформувати у студентів знання щодо реалізації в ігрових додатках основних типів анімації об'єктів, анімації персонажів, застосування анімації предметів за допомогою системи Mecanim.

Контрольні запитання

- Для чого в Unity 3D використовуються система Mecanim?
- Яким чином в Unity 3D здійснюється анімація спрайтів?
- Назвіть типи анімації об'єктів в Unity 3D.
- Що таке атлас спрайтів?
- Яким чином в Unity 3D здійснюється відтворення відеофайлів у вигляді анімованих текстур?

10.10 Лабораторна робота № 5. Анімація спрайтами. Робота з системою анімації Mecanim.

Анотація

Лабораторна робота орієнтована на оволодіння студентами методами анімації за допомогою системи Mecanim, методами анімації спрайтами (окремі спрайти, атлас спрайтів).

Мета лабораторної роботи

Сформувати у студентів знання щодо методів розробки анімованих персонажів, технології анімації за допомогою системи Mecanim, методів анімації спрайтами (окремі спрайти, атлас спрайтів) в Unity 3D.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде вміти створювати ігрові додатки з анімованими персонажами, застосовувати методи анімації спрайтами при розробці ігрового контенту.

Контрольні запитання

- Яким чином здійснюється імпорт спрайтів та їх налаштування?
- Для чого використовується режим Sprite Mode?
- Для чого використовуються актив Animation Controller?
- Яким чином здійснюється створення графа системи Mecanim для анімації предмету?
- Для чого в Unity 3D використовуються тригери?
- Яким чином здійснюється тестування тригерів?
- Для чого використовуються панель умов «Conditions» в інспекторові об'єктів?

10.11 Тема 6: Технологія розробки крос-платформних комп'ютерних ігор в Unity 3D.

Анотація

Лекція знайомить з технологією створення ігрових додатків для різних платформ (Windows, Android) в Unity 3D.

Мета лекції

Ознайомити студентів з технологією створення крос-платформних комп'ютерних 3D-ігор (Windows, Android).

Очікувані результати

Сформувати у студентів знання щодо технології розгортання ігрових додатків на різних платформах без прив'язки до Unity 3D.

Контрольні запитання

- Для яких платформ можна створювати ігрові додатки в Unity 3D ?
- Для чого використовується команда BuildSettings?
- Назвіть відмінності генерації пакетів для мобільних пристроїв.
- Для чого використовується параметр BundleIdentifer?
- Назвіть основні етапи налаштування інструментів збірки для Android ?

10.12 Лабораторна робота № 6. Розробка крос-платформних комп'ютерних ігор (Windows, Android).

Анотація

Лабораторна робота орієнтована на отримання навиків налаштування інструментів збірки ігрових проектів для платформ Windows та Android та генерації ігрових пакетів за допомогою відповідних інструментів.

Мета лабораторної роботи

Сформувати у студентів знання щодо технології розробки крос-платформних комп'ютерних ігор в Unity 3D.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде вміти здійснювати налаштування інструментів збірки ігрових проектів для платформ Windows та Android та генерувати ігрові пакети за допомогою відповідних інструментів.

Контрольні запитання

- Назвіть особливості налаштування інструментів збірки ігрових проектів для Android?
- Яким чином в Unity 3D здійснюється генерація файлів формату APK?
- Назвіть особливості створення крос-платформних комп'ютерних ігор в Unity 3D?
- Назвіть особливості налаштування інструментів збірки ігрових проектів для Windows?
- Які чого потрібен список TargetPlatform?

Вступ

Лекції є основною формою проведення навчальних занять, що призначені для засвоєння теоретичного матеріалу.

При проведенні лекцій навчального модуля «Розробка комп’ютерних ігор за допомогою Unity 3Dp» лектор повинен дотримуватись таких вимог:

1. Доведення до студентів мети лекції та належне її мотивування. Це виховує в них уміння одразу, без зволікань, залучатися до процесу слухання лекції.

2. Доступність і науковість викладу. Доступність передбачає врахування рівня студентів, їх індивідуальних особливостей, а науковість – розкриття причинно-наслідкових зв'язків, явищ, подій, проникнення в їх сутність, міждисциплінарні зв'язки тощо. Матеріал має бути цікаво вибудуваний, щоб легко сприймався і повніше й всебічніше усвідомлювався студентом. Викладач має відстежувати, що зі сказаного ним і якою мірою сприйнято аудиторією, чи не виникли у слухачів запитання через недостатнє розуміння змісту лекції, непідготовленість до її сприйняття; чи встигають вони усвідомити кожне нове положення, чи вміють поєднувати нову інформацію з попередньою тощо.

3. Включення механізму зворотного зв'язку. Це дає змогу лектору не лише контролювати рівень сприймання, а й регулювати процес роздумів залежно від реального стану студентів.

4. Повторення важливих теоретичних положень. Такі повтори підвищують імовірність запам'ятовування, а отже, і розуміння, систематизацію матеріалу, який ґрунтується на міцному фундаменті засвоєних фактів.

5. Завершення кожного питання лекції підсумком і мотивованим переходом до наступного.

6. Емоційність викладу. Вона є засобом мобілізації і підтримання уваги студентів. Емоційність досягається насамперед чіткою, живою, образною, інтонованою мовою викладача. їй сприятимуть також афоризми, вдалі аналогії, ідіоматичні вирази.

7. Налагодження живого контакту. Йдеться про вміння викладача тримати в полі свого зору кожного студента, своєчасно і правильно реагувати на їх міміку, репліки, жести, вдало використати жарт, дотеп, гумор. Такі засоби зближують викладача з аудиторією і сприяють створенню настрою для більш осмисленого сприймання змісту лекції.

8. Створення проблемних ситуацій. Усвідомлення студентами проблеми налаштовує їх на її розв'язання, спонукає до роздумів, активізує їх пізнавальну діяльність.

Рівень та послідовність викладення лекційного матеріалу відповідає освітньо–професійній програмі підготовки фахівців на першому

(бакалаврському) рівні вищої освіти галузі знань 12 «Інформаційні технології», спеціальності 121 «Інженерія програмного забезпечення» за спеціалізацією «Розробка комп'ютерних ігор».

Зміст лекційного матеріалу характеризується достатньою простотою, що дозволяє сподіватися на його глибоке засвоєння студентами.

ЛЕКЦІЯ №1
ТЕМА: «ОСНОВИ РОБОТИ В UNITY 3D»

Анотація

Лекція знайомить з основами роботи в Unity 3D - системою розробки крос-платформних 2D- і 3D-ігор та інтерактивного контенту.

Мета лекції

Ознайомити студентів з перевагами та недоліками розробки ігрових додатків та ігрового контенту за допомогою Unity 3D; з новим функціоналом та відповідним інструментарієм системи Unity 3D, який допоможе розробникам успішно створювати анімаційні та ігрові сцени; сферами застосування комп’ютерних ігор, розроблених за допомогою Unity 3D.

Очікувані результати

Студент оволодіє поняттями: комп’ютерна крос-платформна 2D- і 3D-гра, інтерактивний контент, ігрова сцена. Буде знати основні переваги та недоліки розробки ігрових додатків та ігрового контенту за допомогою Unity 3D, функціонал та відповідний інструментарій системи Unity 3D для створення анімаційних та ігрових сцен, сфери застосування комп’ютерних ігор, розроблених за допомогою Unity 3D.

Вступ

Unity - це інструмент для розробки двох- і тривимірних ігрових додатків.

В двомірних (2D) іграх ігровий простір складається з готових плоских зображень. Завдання комп'ютера зводиться тут лише до комбінування цих зображень відповідно до алгоритмів гри, чим пояснюються низькі вимоги двомірних ігор до потужності комп'ютера. Свобода пересування персонажа в двомірних іграх обмежена: він може пересуватися лише за передбаченим розробниками маршрутом.

Можливі проміжні варіанти між 2D і 3D (іноді для них використовується термін 2.5D):

1. Простір гри тривимірний, але моделі ігрових об'єктів - будівель, рослин, персонажів - представляють собою плоскі картинки, спрайти. Приклад - ранні тривимірні ігри, такі як Wolfenstein 3D і Doom. В сучасних тривимірних іграх спрайти використовуються лише для відображення об'єктів, присутніх в дуже великій кількості (наприклад, трави та дерев), а також об'єктів, віддалених від персонажа на велику відстань. Це здійснюється для економії

обчислювальних ресурсів комп'ютера: наприклад, моделювання повністю тривимірного дерева з тисячами листів - дуже складне завдання для комп'ютера, а моделювання цілого лісу - завдання нездійсненне, і без спрайтів тут обійтися неможливо.

2. Простір гри двовимірний, але моделі персонажів виконані тривимірними. Така технологія використовується в багатьох квестах. Вона дозволяє поєднати високоякісний двовимірний рисований фон (як правило, частково анімований) з «живими» персонажами, які вільно рухаються і при цьому невимогливі до потужності комп'ютера. Яскравий приклад такої гри – «Syberia».

Комп'ютерна тривимірна гра (3D, англ. Three-dimensional) - гра, візуальний простір яке цілком побудовано з тривимірних об'єктів. Персонаж знаходиться в тривимірному просторі і в деяких іграх має повну свободу пересування.

Управління в тривимірних іграх, як правило, здійснюється одночасно клавіатурою і мишкою. За допомогою кнопок можна здійснювати рух персонажа в потрібному напрямку (зазвичай це клавіші W, S, A, D, рідше - клавіші-стрілки). Рух миші змушує персонажа повертатися. Натискання лівою кнопкою миші відповідає якій-небудь дії - в залежності від жанру гри це може бути, наприклад, удар мечем, взаємодія з предметом або іншим персонажем. Натискання правою кнопкою миші, як правило, в шутерах і рольових іграх відповідає блокуванню атаки або альтернативній атаці, а в квестах - відкриття меню. Клавіша Space зазвичай відповідає стрибку, а одночасне натискання комбінації клавіш Shift і W змушує персонажа бігти. Призначення інших клавіш клавіатури індивідуально для кожної гри.

Створені за допомогою Unity програми працюють під операційними системами Windows, OS X, Windows Phone, Android, Apple iOS, Linux, а також на ігрових приставках Wii, PlayStation 3, PlayStation 4, Xbox 360, Xbox One і MotionParallax3D дисплеях (пристрої для відтворення віртуальних голограм), наприклад, Nettlebox.

Unity також має можливість створювати додатки для запуску в браузерах за допомогою спеціального модуля Unity (Unity Web Player) за допомогою реалізації технології WebGL.

Додатки, створені за допомогою Unity, підтримують DirectX і OpenGL.

Таким чином, за допомогою Unity можна створювати крос-платформні комп'ютерні ігри.

Під крос-платформністю мається на увазі здатність програмного забезпечення працювати більш ніж на одній апаратній платформі і (або) операційній системі.

Розглянемо далі поняття інтерактивності та інтерактивного ігрового контенту.

Інтерактивність (від англ. Interaction - «взаємодія») - поняття, яке розкриває характер і ступінь взаємодії між об'єктами або суб'єктами.

В інтерактивних системах додаток складається з одних і тих же об'єктів і під час розробки, і під час виконання. Мало того, відсутність розбиття на інструментальну і виконавчу середовище дозволяє використовувати одні і ті ж засоби і під час розробки, і під час виконання, тому можна змінювати працюючий додаток і негайно бачити результат цієї зміни.

Контент - будь-який вид інформації (текст, аудіо, відео, зображення, текстури, матеріали), що становить зміст програмного продукту.

Інтерактивний ігровий контент - це будь-який контент, який має на увазі активну участь користувачів і, який спонукає їх до вчинення певних дій.

Огляд нових можливостей Unity 2017

Офіційний сайт Unity https://unity3d.com

Версії Unity:
Personal free for beginners, students and hobbyists
Plus 35 $ month for serious creators
Pro 125 $ month for professionals and studios

У грудні 2016 року, компанія Unity Technologies заявила, що поточна основна версія Unity 5.6 буде останньою в цій лінійці, після чого Unity перейде на новий формат оновлень. Нова версія буде називатися Unity 2017, а наступні за нею версії нумеруватимуться так: 2017.1.0, 2017.2.0 і т.д.

Розробники акцентують увагу на трьох нових функціях Unity 2017: **Timeline** (тимчасова шкала), **Cinemachine** (набір «розумних» камер) і **Post-Processing Stack** (набір інструментів для пост-обробки).

Давайте розглянемо їх більш уважно.

Timeline - дуже зручний інструмент для створення анімації, який вже давно є в безлічі відповідних продуктів. За допомогою Timeline можна швидко створювати анімації трансформації, анімації інтерфейсу і т.п. Все здійснюється за допомогою стандартних ключових кадрів, між якими Unity сама змінює стан ваших об'єктів.

Timeline - це відмінна альтернатива і доповнення до стандартних кліпів анімації. Особливо зручно створювати прості cut-сцени, інтро- і інші події, які роблять вашу гру більш цікавою. Крім того, в Timeline можна працювати з аудіо- і прив'язувати аудіо-кліпи до подій.

Відкрити Timeline Editor можна за допомогою меню «Window→ Timeline Editor» (рис. 1.2 -1.3). Зауважимо, що спочатку до ігрової сцени треба додати 3D - об'єкт.

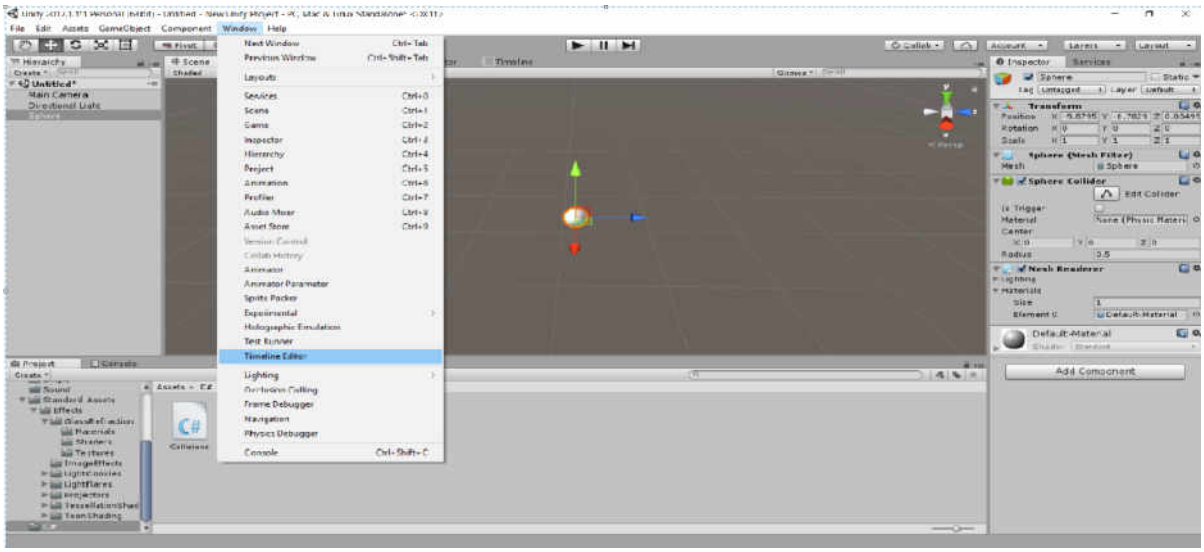

Рисунок 1.1 – Меню «Window→ Timeline Editor»

Cinemachine - набір розумних камер, за допомогою яких ви можете робити cut-сцени з використанням камер, ролики на движку, а також пов'язувати все це з можливостями **Timeline**.

Разом з **Cinemachine** в Unity вперше виникає поняття віртуальної камери, за допомогою якої ви можете майстерно управляти камерами на сцені і робити відмінні переходи між сценами. Крім того, віртуальні камери дають можливість слідувати за персонажем без використання скриптів типу SmoothFollow. Все доступно «з коробки».

Найприємніше - те, що **Cinemachine** дозволяє робити безліч ефектів пов'язаних з камерами (тряска, хитання і т.п) без використання сторонніх Tween плагінів і без єдиної строчки коду.

Інструмент **Post-Processing Stack**, який Unity позиціонує як джерело потужного функціоналу для дизайнерів, які хочуть поліпшити візуальний ряд своєї гри. **Post-Processing Stack** служить для додавання пост-ефектів. У попередніх версіях Unity вже була реалізована така можливість: пост-ефекти можна було додавати на камери у вигляді окремих компонентів, але Post-Processing Stack - це єдиний механізм, в якому зібрані і доступні для виклику такі ефекти як згладжування, змазування, світіння і т. п. Зручність цього інструменту в тому, що вам більше не потрібно звертатися до кожного з них окремо. Замість цього представлений стек ефектів, які можна змінювати як завгодно.

Також до Unity 2017 додано **Asset Bundle Browser**, за допомогою якого можна легко і зручно створювати **Asset Bundle** для завантаження ваших ресурсів. Для завантаження **Asset Bundle Browser** до Unity – проекту необхідно скористатися Asset Store (меню **Window → Asset Store**).

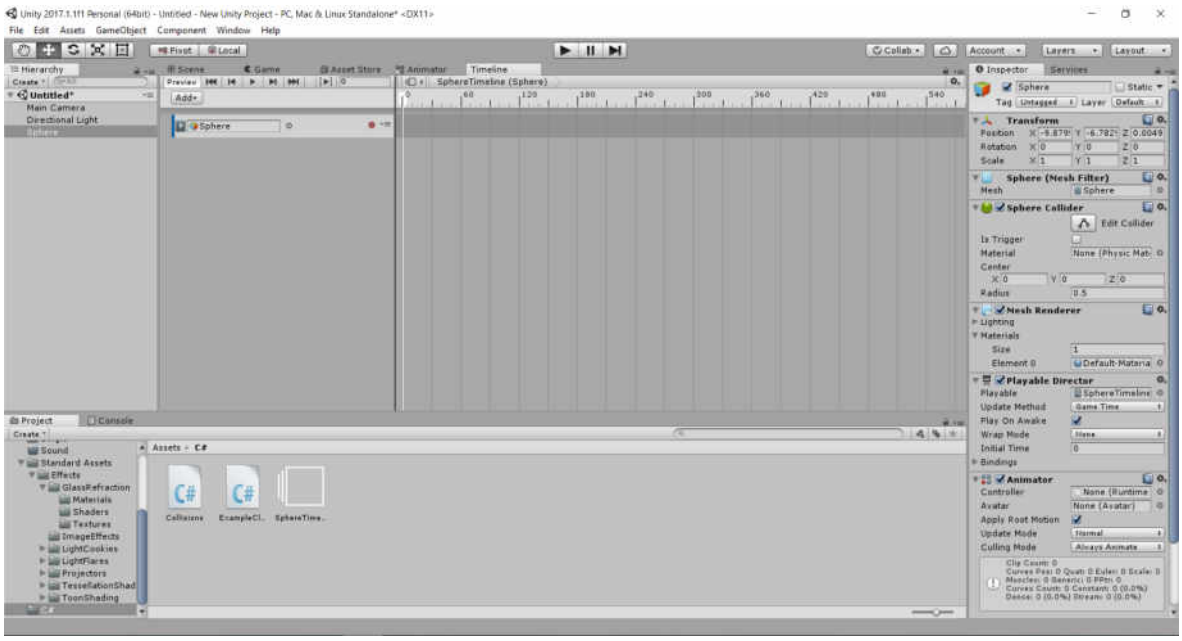

Рисунок 1.2 – Timeline Editor для об'єкту Sphere

Asset Store - це бібліотека, в якій зібрані безкоштовні і комерційні Asset, створені як Unity Technologies, так і членами спільноти Unity. Доступний великий вибір Asset - текстури, моделі та анімації, приклади проектів, підручники і розширення для редактора. Asset доступні через простий інтерфейс, вбудований в редактор Unity, через нього можна завантажити і імпортувати безпосередньо в ваш проект.

Оберіть в **Asset Store** браузер **Asset Bundle Browser** та завантажте його за допомогою кнопки **Download** до Unity – проекту (рис. 1.3).

Відкрити **Asset Bundle Browser** можна за допомогою меню **Window → Asset Bundle Browser** (рис. 1.4).

Крім того, в Unity 2017:

Вдосконалено систему відображення тіней в реальному часі і робота зі світлом в цілому.

Додано механізм **Analytics Event Tracker**, за допомогою якого можна швидко і без використання коду додавати події для відстеження за допомогою Unity Analytics.

З'явився **Unity Teams**, який складається з Unity Collaborate і Cloud Build. Тепер можна спільно редагувати сцени, скрипти і префаби в загальному доступі. Додатково до цього Unity Editor оновлено для більш зручної роботи з загальними ресурсами

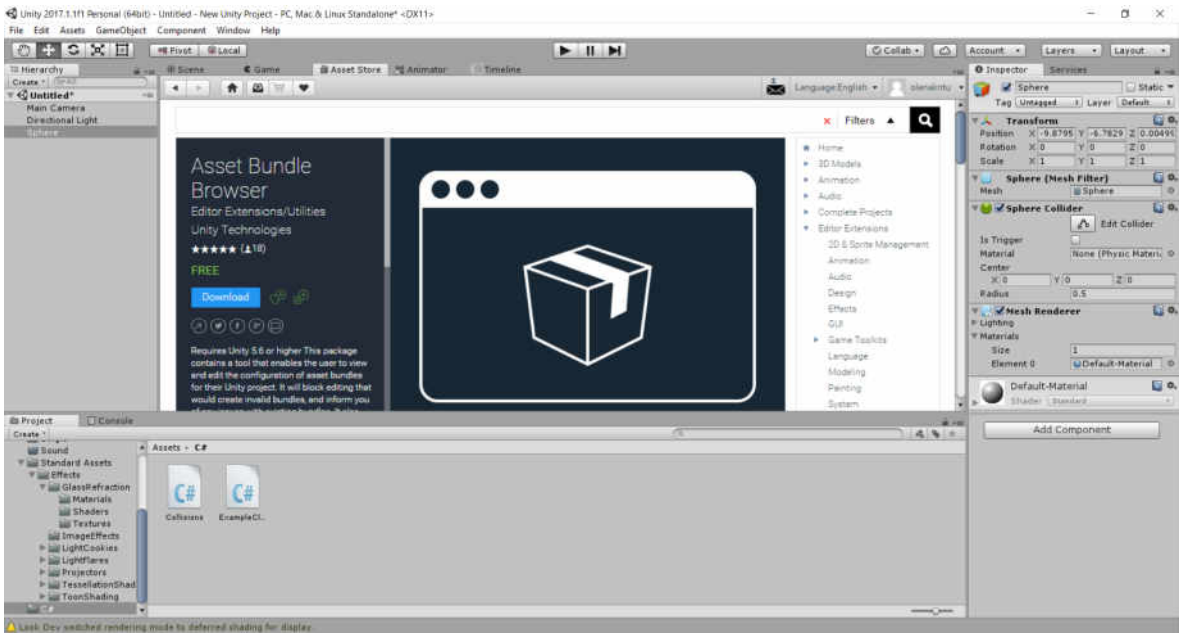

Рисунок 1.3 – **Asset Store** браузер **Asset Bundle Browser**

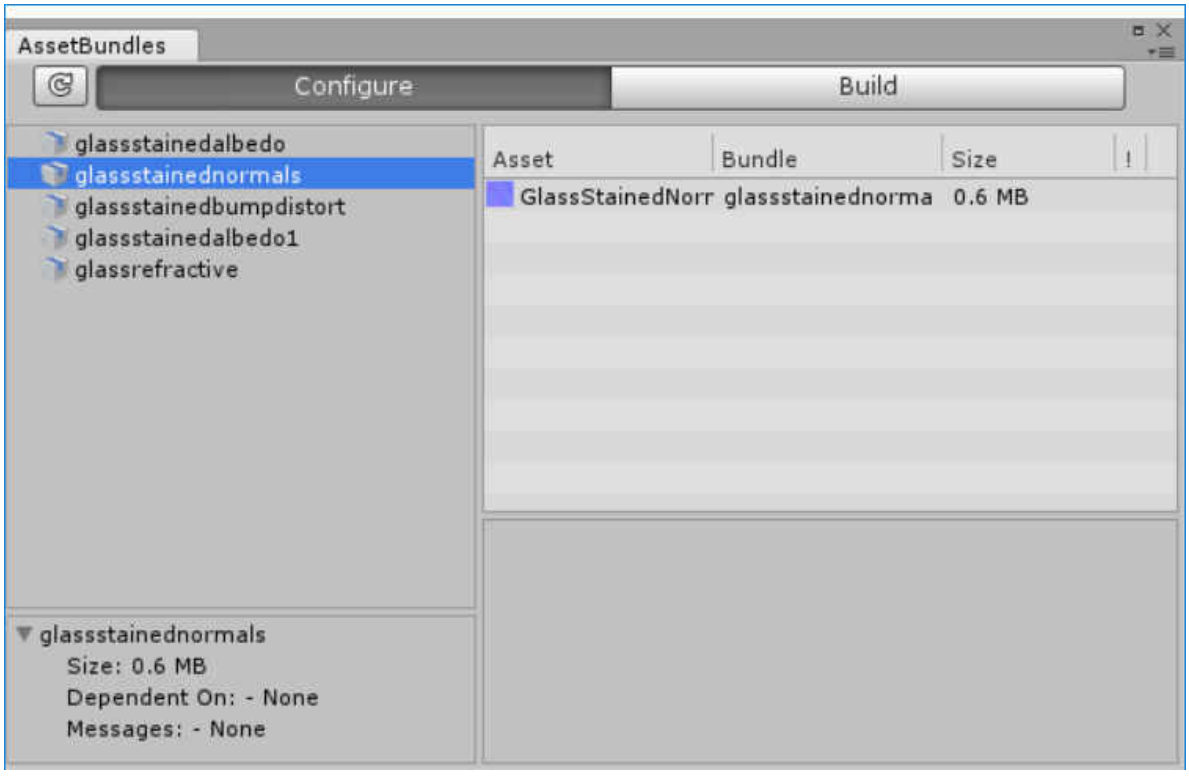

Рисунок 1.4 – **Asset Bundle Browser**

Додано технологію **Ambisonic audio** для створення звукового оточення на всі 360°. Можна буде використовувати джерела звуку не тільки справа і зліва від гравця, але і вище або нижче. Ця система також може використовуватися в 360 ° відео.

Додано **UI профайлер**, який дозволяє оцінити, наскільки вантажить гру ваш інтерфейс.

Покращено підтримку **Visual Studio**, включаючи Visual Studio для Mac OS.

Додано **Universal Windows Platform** замість Windows Store. Universal Windows Platform включає в себе білди для Xbox One, Windows 10, Windows Phone 10 і HoloLens.

Основи роботи в Unity 2017

Редактор Unity має простий Drag&Drop інтерфейс, який складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Движок підтримує дві скриптові мови: C#, JavaScript (модифікація). Розрахунки фізики здійснює фізичний движок PhysX від NVIDIA.

Проект в Unity ділиться на **сцени (рівні)** - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, власне, об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. У будь-якого об'єкта на сцені обов'язково присутній компонент Transform - він зберігає в собі координати місця розташування, повороту і розмірів об'єкта по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель об'єкта видимою.

Кожен Unity - проект містить папку **Assets**. Вміст цієї папки подано в **Project View**. Це ресурси гри: скрипти, 3D-моделі, текстури, аудіофайли, префаби (Prefabs). Можна відкрити папку, яка містить ресурс через стандартний Провідник Windows, натиснувши правою кнопкою миші по ньому і вибравши **Reveal in Exlorer**.

Слід зауважити, що при переміщенні ресурсів зовнішніми інструментами (наприклад, Провідником) будуть втрачені всі метадані, пов'язані з ним.

Метадані зберігають інформацію про ресурс і його зв'язки з іншими ресурсами. Завжди переміщайте ресурси тільки в **Project View**.

Існує два способи додавання ресурсу в проект:

1. Перетягнути файл з Провідника в Project View.

2. На панелі Project View оберіть Assets → Import New Assets.

Ресурс буде автоматично імпортовано в проект і він стане доступний для використання.

Сцени так само зберігаються в папці **Assets** і відображаються в **Project View**. Їх можна вважати окремими рівнями.

Створити нову сцену можна за допомогою меню **File → New Scene (Ctrl + N)**. Збереження сцени можна здійснити за допомогою меню **File → Save Scene (Ctrl + S)**.

Деякі ресурси створюються безпосередньо в Unity. Для цього використовується меню **Create** в **Project View** (рис. 1.6).

Меню **Create** дозволяє додавати в проект скрипти, префаби, папки та інше. Будь-який ресурс або папку можна перейменувати: F2 або два кліка по імені. Якщо затиснути Alt, то при розкритті директорії будуть розкриті і всі піддиректорії.

Панель **Hierarchy** містить всі об'єкти (GameObject) відкритої сцени. Деякі з них - це екземпляри ресурсів (3D-моделі та інше), інші - екземпляри префабів. У Hierarchy об'єктів можна задавати успадкування. Об'єкти, що додаються в сцену або видаляються з неї, відображаються або навпаки перестають відображатися в Hierarchy (рис. 1.7).

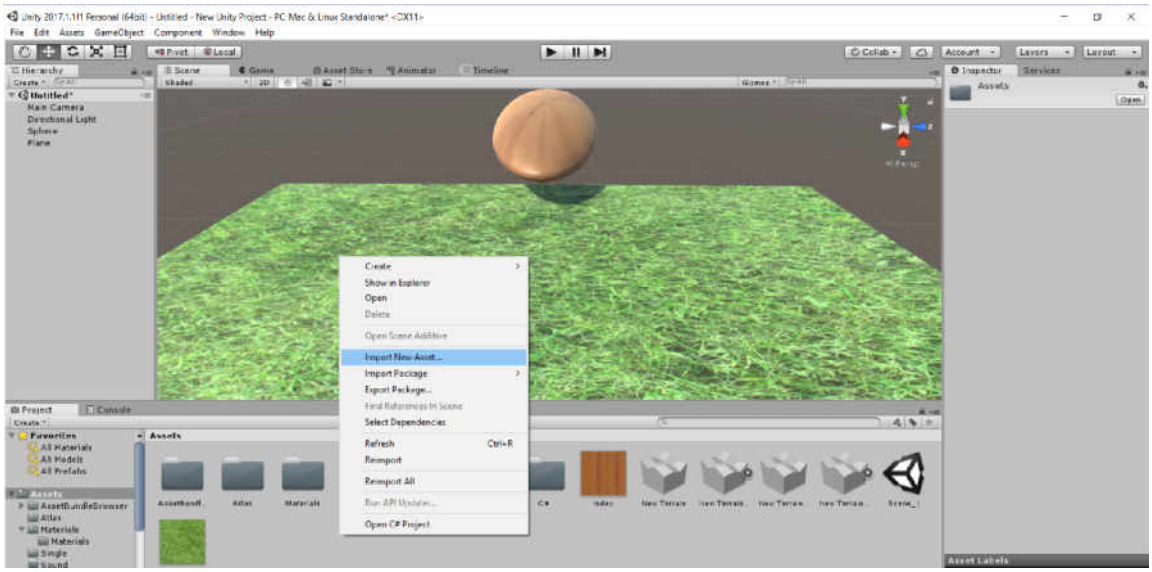

Рисунок 1.5 – **Import New Assets**

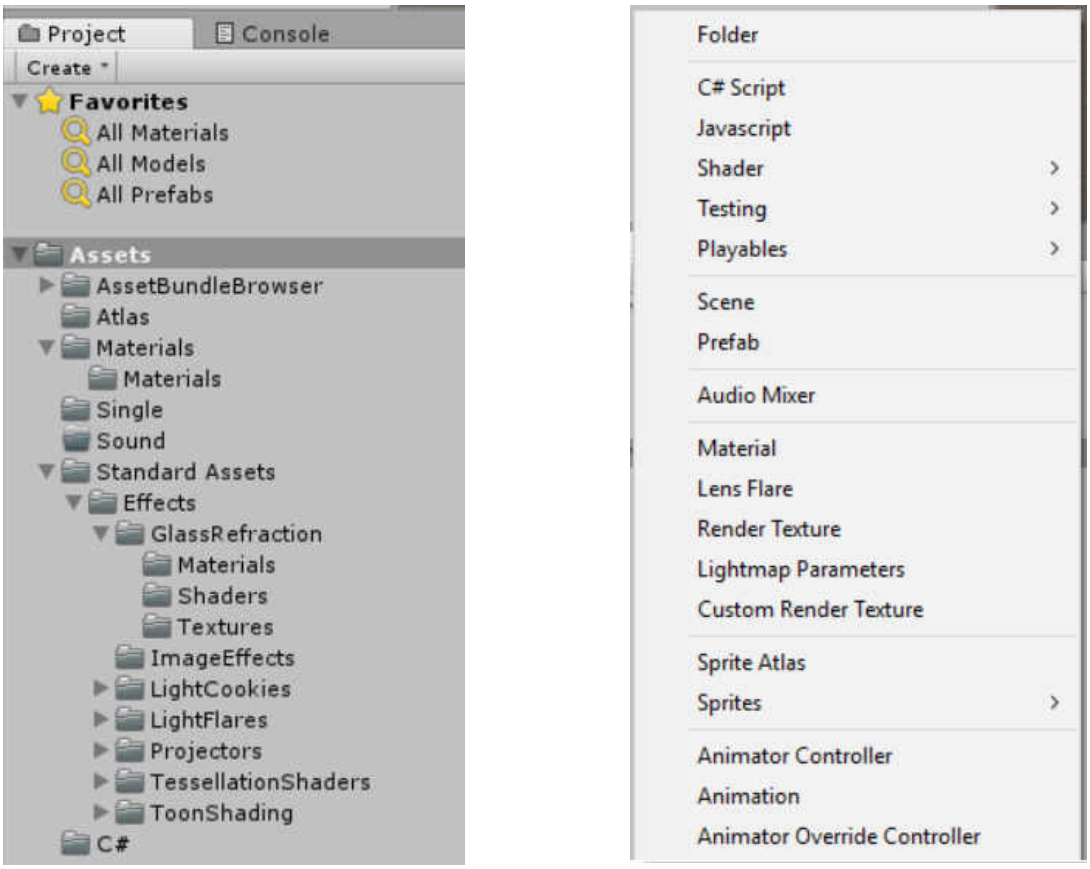

Рисунок 1.6 – Меню **Create** в **Project View**

Рисунок 1.7 – Панель **Hierarchy**

Unity використовує концепцію успадкування (Parenting). Будь-який об'єкт може бути дочірнім по відношенню до іншого. Для цього достатньо перетягнути його на «батька» в Hierarchy. Дочірній об'єкт буде рухатися і обертатися разом з батьком (рис. 1.8).

а) б)

Рисунок 1.8 – Панель **Hierarchy:** а) два незалежні об'єкти; б) один об'єкт є дочірнім.

Панель інструментів (**Toolbar**) складається з п'яти базових груп інструментів. Кожна група відповідає за свою ділянку редактора.

Рисунок 1.9 – Панель інструментів **Toolbar**

Transform Tools - інструменти трансформації об'єктів, використовуються в **Scene View**.

Transform Gizmo Toggles - інструменти для роботи з Гизмо-контейнером, використовуються в Scene View.

Play/Pause / Step Buttons - інструменти перегляду гри, використовуються в Game View.

Layers Drop-down - визначає, які об'єкти будуть відображатися в Scene View.

Рисунок 1.10 – **Layers Drop-down**

Layout Drop-down - визначає взаємне розташування секторів вікна.

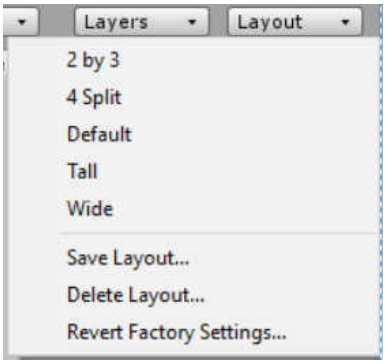

Рисунок 1.11 – **Layout Drop-down**

Scene View використовується для позиціонування об'єктів (оточення, персонажі, камери, системи частинок та інше).

Ось деякі прийоми роботи в Scene View:

1. Затиснута ПКМ активує режим вільного польоту.
2. Переміщатися можна клавішами WASD на манер FPS.
3. Виберіть об'єкт і натисніть F. Scene View буде центрований і масштабований по об'єкту.
4. При затиснутому Alt, ЛКМ буде крутити камеру навколо поточної точки опори.
5. При затиснутому Alt, СКМ буде переміщати камеру.
6. При затиснутому Alt, ПКМ буде масштабувати Scene View.
7. *Альтернативний режим переміщень - клавіша Q.

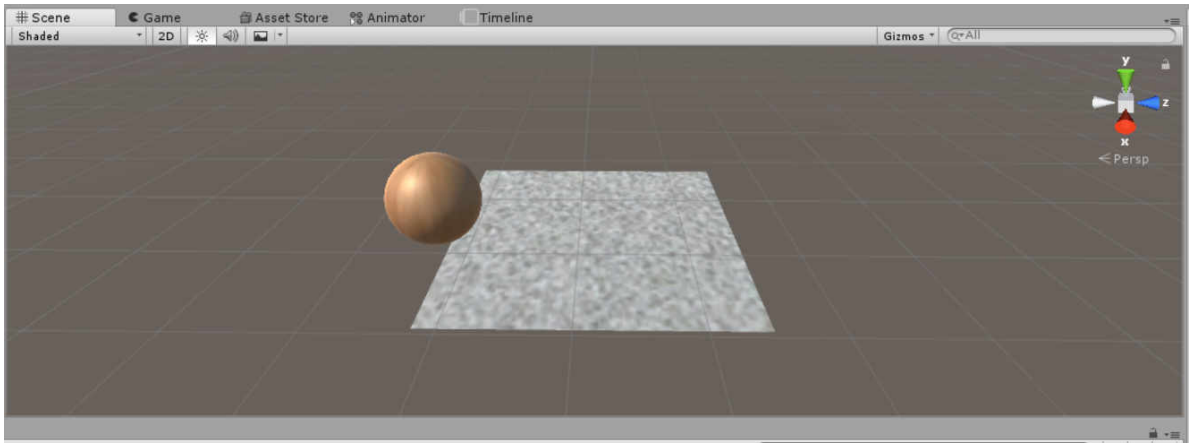

Рисунок 1.12 – **Scene View**

Positioning GameObjects

Для розміщення об'єктів в ігровому світі використовуються інструменти групи Transform Tools: Translate, Rotate і Scale. Кожен об'єкт має контейнер Гизмо, описаний навколо нього. Маніпулювати об'єктами можна

використовуючи «ручки» Гизмо або вводячи числові значення у відповідні властивості компонента Transform в Inspecor View.

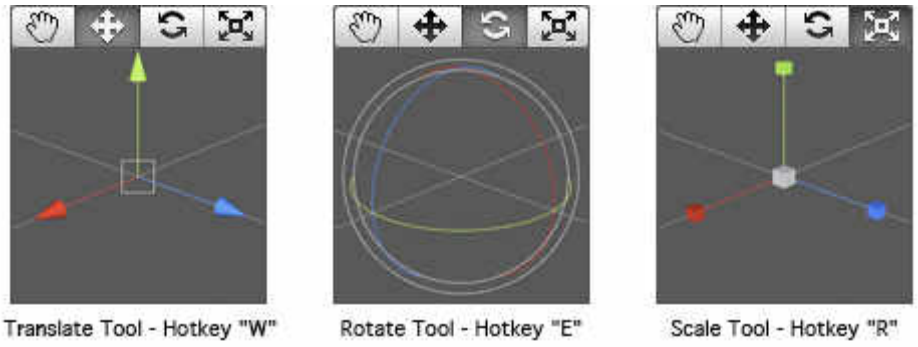

Рисунок 1.13 – **Positioning GameObjects**

Затиснутий Alt при переміщенні Гизмо активує прив'язку відповідно до налаштувань **Snap Settings (Edit → Snap Settings ...).**

Центр Гизмо дозволяє переміщати його по всіх осях одночасно.

Затиснутий Ctrl при переміщенні Гизмо активує прив'язку до перетину з колайдером.

СКМ дозволяє переміщати Гизмо по осі, яка підсвічується в даний момент без натискання на цю вісь.

Gizmo Display Toggles використовується для завдання позиції Гизмо.

Position:

Center розміщує Гизмо в центрі об'єкта, який візуалізується.

Pivot розміщує Гизмо на точці опори.

Rotation:

Local повертає Гизмо паралельно осях локальної системи координат об'єкту.

Global повертає Гизмо паралельно осях світової системи координат.

У верхньому правому куті Scene View розташовується Scene Gizmo. Він позначає положення камери сцени і дозволяє швидко змінювати ракурс огляду. Натискання на ручки **Scene Gizmo** переміщує камеру в відповідну позицію (вид зверху, зліва, спереду і т.д.) Або включає ізометричний режим (Isometric Mode). Включення/вимкнення ізометричного режиму здійснюється за допомогою комбінації клавіш Shift + СКМ.

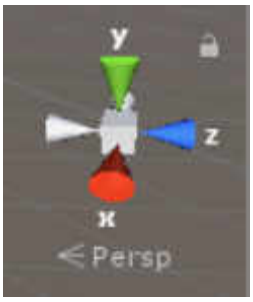

Рисунок 1.14 – **Scene Gizmo**

Scene View Control Bar

Рисунок 1.15 – **Scene View Control Bar**

Перше меню, що випадає дозволяє вибрати режим відтворення (Draw Mode) Scene View.

Варіанти: текстуровані об'єкти (Shaded), каркасна сітка (Wireframe) і комбінований варіант (Shaded Wireframe).

Далі йдуть дві кнопки: **Scene Lighting and Game Overlay**.

Включення Scene Lighting замінить освітлення сцени. Game Overlay включає відображення в сцені елементів на кшталт неба (Skyboxes) і інтерфейсу (GUI Elements).

Game View - рендер з ігрової камери, попередній перегляд гри. Можна використовувати одну камеру або більше.

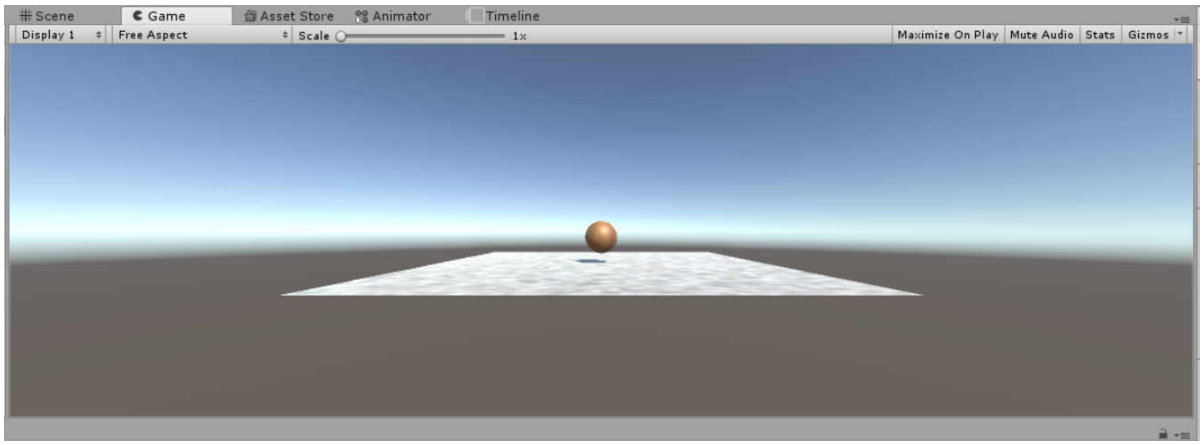

Рисунок 1.16 – **Game View**

Play Mode

Три кнопки відповідають за управління попереднім переглядом гри. Play, Pause і Step.

Всі зміни, зроблені під час попереднього перегляду, скидаються при виході з нього. Винятком є зміни в префаб.

Game View Control Bar

Рисунок 1.17 – **Control Bar**

Free Aspect в Game View - це контроль пропорцій зображення (Aspect Drop-down). На деяких дисплеях це співвідношення відрізняється від стандартного 4:3 (наприклад, на широкоформатних моніторах - 16:10).

Кнопка **Maximize on Play**. Якщо вона натиснута, то Game View розтягується на все вікна редактора при попередньому перегляді.

Кнопка **Gizmos** включає відображення контейнерів Гизмо в Game View.

Кнопка **Stats** показує статистку рендеринга (Rendering Statistics), корисну при оптимізації гри.

Inspector

Об'єкт може містити в собі меши (meshes), скрипти (scripts), звуки, джерела світла (Lights) та інші елементи. Inspector відображає детальну інформацію про виділений об'єкт і всіх прикріплених до нього компонентах (Components), а також їх властивості. Також тут можна редагувати ці властивості (рис. 1.18).

Будь-які властивості, які відображаються в Inspector, можуть бути змінені там же. Це відноситься і до змінних скриптів, прикріплених до об'єкта.

Можна використовувати Inspector для зміни змінних під час попереднього перегляду. Якщо в скрипті визначена публічна змінна об'єктного типу, можна перетягнути об'єкт або префаб прямо в Inspector і зробити привласнення значення.

Натискання на знак питання праворуч від назви компонента в Inspector викличе сторінку довідки по цьому компоненту.

Status Bar

Рядок стану Status Bar розташований у нижній частині вікна редактора. Він відображає помилки компіляції і логи Debug. Якщо виникають проблеми з грою, варто заглянути в рядок стану. Подвійне натискання на нього викличе вікно консолі (Console), в якому відображаються всі помилки.

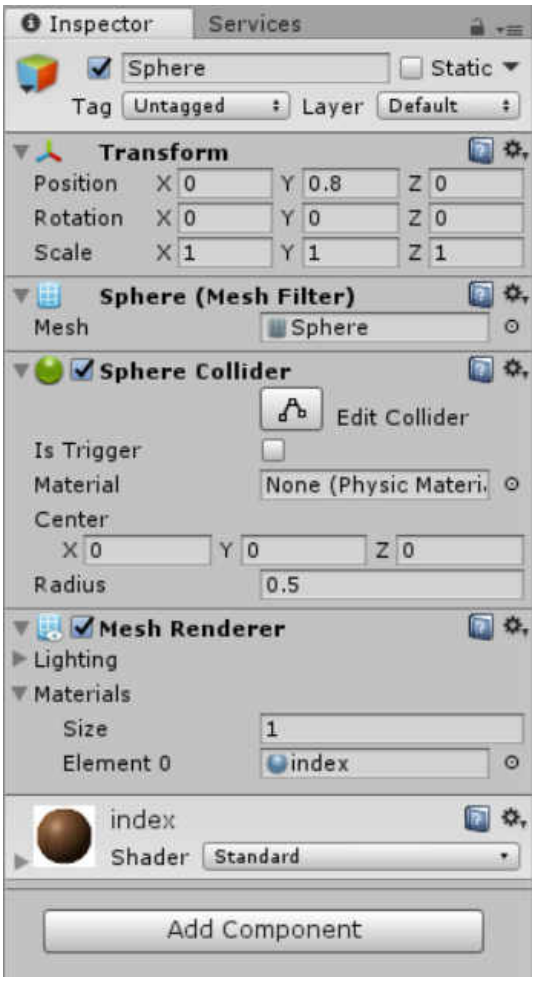

Рисунок 1.18 – **Inspector**

Console

Подвійне натискання на помилку в рядку стану або команда **Window → Console** викличе консоль (рис. 1.19).

Консоль показує повідомлення, попередження і помилки, а так само логі Debug. Можна надсилати власні повідомлення в консоль, використовуючи функції Debug.Log () і Debug.Error (). Подвійний клік по повідомленням або помилку відкриє скрипт, при виконанні якого вони виникли.

В Unity можна налаштовувати розташування (Layout) секторів, перетягуючи їх за закладки. Якщо перетягнути закладку в область закладок вже існуючого вікна, то вона буде додана до присутніх там закладок. Також можна прикріпити сектор до краю екрану або краю іншого сектора.

Закладки можуть відкріплятися від головного вікна редактора і включатися до складу плаваючого вікна редактора. Плаваюче вікно може містити сектора і закладки так само, як і головне вікно.

Коли розташування секторів задано, його можна зберегти і завантажити в потрібний момент через меню Layout (Save і Load).

Рисунок 1.19 – **Console**

Контрольні запитання

- Дайте визначення поняттям «комп'ютерна крос-платформна 3D-гра», «інтерактивний контент», «ігрова сцена».
- Назвіть основні переваги розробки ігрових додатків та ігрового контенту за допомогою Unity 3D.
- Який інструментарій Unity 3D використовується для створення анімаційних та ігрових сцен?
- У чому полягають відмінності крос-платформної розробки комп'ютерних ігор?
- Назвіть сфери застосування комп'ютерних ігор, розроблених за допомогою Unity 3D.

ЛЕКЦІЯ №2
ТЕМА: «ФІЗИКА 3D - ОБ'ЄКТІВ»

Анотація

Лекція знайомить з основними компонентами, які використовуються для роботи з фізикою 3D - об'єктів.

Мета лекції

Ознайомити студентів з основними компонентами Unity 3D для роботи з фізикою 3D - об'єктів. Ознайомити студентів з властивостями компонентів Rigidbodies, Colliders, Joints, Character Controllers та можливостями, які вони надають розробникам ігрового контенту.

Очікувані результати

У разі успішного виконання лабораторної роботи студент буде знати та вміти застосовувати основні компоненти та їх властивості для роботи з фізикою 3D - об'єктів при розробці ігрових додатків.

Компоненти, які використовуються для роботи з фізикою 3D - об'єктів

Unity використовує движок **Ageia PhysX** від **NVIDIA** для симуляції фізики.

Ageia PhysX є першим процесором, спеціально розробленим з метою зробити революцію в світі комп'ютерних ігор завдяки неймовірним фізичним взаємодіям в грі, що дозволяє отримати неперевершену динаміку і реалістичність.

Сьогодні процесор Ageia PhysX є ключовим апаратним елементом у забезпеченні оптимізованої ігрової фізики. Його взаємодіючі PhysX ядра, що володіють неймовірними можливостями для паралельних обчислень, оптимізовані під обробку динамічної і великомасштабної фізики, щоб забезпечити прискорення фізичного руху і взаємодії на рівні, що сильно перевищує масштаби і якість, які надають звичайні процесори.

Rigidbodies (Тверде тіло)

Для того, щоб підпорядкувати об'єкт законам фізики в Unity 3D, необхідно додати до цього об'єкта компонент **Rigidbody**. Тоді на об'єкт буде діяти гравітація і він буде стикатися з іншими об'єктами.

Додати компонент **Rigidbody** до об'єкта можна за допомогою кнопки «**Add Component→Physics→ Rigidbody**». Доступ до властивостей компоненту **Rigidbody** відкрито на панелі «**Inspector**».

Компонент **Rigidbody** має властивість «**Is Kinematic**», яка дозволяє вилучити об'єкт з-під контролю фізичного движка, і дозволити переміщати його кінематично за допомогою скрипта. Значення «**Is Kinematic**» можна змінювати за допомогою коду, щоб ввімкнути або вимкнути фізику для об'єкта, але ця можливість вимагає додаткових ресурсів.

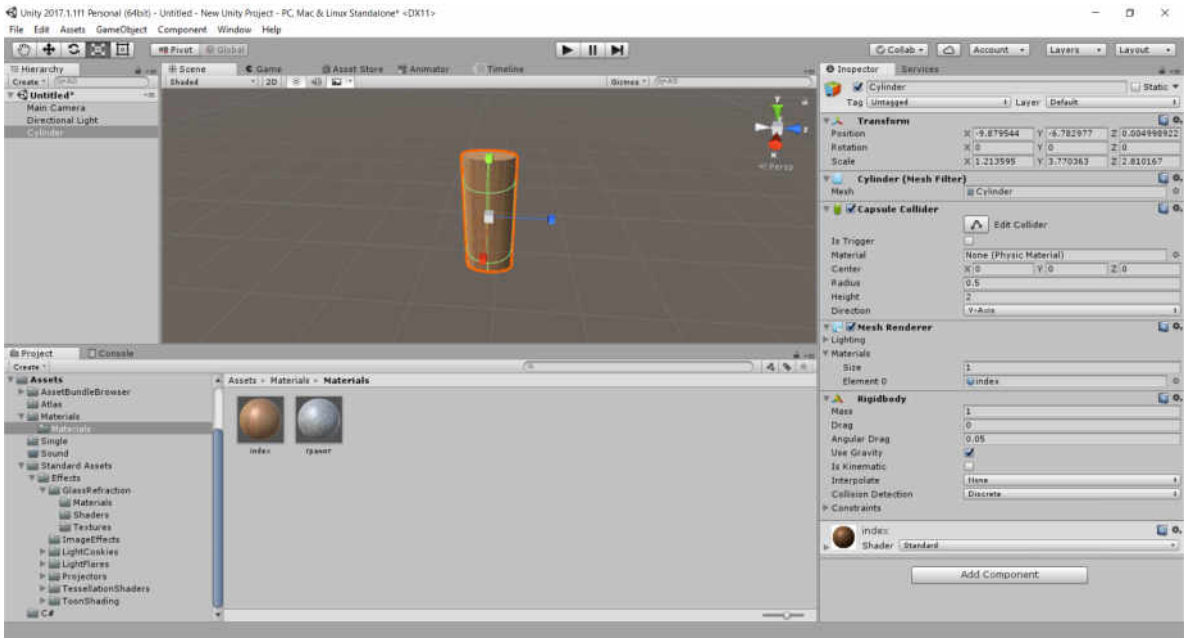

Рисунок 2.1 – Властивості компоненту **Rigidbody**

Kinematic Rigidbodies не схильні до впливу сил, гравітації і зіткнень. Ними керують, встановлюючи їх положення і кут повороту або анімуючи їх, але все ж вони можуть взаємодіяти з іншими некінематичними Rigidbodies (простий приклад: анімація бігу - це кінематика, а Ragdoll - це не кінематика, а фізика.)

Kinematic Rigidbodies використовуються в наступних випадках:

1. Іноді потрібно, щоб об'єкт знаходився під контролем фізики, але в деяких ситуаціях їм потрібно управляти явно за допомогою скрипта або за допомогою анімації. Наприклад, ви могли зробити анімований персонаж, до скелету якого застосовано Rigidbodies, і який в свою чергу пов'язано з вузлами (joints) для використання в якості Ragdoll. Більшість часу персонаж знаходиться під контролем анімації, таким чином застосовується Kinematic Rigidbody. Але коли персонаж убито, потрібно, щоб він перетворився в Ragdoll і був підпорядкований фізиці. Щоб досягти цього, необхідно просто відключити властивість **isKinematic**.

2. Kinematic Rigidbodies краще взаємодіють з іншими Rigidbodies. Наприклад, якщо у вас є анімована платформа, і ви хочете помістити деякі

бокси Rigidbody зверху, ви повинні зробити платформу **Kinematic Rigidbody** замість того, щоб зробити тільки колайдери без **Rigidbody**.

Colliders (Колайдери)

Колайдери це наступний тип компонентів, які повинні бути додані до твердих тіл, щоб задіяти зіткнення. Якщо два твердих тіла врізаються один в одного, фізичний движок не буде прораховувати зіткнення, поки до обох об'єктів не буде додано колайдер. Тверді тіла, які не мають колайдерів будуть просто проходити крізь один одного при прорахунку зіткнень.

Види колайдерів в Unity 3D

Box Collider базовий кубічний примітив зіткнень.

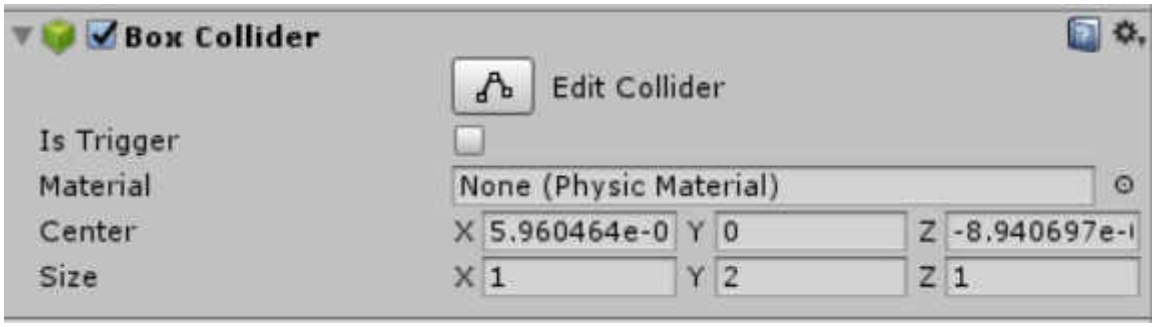

Рисунок 2.2 – **Box Collider**

Властивості компоненту **Box Collider** подано у табл. 2.1.

Таблиця 2.1 - Властивості компоненту **Box Collider**

Властивість	Функція
Is Trigger	Колайдер використовується для запуску подій, і ігнорується фізичним движком.
Material	Визначає, як цей колайдер взаємодіє з іншими.
Center	Позиція колайдера.
Size	Розміри колайдера в напрямках X, Y, Z.

Додати компонент **Box Collider** до об'єкта можна на панелі «**Inspector**» за допомогою кнопки «**Add Component→Physics→ Box Collider».** Доступ до властивостей компоненту **Box Collider** також буде відкрито на панелі «**Inspector**».

Capsule Collider

Capsule Collider складається з двох півсфер, з'єднаних між собою циліндром. Це така ж форма, як і у примітиву капсула (Capsule).

Властивості компоненту **Box Collider** подано у табл. 2.2.

Таблиця 2.2 - Властивості компоненту **Capsule Collider**

Властивість	Функція
Is Trigger	Колайдер використовується для запуску подій, і ігнорується фізичним движком.
Material	Визначає, як цей колайдер взаємодіє з іншими.
Center	Позиція колайдера у локальному просторі об'єкту.
Radius	Радіус ширини колайдера в локальному просторі.
Height	Загальна висота колайдера.
Direction	Ось, уздовж якої розташований колайдер в локальному просторі об'єкта.

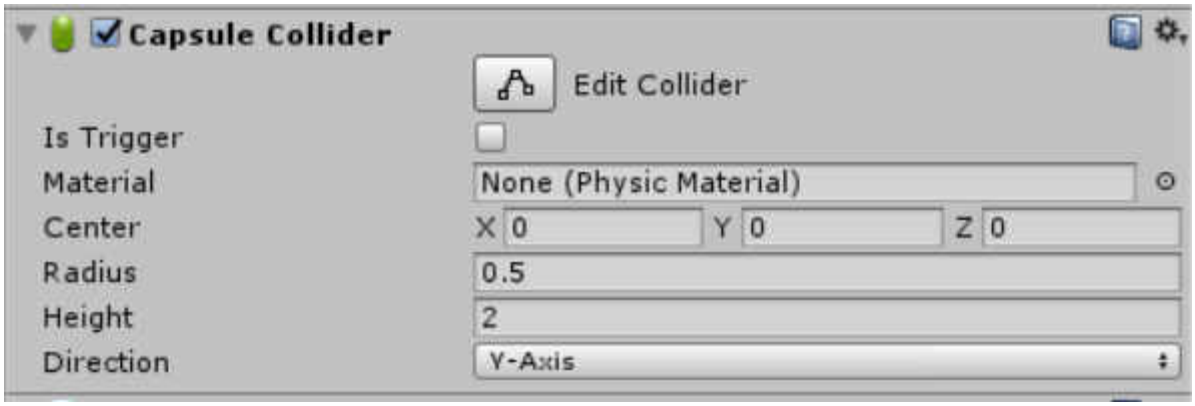

Рисунок 2.3 – Властивості **Capsule Collider**

Ви можете налаштувати радіус (Radius) і висоту (Height) колайдера незалежно один від одного. Ця можливість використовується в компоненті Character Controller і непогано працює для довгих предметів, а також надає можливість комбінувати колайдери для незвичайних форм.

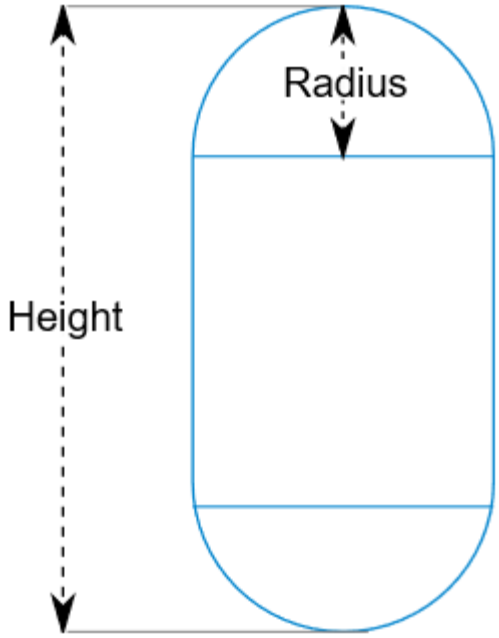

Рисунок 2.4 – **Capsule Collider**

Одним з параметрів компонента **Collider** є **Physics Material**, який за замовчуванням має значення **None** (Physics Material).

Для того, щоб створити новий фізичний **Material** скористайтеся меню «**Assets→Create→Physic Material**».

Властивості **Physics Material** подано у табл. 2.3.

Таблиця 2.3 - Властивості **Physics Material**

Властивість	Функція
Dynamic Friction	Тертя, що використовується під час руху. Зазвичай використовуються значення в діапазоні від 0 до 1. Значення, яке дорівнює 1 відповідає тертю як на льоду, в той час як значення, яке дорівнює 0 означає слабке тертя, в результаті якого об'єкту буде складно рухатися без впливу зовнішніх сил.
Static Friction	Тертя, що використовується коли об'єкт лежить на поверхні. Зазвичай використовуються значення в діапазоні від 0 до 1. Значення, яке дорівнює 0 означає відсутність тертя, в той час як значення, яке дорівнює 1 позначатиме абсолютне тертя (тобто об'єктам по такій поверхні буде складно пересуватися).
Bounciness	Вказує наскільки пружною є поверхня. Значення, яке дорівнює 0 означає непружну поверхню. Значення, яке

	дорівнює 1 призведе до пружності, при якій об'єкт не буде втрачати початкову енергію.
Friction Combine	Вказує на те, як комбінується між собою тертя двох об'єктів. - Average Середнє значення - Minimum З двох значень використовується те, що менше. - Maximum З двох значень використовується те, що більше. - Multiply Значення множаться один на одне.
Bounce Combine	Вказує на те, як комбінується пружність двох об'єктів. Bounce Combine підтримує ті ж самі режими, що і Friction Combine режим.

Властивості **Physics Material** подано на рис. 2.5.

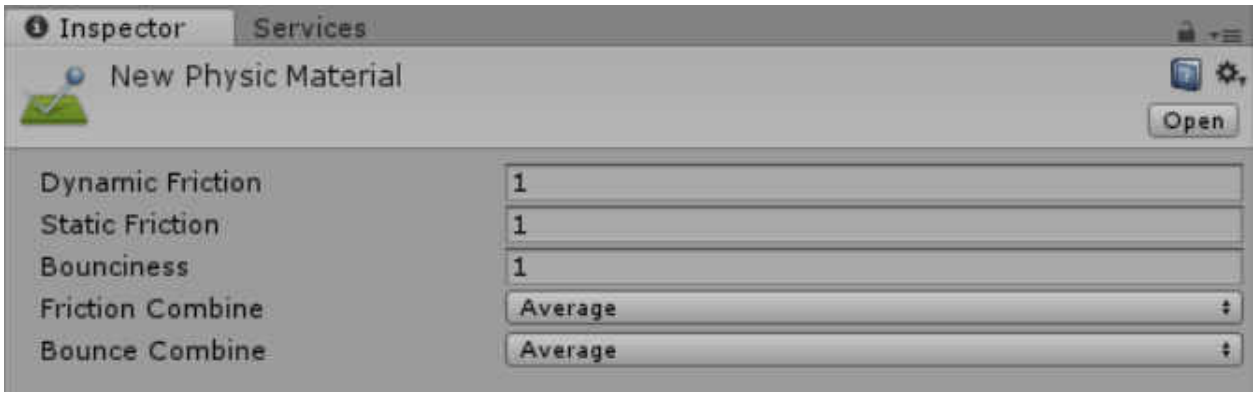

Рисунок 2.5 – Властивості **Physics Material**

Sphere Collider

Sphere Collider - базовий колайдер в формі сфери.

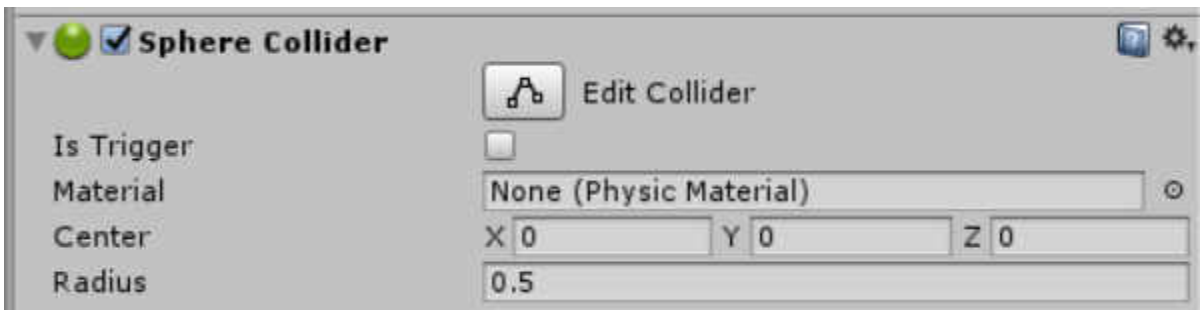

Рисунок 2.6 – Властивості **Physics Material**

Властивості компоненту **Sphere Collider** подано у табл. 2.4.

Таблиця 2.4 - Властивості компоненту **Sphere Collider**

Властивість	Функція
Is Trigger	Колайдер використовується для запуску подій, і ігнорується фізичним движком.
Material	Визначає, як цей колайдер взаємодіє з іншими.
Center	Позиція колайдера у локальному просторі об'єкту.
Radius	Розмір колайдера.

Взаємодія колайдерів

Колайдери взаємодіють один з одним по різному, в залежності від того, як налаштовані їх компоненти Rigidbody. Трьома важливими конфігураціями є статичний колайдер (**Static Collider**) (тобто компонент Rigidbody відсутній взагалі), Rigidbody колайдер (**Rigidbody Collider**), і кінематичний Rigidbody колайдер (**Kinematic Rigidbody Collider**).

Статичний колайдер (Static Collider)

Це ігровий об'єкт, у якого є колайдер, але немає Rigidbody. Статичні колайдери використовуються для геометрії рівнів, яка завжди стоїть на місці і зовсім не рухається. Зустрічні Rigidbody об'єкти будуть врізатися в статичний колайдер, але його не зрушать.

В фізичний движок закладено припущення, що статичні колайдери ніколи не рухаються і не змінюються, і, на основі цього припущення, движок робить корисні оптимізації. Отже, статичні колайдери не можна вмикати/вимикати, рухати або масштабувати під час ігрового процесу. Якщо ви змініте статичний колайдер, то в результаті фізичним движком буде викликаний додатковий внутрішній перерахунок, який буде супроводжуватися падінням продуктивності. Гірше того, зміни іноді можуть залишити колайдер в невизначеному стані, в результаті чого будуть проводитися помилкові фізичні розрахунки. З цих причин, слід змінювати тільки колайдери з Rigidbody.

Rigidbody колайдер (Rigidbody Collider)

Це ігровий об'єкт, до якого прикріплений колайдер і некінематичний Rigidbody. Rigidbody колайдери повністю симулюються фізичним движком і можуть реагувати на колізії і сили, прикладені з скрипта. Вони можуть стикатися з іншими об'єктами (включаючи статичні колайдери) і є найпоширенішою конфігурацією колайдера в іграх, які використовують фізику.

Кінематичні Rigidbody колайдери (Kinematic Rigidbody Collider)

Це ігровий об'єкт, до якого прикріплений колайдер і кінематичний Rigidbody (тобто властивість IsKinematic компонента Rigidbody включено).

Змінюючи компонент **Transform**, ви можете переміщати об'єкт з кінематичним Rigidbody, але він не буде реагувати на колізії та додані сили так само, як і некінематичні Rigidbody. Кінематичні Rigidbody повинні використовуватися для колайдерів, які можуть рухатися або періодично вимикатися/вмикатися, інакше вони будуть вести себе як статичні колайдери.

На відміну від статичного колайдера, кінематичний Rigidbody буде застосовувати тертя до інших об'єктів і, в разі контакту, буде «будити» інші Rigidbody.

Навіть коли вони нерухомі, кінематичні Rigidbody колайдери поводяться інакше, на відміну від статичних колайдерів. Наприклад, якщо колайдер налаштований як тригер, то вам також знадобиться додати до нього Rigidbody, щоб можна було в вашому скрипті приймати події тригера. Якщо ви не хочете, щоб тригер падав під дією сили гравітації або піддавався впливу фізики, то тоді ви можете включити властивість IsKinematic.

Triggers (Тригери)

Система сценаріїв може виявляти, коли відбуваються зіткнення, і ініціювати дії за допомогою функції **OnCollisionEnter**. Однак ви також можете використовувати фізичний движок просто для виявлення того, коли один колайдер входить в простір іншого, не створюючи зіткнення. Колайдер, який конфігуровано як тригер (з використанням властивості Is Trigger) просто дозволяє іншим колайдерам проходити крізь себе. Коли колайдер входить в простір іншого, тригер буде викликати функцію **OnTriggerEnter** в скрипті об'єкта тригера.

Матриця дій колізії

Коли стикаються 2 об'єкти, кількість різних подій в скрипті залежить від конфігурацій компонентів Rigidbody об'єктів, які стикаються. Схеми подані нижче містять деталі того, які функції подій будуть викликані, ґрунтуючись на компонентах, які приєднано до об'єктів. У деяких комбінаціях ефект розповсюджується тільки на один з двох об'єктів, так що пам'ятаєте правило - закони фізики не застосовуються до об'єктів, у яких немає приєднаного Rigidbody.

Таблиця 2.5 - Відбувається визначення зіткнень, і при їх виникненні надсилаються повідомлення

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Так				
Rigidbody Collider	Так	Так	Так			
Kinematic Rigidbody Collider		Так				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

Таблиця 2.6 - При колізіях відсилаються повідомлення тригера

	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Так	Так
Rigidbody Collider				Так	Так	Так
Kinematic Rigidbody Collider				Так	Так	Так
Static Trigger Collider		Так	Так		Так	Так
Rigidbody Trigger Collider	Так	Так	Так	Так	Так	Так
Kinematic Rigidbody Trigger Collider	Так	Так	Так	Так	Так	Так

Character Controllers (Контролери персонажа)

Компонент **Character Controller** використовується для управління від третьої або першої особи, де не потрібна фізика **Rigidbody**.

Цей компонент дає персонажу простий колайдер в формі капсули, який завжди знаходиться у вертикальному положенні.

У компонента Character Controller є свої особливі функції для призначення швидкості і напряму об'єкта (табл.2.7).

Таблиця 2.7 - Властивості **Character Controllers**

Властивість	Функція
Slope Limit	Обмежує можливість колайдера підійматися по схилах, значення яких дорівнює або менше ніж вказане в Slope Limit.
Step Offset	Персонаж ступить на поверхню, тільки якщо вона ближче до землі, ніж задане в Step Offset значення.
Skin width	Два колайдери можуть перетнутися один з одним на глибину, яка дорівнює значенню Skin Width. Найкращим варіантом є встановлення цього значення рівним 10% від радіуса.
Min Move Distance	Якщо персонаж спробує зрушити нижче зазначеної величини, то він не зрушить. У більшості ситуацій це значення варто залишити рівним нулю (0).
Center	Зрушення колайдера в світовому просторі без впливу на те, як обертається персонаж.
Radius	Значення радіуса колайдера.
Height	Висота колайдера Capsule Collider персонажа. Зміна значення Height розтягне колайдер уздовж осі X в обидва напрямки.

Додати компонент **Character Controller** до об'єкта можна на панелі «**Inspector**» за допомогою меню «**Add Component→Physics→Character Controller**».

Доступ до властивостей компоненту **Character Controller** також буде відкрито на панелі «**Inspector**».

Для того, щоб переміщати 3D-об'єкт, який володіє компонентом **Characler Controller**, необхідно створити C# -скрипт.

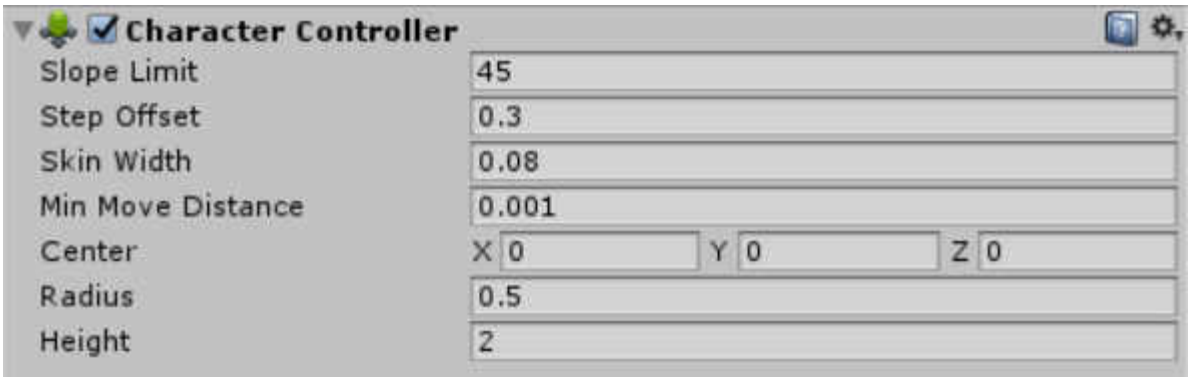

Рисунок 2.7 – Компонент **Character Controller**

Fixed Joints (Нерухоме з'єднання)

Компонент **Fixed Joints** обмежує рух певного об'єкта, пов'язуючи його з іншим об'єктом.

Найчастіше цей компонент використовується, якщо в певний момент часу може знадобитися роз'єднати два об'єкти, або навпаки, з'єднати два об'єкти без необхідності зміни ієрархії.

Властивості **Fixed Joints** подано у табл. 2.8.

Таблиця 2.8 - Властивості **Fixed Joints**

Властивість	Функція
Connected Body	Необов'язкове посилання на інший об'єкт з Rigidbody, до якого приєднується поточний об'єкт. Якщо поле залишити порожнім, об'єкт приєднується до заданої точки в просторі.
Break Force	Сила, яку потрібно прикласти до об'єкта, щоб розірвати з'єднання.
Break Torque	Крутний момент, який необхідно прикласти до об'єкта, щоб розірвати з'єднання.
Enable Collision	Якщо включено Enable Collision, то два з'єднаних об'єкта будуть стикатися один з одним.
Enable Preprocessing	Вимкнення попередньої обробки допомагає стабілізувати неможливі конфігурації.

При створенні ігор іноді виникають випадки, коли потрібно, щоб об'єкти рухались разом (тимчасово або постійно). Компоненти **Fixed Joints** дозволяють спростити реалізацію подібних ситуацій, оскільки вам не доводиться змінювати положення об'єкта в ієрархії за допомогою скриптів.

Минус такого рішення в тому, що вам доведеться додати компоненти **Rigidbodies** на об'єкти, які потрібно з'єднати за допомогою **Fixed Joints.**

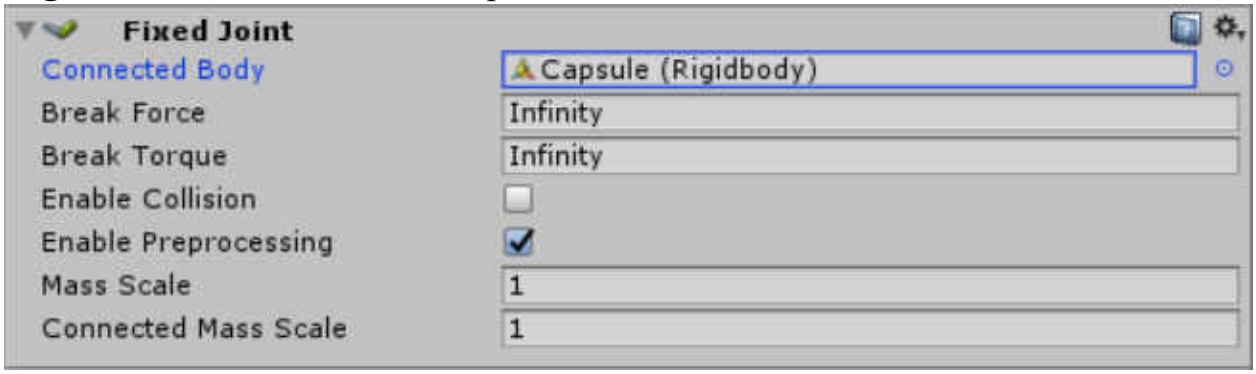

Рисунок 2.8 – Компонент **Fixed Joint**

Spring Joint (Пружне з'єднання)

Spring Joint з'єднує пружиною два Rigidbody-об'єкта. Властивості **Spring Joint** подано у табл. 2.9.

Таблиця 2.9 - Властивості **Spring Joint**

Властивість	Функція
Connected Body	Об'єкт з пружинним з'єднанням з'єднаний з іншим об'єктом. Якщо об'єкту не призначено Connected Body, то пружина буде з'єднана з фіксованою точкою в просторі.
Anchor	Точка в локальному просторі об'єкта, на якій знаходиться Joint.
Auto Configure Connected Anchor	Unity автоматично обчислює позицію підключеної опорної точки.
Connected Anchor	Точка в локальному просторі з'єднаного об'єкту, на якій знаходиться Joint.
Spring	Сила пружини.
Damper	Значення, при якому пружину скорочено, якщо вона є активною.
Min Distance	Нижня межа діапазону відстані, над якою пружина не буде застосовувати будь-яку силу.
Max Distance	Верхня межа діапазону відстані, над якою пружина не буде застосовувати будь-яку силу.
Tolerance	Змінює толерантність до помилок.
Break Force	Сила, яку слід прикласти до Joint, щоб розбити його.
Break Torque	Крутний момент, який треба прикласти до Joint, щоб розбити його.

Enable Collision	Виявлення колізій (зіткнень).
Enable Preprocessing	Вимкнення попередньої обробки допомагає стабілізувати неможливі конфігурації.

Контрольні запитання

- Назвіть базові компоненти системи Unity 3D для роботи з фізикою 3D - об'єктів.
- Назвіть основні властивості компоненту Rigidbodies.
- Назвіть основні властивості компоненту Colliders.
- Для чого в системи Unity 3D використовуються Joint – компоненти?
- Назвіть опції Joint - компонента, які можна включити для різних ефектів.
- Для чого в системи Unity 3D використовуються контролери персонажа (Character Controllers)?
- Назвіть основні властивості компоненту Character Controllers.
- Які можливості надають розробникам ігрового контенту компоненти Box Collider, Capsule Collider, Character Joints, Configurable Joint, Fixed Joints, Sphere Collider, Spring Joint?

ЛЕКЦІЯ №3
ТЕМА: «РОЗРОБКА ТА ВИКОРИСТАННЯ СКРИПТІВ В UNITY 3D. СЕРЕДОВИЩЕ РОЗРОБКИ «MONODEVELOP». СИСТЕМА ПОДІЙ «EVENTSYSTEM»

Анотація

Лекція знайомить з технологією розробки скриптів в системі Unity 3D, методами створення і знищення ігрових об'єктів, управління ігровими об'єктами за допомогою відповідних компонентів, методами роботи в інтегрованому середовищі розробки MonoDevelop, яке поєднує в собі функції текстового редактора з додатковими можливостями для налагодження і виконання завдань з управління ігровими проектами, системою подій «EventSystem» Unity 3D.

Мета лекції

Ознайомити студентів з методами створення, управління і знищення ігрових об'єктів, методами роботи в інтегрованому середовищі розробки MonoDevelop, способами відправки подій до об'єктів в ігровому додатку – системою подій «EventSystem».

Очікувані результати

Сформувати у студентів знання щодо основних методів створення і знищення ігрових об'єктів в системі Unity 3D, основних компонентів, які використовуються для управління ігровими об'єктами, основними принципами роботи в інтегрованому середовищі розробки MonoDevelop, основних методів роботи з системою подій «EventSystem».

Створення та використання скриптів в Unity 3D

Поведінка ігрових об'єктів контролюється за допомогою компонентів (Components), які приєднуються до них. Незважаючи на те, що вбудовані компоненти Unity можуть бути дуже різнобічними, незабаром ви виявите, що вам потрібно вийти за межі їх можливостей, щоб реалізувати ваші власні особливості геймплею. Unity дозволяє вам створювати свої компоненти, використовуючи скрипти. Вони дозволяють активувати ігрові події, змінювати параметри компонентів, і відповідати на дії користувача будь-яким способом.

Unity підтримує такі мови програмування:

1. C # - мова подібна Java або C ++;

2. UnityScript - мова, яку розроблено спеціально для використання в Unity за зразком JavaScript.

Створення скриптів

На відміну від інших **Asset**, скрипти зазвичай створюються безпосередньо в Unity. Ви можете створити скрипт, використовуючи меню **Create** в лівому верхньому кутку панелі **Project** або вибравши **Assets→ Create→ C# Script (або JavaScript)** в головному меню.

Новий скрипт буде створено в папці, яку ви обрали в панелі **Project**.

Структура файлу скрипта

Після подвійного натискання на файлі скрипта, його буде відкрито в текстовому редакторі. За замовчуванням Unity буде використовувати **MonoDevelop**, але ви можете вибрати будь-який редактор з панелі **External Tools** в налаштуваннях Unity.

Вміст файлу буде виглядати так:

```
using UnityEngine;

using System.Collections;

public class MainPlayer : MonoBehaviour {

   // Use this for initialization

   void Start () {

   }

   // Update is called once per frame

   void Update () {

   }

}
```

Скрипт взаємодіє з внутрішніми механізмами Unity за рахунок створення класу, успадкованого від вбудованого класу **MonoBehaviour**.

Кожен раз, коли ви приєднуєте скриптовий компонент до ігрового об'єкту, створюється новий екземпляр об'єкта. Ім'я класу і ім'я файлу повинні бути однаковими, для того, щоб скриптовий компонент міг бути приєднаний до ігрового об'єкту.

Функція **Update** - це місце для розміщення коду, який буде обробляти оновлення кадру для ігрового об'єкта. Це може бути рух, реакція на дії користувача та все, що повинно бути оброблено у ігровому процесі. Щоб дозволити функції **Update** виконувати свою роботу, необхідно ініціалізувати змінні і здійснити зв'язок з іншими ігровими об'єктами до того, як будуть здійснені будь-які дії.

Функцію **Start** буде викликано Unity до початку ігрового процесу (тобто до першого виклику функції Update), і це ідеальне місце для виконання ініціалізації.

Ініціалізація об'єкта в Unity виконується не в функції-конструкторі тому, що створення об'єктів обробляється редактором і відбувається не на початку ігрового процесу. Якщо ви спробуєте визначити конструктор для скриптового компонента, він буде заважати нормальній роботі Unity і може викликати серйозні проблеми з проектом.

Управління ігровим об'єктом

Код скрипта не буде активовано до тих пір, поки екземпляр скрипта не буде приєднано до ігрового об'єкту. Ви можете прикріпити скрипт додавши **Asset** скрипта на ігровий об'єкт в панелі **Hierarchy** або через вікно **Inspector** обраного ігрового об'єкта. Існує також підменю **Scripts** в меню **Component**, яке містить всі скрипти, доступні в проекті, включаючи ті, які ви створили самі.

Примірник скрипта виглядає так само, як і інші компоненти у вікні **Inspector**.

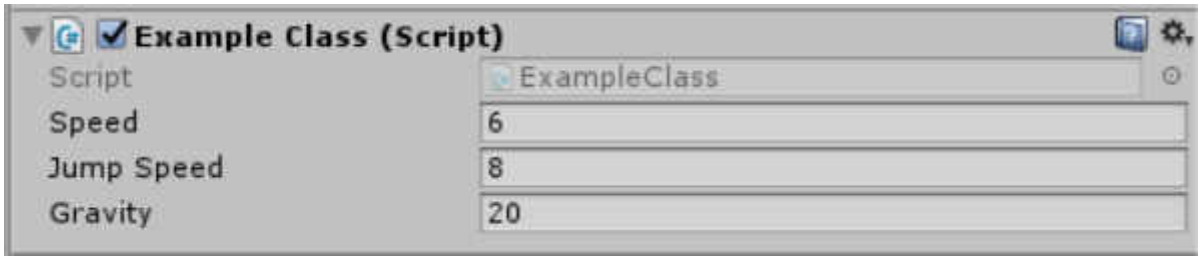

Рисунок 3.1 – Скрипт на панелі Inspector

Після приєднання скрипт почне працювати, коли ви натиснете **Play** і запустіть гру. Ви можете перевірити це додавши наступний код в функцію **Start**.

```
// Use this for initialization
void Start () {
```

```
    Debug.Log("I am alive!");
}
```

Debug.Log це команда, яка просто виводить повідомлення на консоль **Unity**. Якщо ви натиснете **Play**, то побачите повідомлення в вікні **Console** (меню **Window→ Console**).

Під час створення сценарію ви, власне кажучи, створюєте свій власний новий тип компонента, який можна приєднати до ігрових об'єктів, як і будь-який інший компонент.

Подібно до того, як інші компоненти мають властивості, які можна редагувати в інспекторі, ви можете дозволити редагувати значення у вашому сценарії також за допомогою інспектора.

```
using UnityEngine;
using System.Collections;
public class MainPlayer : MonoBehaviour {
    public string myName;
    // Use this for initialization
    void Start () {
        Debug.Log("I am alive and my name is " + myName);
    }
    // Update is called once per frame
    void Update () {
    }
}
```

Цей код створює поле для редагування в **Inspector** з назвою «**MyName**».

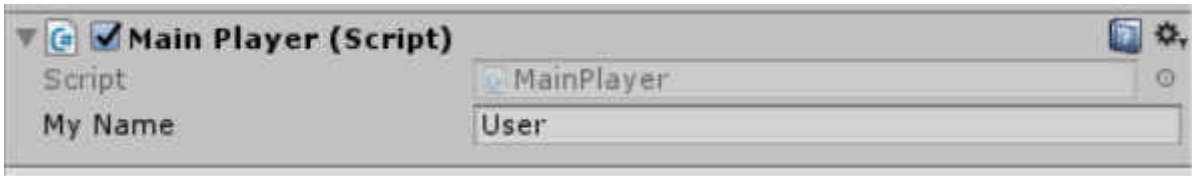

Рисунок 3.2 – Редагування змінної за допомогою панелі Inspector

Unity створює мітку в вікні **Inspector** шляхом додавання пробілу всюди, де в імені змінної зустрічається велика літера. Однак це виключно в цілях відображення, і ви повинні завжди в коді використовувати ім'я змінної. Якщо ви відредагуєте значення змінної і потім натиснете **Play**, то побачите, що повідомлення містить введений вами текст.

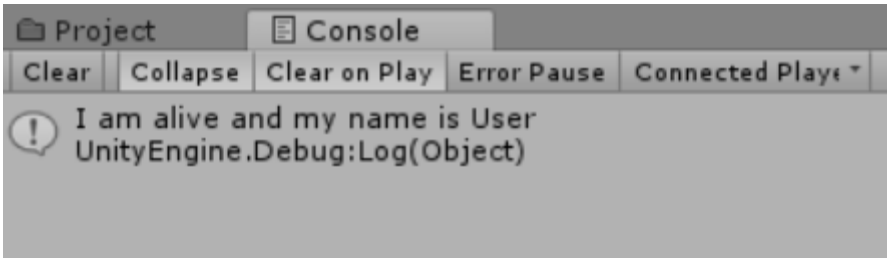

Рисунок 3.3 – Повідомлення в вікні **Console**

Unity дозволяє змінювати значення змінних скрипта в запущеній грі. Це дуже корисно і надає можливість подивіться всі ефекти від змін відразу ж, без зупинки і перезапуску. Коли програвання закінчиться, значення змінних скидаються в той стан, який вони мали до натискання кнопки **Play**. Це гарантує, що ви вільно можете грати з настройками об'єктів, не боячись щось зіпсувати**.**

Управління ігровими об'єктами (GameObjects) за допомогою компонентів

У редакторі Unity можна змінювати властивості компонента використовуючи вікно **Inspector**. Так, наприклад, зміна позиції компонента **Transform** призведе до зміни позиції ігрового об'єкта. Аналогічно, ви можете змінити колір матеріалу компонента **Renderer** або масу твердого тіла **RigidBody** з відповідним впливом на відображення або поведінку ігрового об'єкта. Здебільшого скрипти також змінюють властивості компонентів для управління ігровими об'єктами. Різниця, однак, в тому, що скрипт може змінювати значення властивості поступово з часом. За рахунок зміни, створення і знищення об'єктів в заданий час може бути реалізований будь-який ігровий процес.

Звернення до компонентів

Найбільш простим і поширеним є випадок, коли скрипту необхідно звернутися до інших компонентів, приєднаних до того ж **GameObject**. Як вже згадувалося, компонент насправді є екземпляром класу, так що першим кроком буде отримання посилання на екземпляр компонента, з яким ви хочете працювати. Це здійснюється за допомогою функції **GetComponent**. Зазвичай

об'єкт компонента зберігають в змінну, це здійснюється в C# за допомогою наступного синтаксису:

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

Як тільки у вас є посилання на екземпляр компонента, ви можете встановлювати значення його властивостей.

```
void Start () {
   Rigidbody rb = GetComponent<Rigidbody>();
   // Change the mass of the object's Rigidbody.
   rb.mass = 10f;
}
```

Додаткова можливість, що недоступна у вікні Inspector – це виклик функцій екземпляра компонента:

```
void Start () {
   Rigidbody rb = GetComponent<Rigidbody>();
   // Add a force to the Rigidbody.
   rb.AddForce(Vector3.up * 10f);
}
```

Звернення до інших об'єктів

Зазвичай, скрипти відстежують інші об'єкти. Unity надає кілька шляхів отримання інших об'єктів, кожен підходить для конкретної ситуації.

Зв'язування об'єктів за допомогою змінних

Найпростіший спосіб знайти потрібний ігровий об'єкт - додати в скрипт змінну типу **GameObject** з рівнем доступу **public**:

```
public class Enemy : MonoBehaviour {

    public GameObject player;

    // Other variables and functions...

}
```

Змінну буде видно у вікні Inspector, як і будь-які інші:

Рисунок 3.4 – Зв'язування об'єктів за допомогою змінних

Тепер ви можете перетягнути об'єкт зі сцени або з панелі **Hierarchy** в цю змінну. Функція **GetComponent** і доступ до змінних компонента доступні як для цього об'єкта, так і для інших, тобто ви можете використовувати наступний код:

```
public class Enemy : MonoBehaviour {

    public GameObject player;

    void Start() {

        // Start the enemy ten units behind the player character.

        transform.position = player.transform.position - Vector3.forward *
10f;

    }

}
```

Крім того, якщо оголосити змінну з доступом **public** і заданим типом компонента в вашому скрипті, ви зможете перетягнути будь-який об'єкт, який містить приєднаний компонент такого типу. Це дозволить звертатися до компоненту безпосередньо, а не через ігровий об'єкт.

```
public Transform playerTransform;
```

З'єднання об'єктів через змінні найбільш корисно, коли ви маєте справу з окремими об'єктами, що мають постійний зв'язок. Ви можете використовувати масив для зберігання зв'язку з декількома об'єктами одного типу, але вони все одно повинні бути задані в редакторі Unity, а не під час виконання. Але часто необхідно знаходити об'єкти під час виконання, і Unity надає два основних способи зробити це.

Пошук дочірніх об'єктів

Іноді ігрова сцена може використовувати кілька об'єктів одного типу, таких як вороги, шляхові точки і перешкоди. Може виникнути необхідність відстеження їх в певному скрипті, який управляє або реагує на них (наприклад, всі шляхові точки можуть знадобитися для скрипта пошуку шляху). Можна використовувати змінні для зв'язування цих об'єктів, але це зробить процес проектування утомливим, якщо кожну нову точку маршруту потрібно буде перетягнути в змінну в скрипті. Аналогічно, при видаленні точки маршруту доведеться видаляти посилання на відсутній об'єкт. У випадках, на зразок цього, найчастіше зручно управляти набором об'єктів, зробивши їх дочірніми одного батьківського об'єкта. Дочірні об'єкти можуть бути отримані, використовуючи компонент **Transform** батька (так як всі ігрові об'єкти неявно містять **Transform**):

```
using UnityEngine;

public class WaypointManager : MonoBehaviour {

    public Transform[] waypoints;

    void Start() {

        waypoints = new Transform[transform.childCount];

        int i = 0;

        foreach (Transform t in transform) {

            waypoints[i++] = t;

        }

    }

}
```

Ви можете також знайти заданий дочірній об'єкт по імені, використовуючи функцію **Transform.Find**:

```
transform.Find("Gun");
```

Це може бути корисно, коли об'єкт містить дочірній елемент, який може бути доданий або вилучений в ігровому процесі. Хороший приклад - зброя, яку можна підібрати і викинути.

Пошук об'єктів за допомогою імені або тегу.

Окремі об'єкти можуть бути отримані за допомогою ім'я, використовуючи функцію **GameObject.Find:**

```
ameObject player;
void Start() {
   player = GameObject.Find("MainHeroCharacter");
}
```

Об'єкт або колекція об'єктів можуть бути також знайдені за допомогою тегу, використовуючи функції **GameObject.FindWithTag** і **GameObject.FindGameObjectsWithTag.**

```
GameObject player;
GameObject[] enemies;
void Start() {
   player = GameObject.FindWithTag("Player");
   enemies = GameObject.FindGameObjectsWithTag("Enemy");
}
```

Середовище розробки MonoDevelop

MonoDevelop - це інтегроване середовище розробки (IDE), що поставляється разом з **Unity**. IDE поєднує в собі функції текстового редактора

з додатковими можливостями для налагодження і виконання інших завдань з управління ігровими проектами.

MonoDevelop встановлюється за умовчанням разом з **Unity**. Під час установки **Unity**, ви можете скасувати установку **MonoDevelop**.

Переконайтеся, що **MonoDevelop** встановлений в якості зовнішнього редактора скриптів в Preferences за допомогою меню «**Edit→ Preferences»**, а потім виберіть вкладку **External Tools**. Якщо в опції **External Script Editor** вибрана опція **MonoDevelop (built-in)**, **Unity** запустить **IDE MonoDevelop** і буде використовувати його в якості редактора за замовчуванням для всіх скриптових файлів.

Слід зауважити, що в опції **External Script Editor** в якості редактора для всіх скриптових файлів може бути вибране середовище розробки **Visual Studio.**

Перш ніж почати налагодження коду в **MonoDevelop**, вам спершу слід перевірити, що в **Preferences**, на панелі **External Tools** включена опція **Editor Attaching** (рис.3.5).

Переконайтеся, що в **BuildSettings** цільової платформи (меню: **File→ Build Settings**) включені опції **Development Build** і **Script Debugging** (рис.3.6).

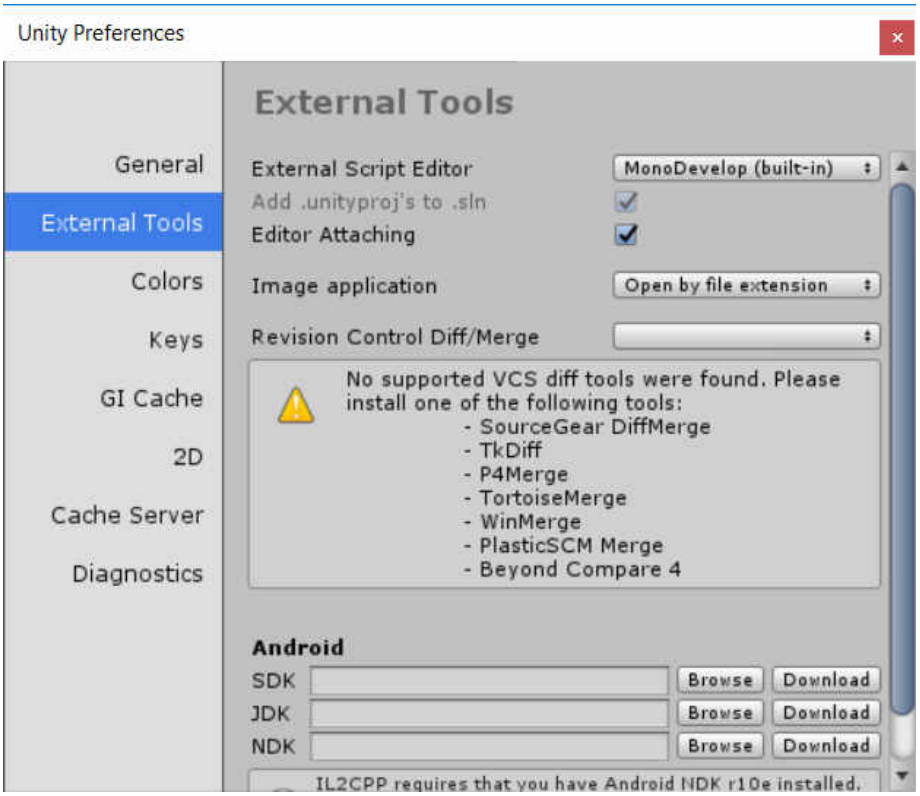

Рисунок 3.5 - Вкладка External Tools

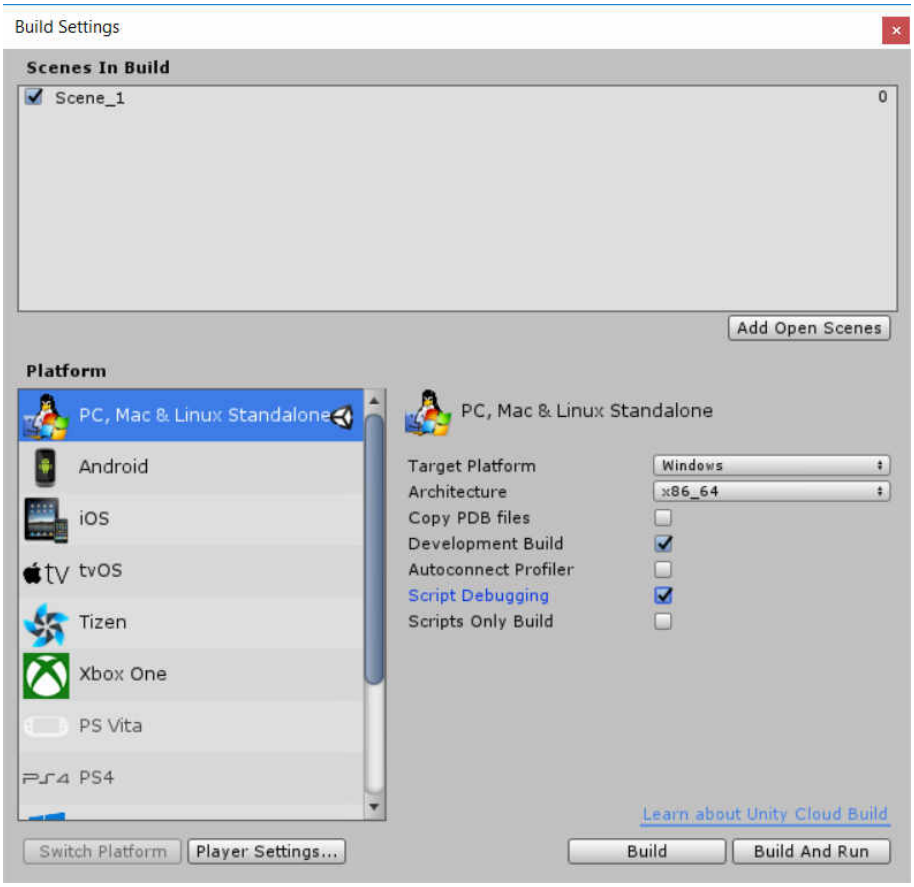

Рисунок 3.6 - Вікно BuildSettings

Створення проекту в MonoDevelop

Для створення нового проекту в меню «**File**» вибираємо пункт «**New→Workspace**». Відкриється майстер проекту (рис. 3.7).

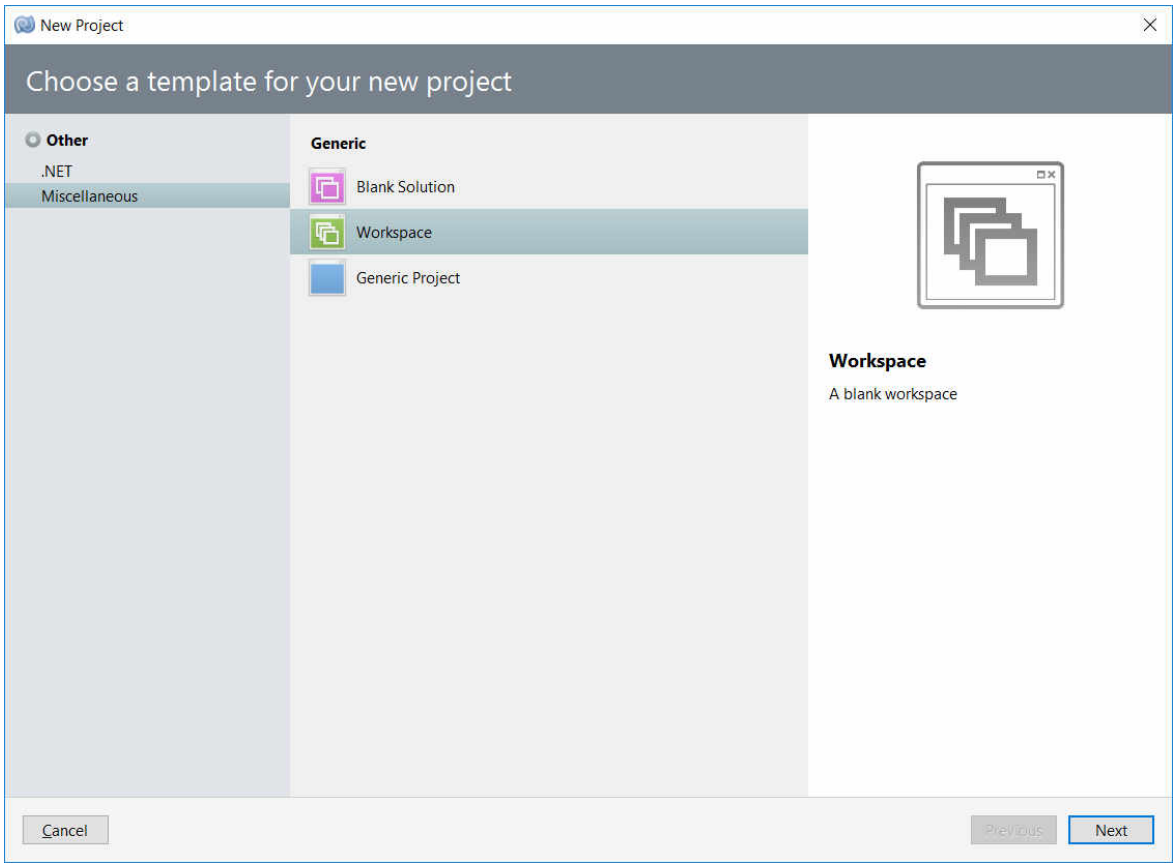

Рисунок 3.7 - Майстер проекту

У текстовому полі **Project Name** вводимо ім'я проекту, в текстове поле **Solution Name** - ім'я рішення (ці два пункти в даному випадку збігаються, а можуть і не збігатися), а в **Location** вказуємо місце розташування проекту.

Галочка під **Location** позначає створення внутрішньої папки з ім'ям рішення, куди будуть поміщені всі оригінали - рекомендується залишити цю позначку як є, але якщо хочеться іншого варіанту, то можете сміливо діяти.

Інші пункти призначені для управління контролем версій (рис. 3.8).

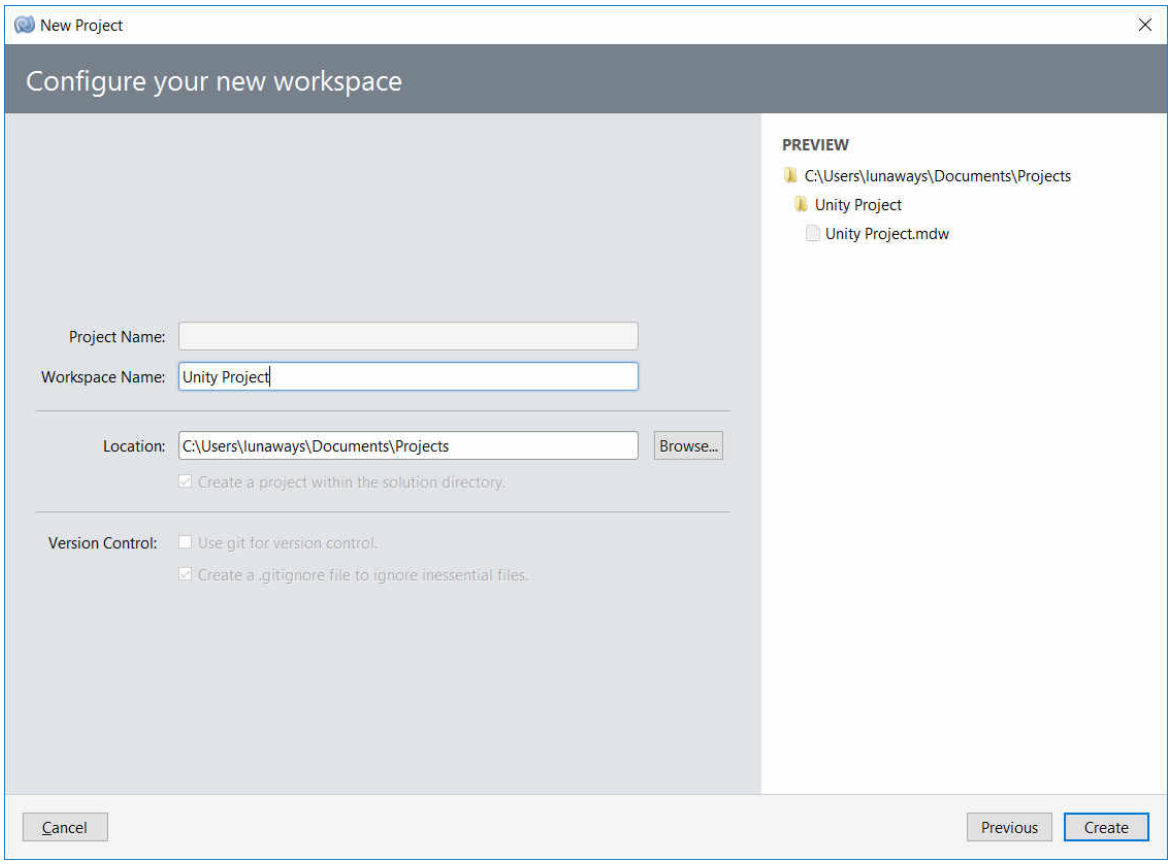

Рисунок 3.8 - Створення проекту

Натискаємо кнопку **Create**, після чого отримуємо порожню папку проекту (рис. 3.9).

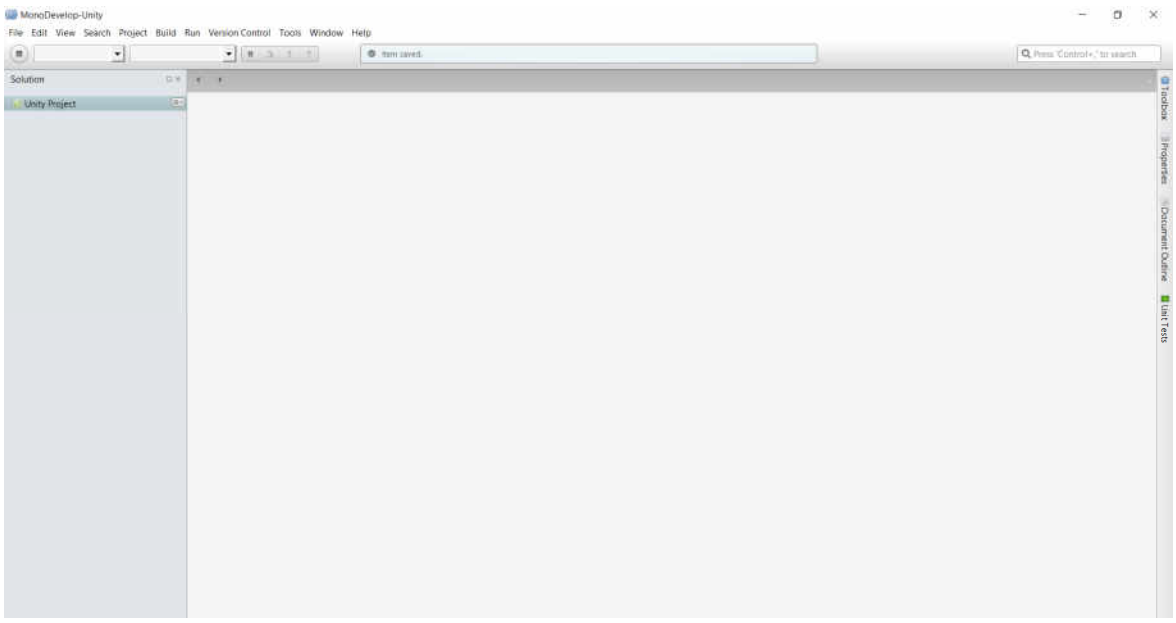

Рисунок 3.9 - Папка проекту

Створимо новий C# - файл за допомогою меню **File→ New→ File** (рис. 3.10).

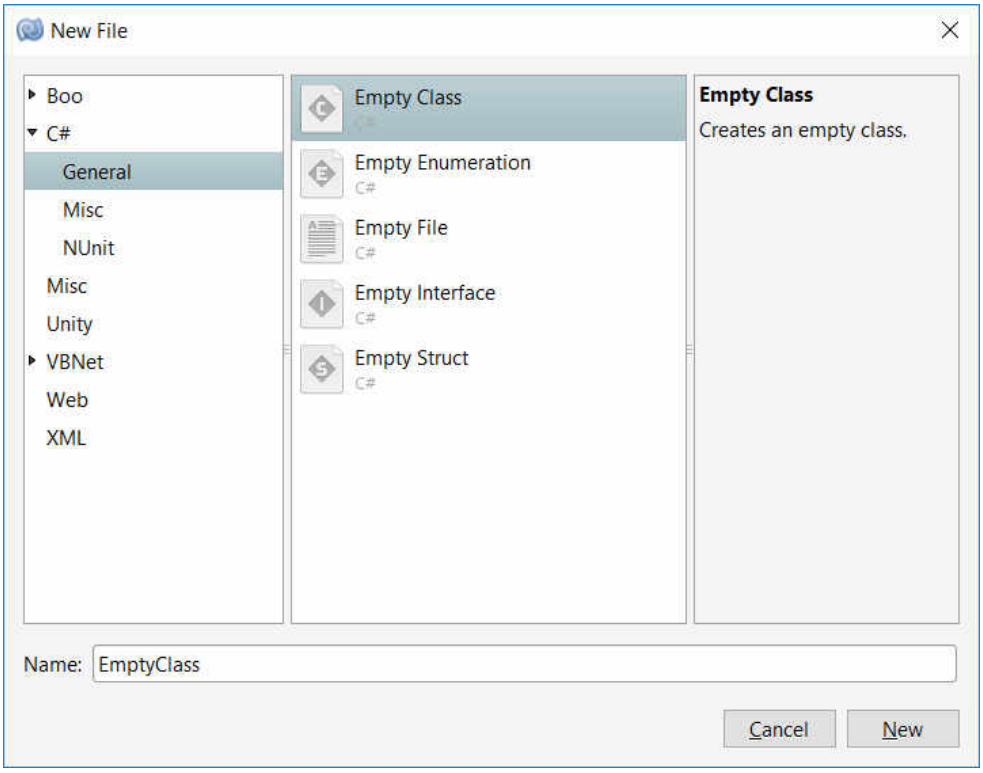

Рисунок 3.10 - Створення нового C# - файлу

Отримуємо новий файл, який є частиною проекту і який вже відкритий на редагування.

Система подій (EventSystem)

Система подій - спосіб відправки подій до об'єктів в додатку. Система складається з декількох компонентів, які працюють разом.

Додавши компонент **EventSystem** до ігрового об'єкту, ви помітите, що він не містить особливої функціональності, тому що призначений для управління і організації зв'язку між модулями компонента.

Первинні ролі системи подій:

1) визначає який об'єкт вважається виділеним;
2) визначає який InputModule використовується;
3) управляє райкастінгом (якщо потрібно), оновленням всіх модулів введення (InputModules), якщо потрібно.

Модулі введення

У модулях введення знаходиться основна логіка бажаної поведінки **EventSystem**, вони використовуються для:

1) обробки введення;
2) управління станами подій;
3) відправки подій об'єктам на сцені.

Тільки один модуль введення може бути активним в системі подій одночасно. Він повинен знаходитися на тому ж ігровому об'єкті, що і компонент **EventSystem**.

Якщо ви бажаєте написати власний модуль введення, рекомендується відправляти події, які підтримуються вже існуючими UI-компонентами Unity.

Трасувальники променів (Raycasters)

Трасувальники променів використовуються для визначення того над чим знаходиться покажчик пристрою введення.

За замовчуванням, надається 3 трасувальника променів:

1) **GraphicRaycaster** - використовується для UI-елементів;
2) **2DPhysicsRaycaster** - використовується для двовимірних фізичних елементів;
3) **3DPhysicsRaycaster** - використовується для тривимірних фізичних елементів.

Система обміну повідомленнями Messaging System

Нова система користувальницького інтерфейсу використовує систему обміну повідомленнями, призначену для заміни **SendMessage**. Система є чистою C# і спрямована на вирішення деяких проблем, що постають у **SendMessage**. Система працює з використанням користувацьких інтерфейсів, які можуть бути реалізовані на **MonoBehavior**, і вказує, що компонент здатний отримувати зворотний виклик із системи обміну повідомленнями.

Система обміну повідомленнями дозволяє передавати користувацькі дані, а також те, наскільки далеко через ієрархію GameObject поширюватиметься подія. Окрім цього, система обміну повідомленнями надає допоміжні функції для пошуку GameObjects, які реалізують певний інтерфейс обміну повідомленнями.

Система обміну повідомленнями є загальною і призначена для використання не тільки системою користувальницького інтерфейсу, але і загальним ігровим кодом. Досить тривіальним є додавання користувацьких

подій для обміну повідомленнями, і вони працюватимуть з тією ж схемою, яку використовує система інтерфейсу користувача для обробки всіх подій.

Створення Message

У просторі імен **UnityEngine.EventSystems** є базовий інтерфейс **IEventSystemHandler** за допомогою якого здійснюється **Message.**

```
public interface ICustomMessageTarget : IEventSystemHandler
{
   // functions that can be called via the messaging system
   void Message1();
   void Message2();
}
```

Після визначення цього інтерфейсу він може бути реалізований за допомогою **MonoBehaviour**. При реалізації він визначає функції, які будуть виконуватись, якщо це повідомлення буде випущено для **GameObject MonoBehaviours**.

```
public class CustomMessageTarget: MonoBehaviour, ICustomMessageTarget
{
   public void Message1()
   {
      Debug.Log ("Message 1 received");
   }
   public void Message2()
   {
      Debug.Log ("Message 2 received");
   }
}
```

Для того, щоб надіслати повідомлення, існує статичний клас **ExecuteEvents**. В якості аргументу для нього потрібен цільовий об'єкт, деякі конкретні дані користувача та функтор, який відображає певну функцію в інтерфейсі повідомлення:

ExecuteEvents.Execute <ICustomMessageTarget> (target, null, (x,y)=>x.Message1());

Контрольні запитання

- Які мови програмування підтримує система Unity 3D?
- Яким чином здійснюється управління ігровими об'єктами (GameObjects) в системі Unity 3D?
- Назвіть переваги використання інтегрованого середовища розробки MonoDevelop для налагодження і виконання завдань з управління ігровими проектами.
- Назвіть основні методи роботи з системою подій «EventSystem» в Unity 3D.

ЛЕКЦІЯ №4
ТЕМА: «АУДІО-КОМПОНЕНТИ UNITY 3D. ІМПОРТ І НАЛАШТУВАННЯ ЗВУКУ»

Анотація

Лекція знайомить з аудіо-компонентами Unity 3D, можливостями імпорту і відтворення звукових ефектів, методами активації звукових ефектів за допомогою програмного коду, методами роботи з AudioManager в Unity 3D.

Мета лекції

Ознайомити студентів з системою аудіо-компонентів Unity 3D. Розглянути можливості імпорту і відтворення звукових ефектів в Unity 3D, методи активації звукових ефектів за допомогою програмного коду, методи налаштування та роботи з AudioManager в Unity 3D.

Очікувані результати

Сформувати у студентів знання щодо реалізації звукового супроводження ігрових додатків за допомогою використання аудіо-компонентів Unity 3D та методів активації звукових ефектів за допомогою програмного коду.

Імпорт звукових ефектів

Щоб у вас з'явилася можливість відтворення звуків, потрібно імпортувати аудіофайли в Unity-проект. Процедура починається з підбору файлів потрібного формату, які потім переносяться в Unity і налаштовуються під ваші цілі.

Формати аудіофайлів, які підтримує Unity:

WAVE, WAV. Waveform Audio File Format (WAVE, WAV, від англ. Waveform - «в формі хвилі») - формат файлу-контейнера для зберігання запису оцифрованого аудіопотоку, підвид RIFF. Цей контейнер, як правило, використовується для зберігання нестислого звуку в імпульсно-кодової модуляції. Однак контейнер не накладає жодних обмежень на використання алгоритму кодування.

AIFF. Audio Interchange File Format (AIFF) - формат аудіофайлів, який застосовується для зберігання звукових даних.

AIFF був розроблений компанією Apple Computer на основі формату IFF компанії Electronic Arts і найчастіше використовується в комп'ютерах Apple Macintosh.

Звукові дані в стандартному файлі формату AIFF представляють собою нестиснуту імпульсно-кодову модуляцію. Також існує і стисла версія формату AIFF, яку називають AIFC (іноді AIFF-C), в якій для стиснення можуть бути використані різні кодеки.

AIFF, поряд з CDA і WAV, є одним з форматів, що використовуються в професійних аудіо- та відеододатках, так як на відміну від більш популярного формату MP3, звук в AIFF кодується без втрат в якості. Як і будь-які стиснені файли, файли AIFF займають набагато більше дискового простору, ніж їх стислі аналоги.

MP3. MP3 - кодек третього рівня, розроблений командою MPEG, формат файлу для зберігання аудіоінформації. Формат був ліцензований, але 23 квітня 2017 року термін дії всіх патентів закінчився і ліцензійні збори припинилися.

Формат MP3 використовує спектральні відсікання, згідно психоакустичної моделі. Звуковий сигнал розбивається на рівні по тривалості відрізки, кожен з яких після обробки упаковується в свій фрейм (кадр). Розкладання в спектр вимагає безперервності вхідного сигналу, в зв'язку з цим для розрахунків використовується також попередній і наступний фрейм. У звуковому сигналі є гармоніки з меншою амплітудою і гармоніки, що лежать поблизу більш інтенсивних - такі гармоніки відсікаються, так як середньостатистичне людське вухо не завжди зможе визначити наявність або відсутність таких гармонік. Така особливість слуху називається ефектом маскування. Також можлива заміна двох і більше сусідніх піків одним усередненим (що, як правило, і призводить до спотворення звуку).

Критерій відсікання визначається вимогою до вихідного потоку. Оскільки весь спектр актуальний, високочастотні гармоніки не відсікаються, а тільки вибірково видаляються, щоб зменшити потік інформації за рахунок розрідження спектра. Після спектральної «зачистки» застосовуються математичні методи стиснення і упаковка у фрейми. Кожен фрейм може мати кілька контейнерів, що дозволяє зберігати інформацію про декілька потоків (лівий і правий канал або центральний канал і різниця каналів). Ступінь стиснення можна варіювати, в тому числі в межах одного фрейму. Інтервал можливих значень бітрейту складає 8-320 кбіт/с.

Режими кодування і опції:

Існує три версії MP3 формату для різних потреб: MPEG-1, MPEG-2 і MPEG-2.5. Відрізняються вони можливими діапазонами бітрейту і частоти дискретизації:

32—320 кбіт/с при частотах дискретизації 32000 Гц, 44100 Гц і 48000 Гц для MPEG-1 Layer 3;

16—160 кбіт/с при частотах дискретизації 16000 Гц, 22050 Гц і 24000 Гц для MPEG-2 Layer 3;

8—160 кбіт/с при частотах дискретизації 8000 Гц і 11025 Гц для MPEG-2.5 Layer 3.

OGG. Ogg - відкритий стандарт формату мультімедіаконтейнера, який є основним файловим і потоковим форматом для мультімедіакодеков фонду Xiph.Org, а також назва проекту, що займається розробкою цього формату і кодеків для нього. Як і всі технології, що розробляються під егідою Xiph.Org, формат Ogg є відкритим і вільним стандартом, які не мають патентних чи ліцензійних обмежень.

Ogg є всього лише контейнером. Звук або відео стискаються кодеками, а результат обробки зберігається в подібних контейнерах. Контейнери Ogg можуть зберігати потоки, закодовані декількома кодеками. Наприклад, файл з відео і звуком може містити дані, закодовані аудіо- та відеокодеками.

У контейнері Ogg можна зберігати звук і відео в різних форматах (таких як MPEG-4, Dirac, MP3 та інші), але зазвичай Ogg використовується з наступними аудіокодеками:

з втратами:

Opus (раніше Harmony) - з низькою затримкою кодування (від 2,5 мс до 60 мс, настроюється) і більш високою компресією аудіо, так само бітрейт від 6 до 510 кбіт/с;

Speex - для стиснення мовного сигналу на низьких бітрейтах (~ 8-32 (кбіт / с)/канал);

Vorbis - для стиснення звуку на середніх і високих бітрейтах (~ 16-500 (кбіт / с)/канал).

без втрат:

FLAC - для обробки звукових архівів та інших аудіо, що вимагають високої якості відтворення.

MOD. MOD - формат файлів, розроблений для створення, зберігання та відтворення музичних композицій на ПК Amiga. Свою назву отримав від того, що став першим форматом, що зберігає свої фрагменти (наприклад, семпли) в інших файлах (принцип модульності). Файли цього формату мають, як правило, розширення .mod.

Кожен файл формату MOD містить в собі оцифровані записи реального звучання інструментів, так звані семпли. Композитор, який пише в форматі MOD, використовує програму, яка називається трекером, в якій вказує, який саме інструмент, в який час, якою нотою і якою з октав повинен прозвучати. Послідовність нот записується в список - трек, а кілька паралельно відтворюваних треків утворюють блок, який називається патерном. Патерни, які створюються композитором отримують номери, після чого композитор може в довільній формі вказувати який патерн і коли повинен прозвучати. Сукупність патернів і утворює модуль - файл у форматі MOD.

XM. Формат XM – це розширений модуль MOD, тип аудіофайлу FastTracker 2, який було введено розробником - демогрупою Triton.

XM-файл є мультісемплінгом за допомогою доступних інструментів з об'ємним сигналом в панорамній оболонці, а також стислою структурою. Формат розширив список доступних команд ефектів і каналів, додав 16-бітну підтримку і запропонував альтернативну таблицю частот для портаменто (спосіб виконання, при якому наступна нота не відразу береться точно (в звуко-висотному відношенні), а використовується плавний перехід до потрібної висоти від попередньої ноти).

XM є основним форматом для більшості трекерної музики.

Колекцію аудіофайлів необхідно імпортувати в Unity.

Для імпорту необхідних ресурсів (аудіофайлів) на вкладці **Project** в розділі **Assets** створюємо папку для зберігання аудіофайлів «**Sound**». Переходимо в папку, вибираємо меню «**Assets → Import NewAssets**» та імпортуємо необхідні аудіофайли (рис. 4.1). В нашому випадку це файл формату .mp3.

На панелі **«Inspector»** можна задати параметри імпорту аудіофайлів.

Прапорець «**Force To Mono**» дозволяє вибрати між моно- і стереозвуком.

Прапорець «**LoadInBackground»** дозволяє завантаження аудіофайлів у фоновому режимі.

Прапорець «**Ambisonic»** вказує на те, що аудіофайл створено за технологією **Ambisonics**.

Список «**Load Type**» дозволяє вказати, яким чином комп'ютер буде завантажувати дані з файлу. Параметр «**Vorbis**», вказує на те, що буде завантажено аудіоформат зі стисненням. Короткі звукові файли в стисненні не потребують, тому для них необхідно обрати варіант PCM (Pulse Code Modulation - імпульсно-кодова модуляція). Параметр ADPCM – є варіацією PCM та надає більш якісний звук.

Відтворення звукових ефектів

Для відтворення звуку в Unity необхідно додати три компоненти **AudioClip**, **AudioSource** та **AudioListener**.

Компонент **Audio Clip** містить дані аудіо, які використовуються в джерелах аудіо. Unity підтримує моно, стерео і мультиканальні звукові ресурси (до восьми каналів). Unity може імпортувати такі типи файлів: .aif, .wav, .mp3, .ogg. Також, Unity може імпортувати трекерні модулі з наступних типів файлів: .xm, .mod, .it, .s3m. Ресурси з трекерними модулями працюють так само, як і будь-які інші аудіо ресурси в Unity, щоправда для них недоступний перегляд форми сигналу в інспекторі імпорту.

Компонент **Audio Source** відтворює **Audio Clip** в сцені. Якщо **Audio Clip** є 3D кліпом, джерело програється в заданому положенні в просторі і буде приглушатися в залежності від відстані. Аудіо може бути розподілене по колонках за допомогою властивості **Spread** і трансформуватися між 3D і 2D за допомогою властивості **PanLevel**. Можна контролювати залежність цих ефектів від відстані за допомогою кривих загасання. Для збагачення аудіо ряду, до джерела можна застосовувати окремі аудіо фільтри.

Компонент **Audio Listener** поводиться як мікрофон. Він отримує вхідні дані з будь-якого джерела звуку (Audio Source) в сцені і програє звуки через динаміки. Для більшості додатків має сенс додавати **Audio Listener** до головної камери - об'єкту **Main Camera**. Більш того, до **Audio Listener** можна додавати аудіо ефекти, щоб застосувати їх до всіх чутних в сцені звуків.

У компонента **Audio Listener** немає властивостей. Його досить просто додати, щоб він запрацював. За замовчуванням, він завжди додається до об'єкту **Main Camera**.

Audio Listener працює в зв'язці з джерелами звуку (компонент **Audio Source**), дозволяючи вам створювати акустичне оточення в іграх. Якщо **Audio Listener** додано до **GameObject** в сцені, то будь-які досить близькі до слухача джерела будуть чутні в динаміках. У кожній сцені може бути тільки один **Audio Listener** для коректної роботи системи.

Якщо джерела у форматі 3D, тоді слухач буде імітувати положення, швидкість і орієнтацію звуку в 3D просторі. В 2D режимі буде ігноруватися будь-яка 3D обробка.

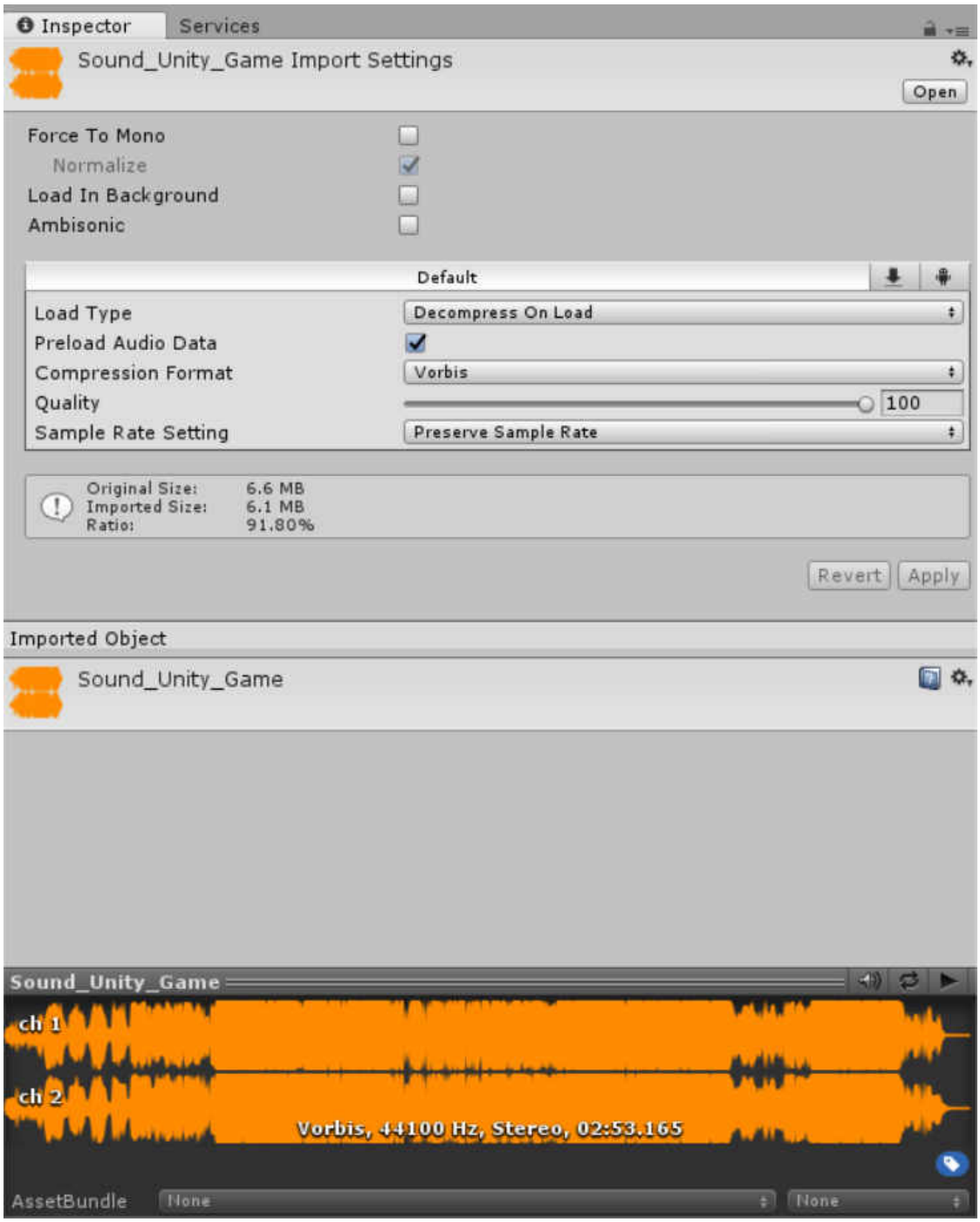

Рисунок 4.1 – Імпорт аудіофайлу

Відтворення аудіокліпу

Налаштуємо звук в Unity. **AudioClip** вже імпортовано, до камери додано компонент **Audio Listener** за замовчуванням. Залишилося додати тільки компонент **Audio Source.**

Audio Source виступає в ролі контролера, який запускає і зупиняє відтворення аудіокліпу.

Для створення нового джерела звуку **Audio Source** необхідно вибрати пункт меню «**GameObject→ Create Empty**».

На панелі «**Hierarchy**» виділіть щойно створений **GameObject**. На панелі «**Inspector**» за допомогою кнопки «**Add Component→ Audio→ Audio Source**» додайте компонент **Audio Source.**

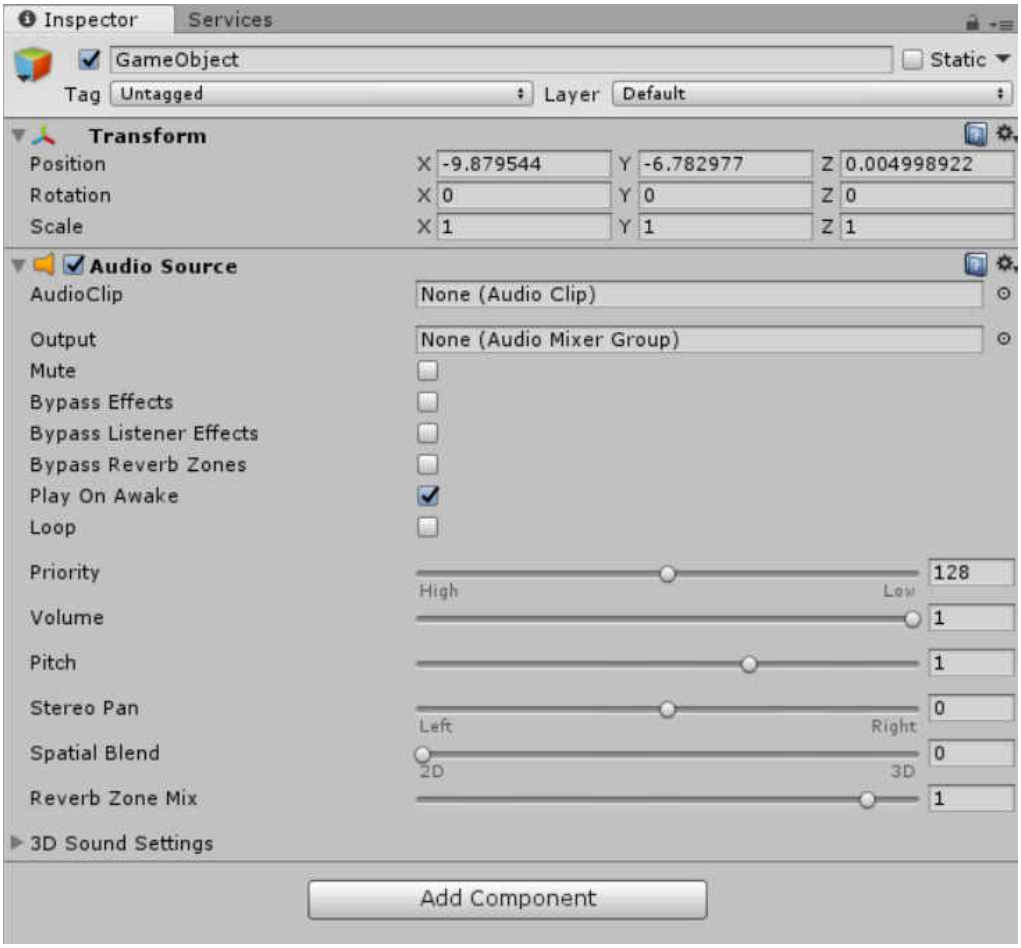

Рисунок 4.2 – Властивості компоненту **Audio Source**

На панелі «**Inspector**» вкажіть властивість **AudioClip** компонента **Audio Source (**аудіокліп, який треба прослухати). Для цього необхідно перетягнути файл зі звуком зі вкладки **Project** на чарунку **AudioClip** панелі «**Inspector**».

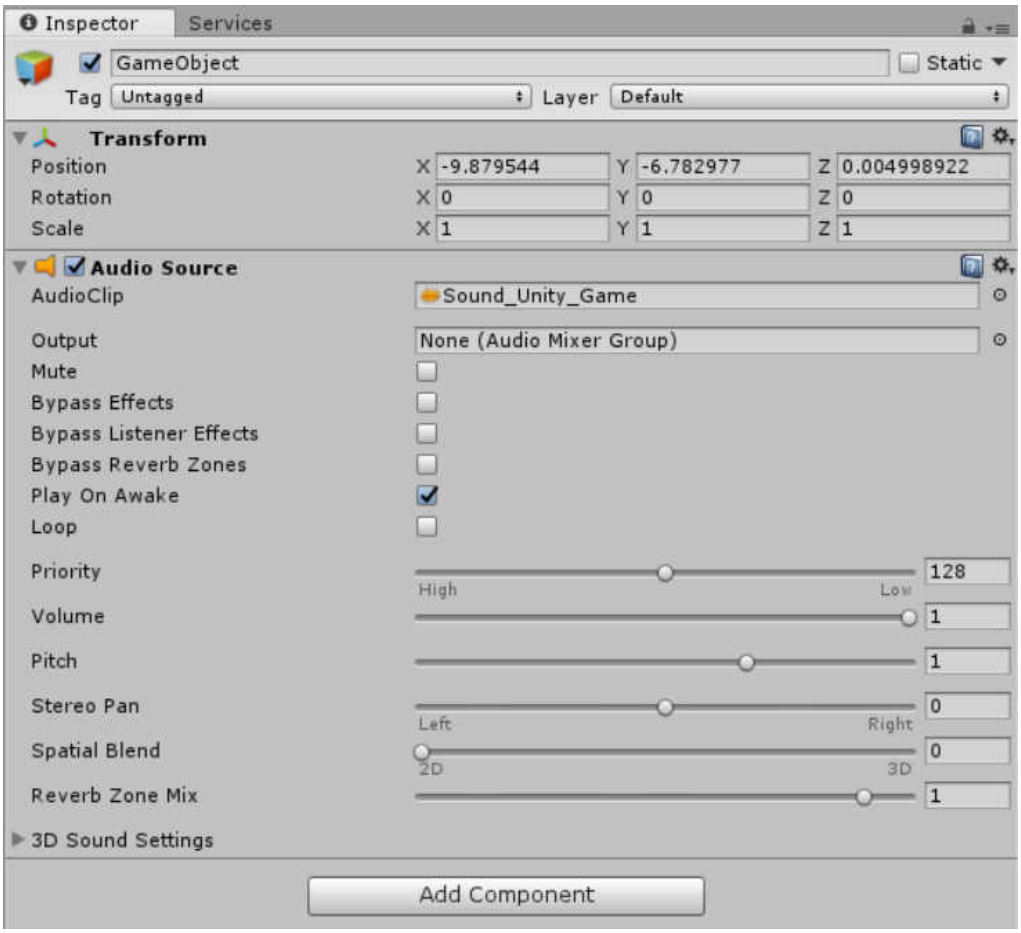

Рисунок 4.3 – Властивість **AudioClip** компонента **Audio Source**

Встановіть прапорці **Play On Awake** та **Loop,** обов'язково переконайтеся, що прапорець **Mute** знято.

Диспетчер AudioManager

Audio Manager (менеджер звуків) дозволяє налаштовувати максимальну гучність всіх звуків, які буде відтворено в сцені. Щоб відкрити цей менеджер, виберіть пункт меню «**Edit → Project Settings → Audio**» (рис. 4.4).

Властивості диспетчеру AudioManager:

Global Volume.Гучність всіх звуків.

Volume Rolloff Scale. Встановлює глобальний фактор ступеня загасання для логарифмічних згасаючих джерел звуку. Чим вище значення, тим швидше гучність буде затухати, і навпаки, чим менше значення, тим повільніше вона буде затухати (при значенні 1 буде імітовано «реальний світ»).

Doppler Factor. Ступінь чутності ефекту Доплера. Ефект відключений при значенні 0. 1 означає, що ефект буде досить добре чутно для об'єктів з високою швидкістю пересування.

Default Speaker Mode. Визначає режим динаміків для вашого проекту. За замовчуванням має значення 2 - стерео колонки.

System Sample Rate. Вихідна частота вибірки. Якщо встановлено значення 0, буде використовуватися коефіцієнт вибірки системи. Також зауважте, що лише деякі платформи дозволяють змінити цей параметр, наприклад, iOS або Android.

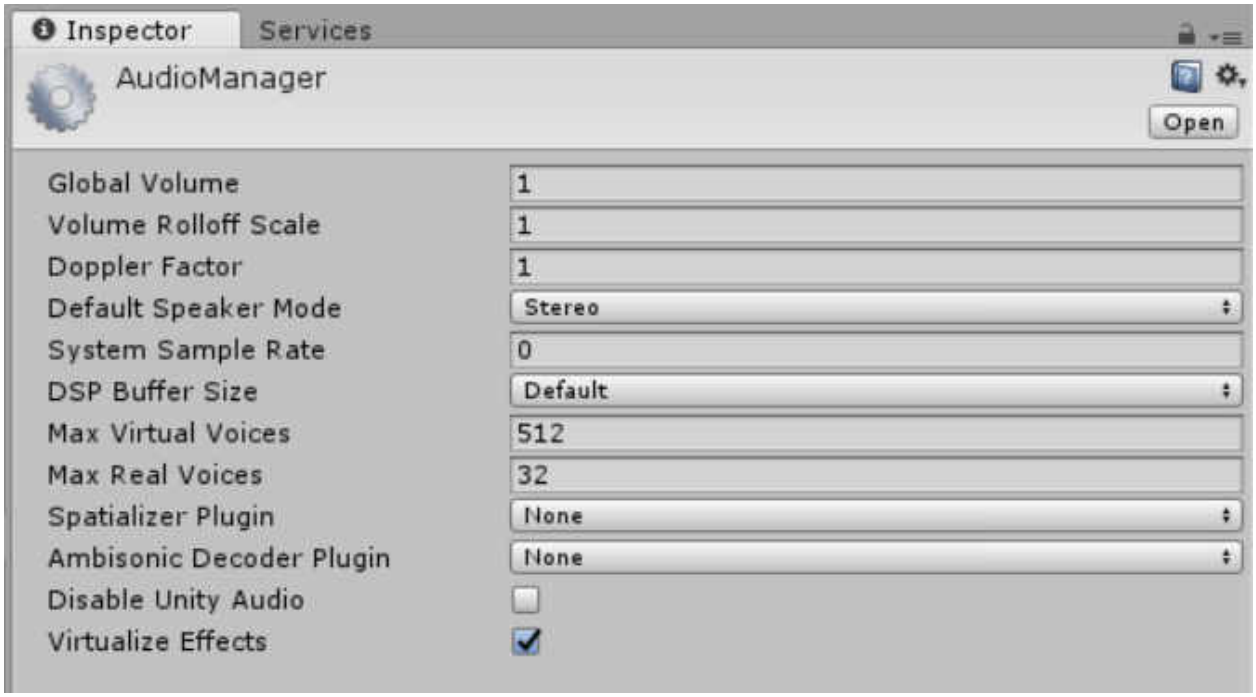

Рисунок 4.4 – Диспетчер **AudioManager**

DSP Buffer Size. Розмір DSP буфера можна встановити для оптимізації між латентністю і продуктивністю.

Default. Розмір буфера за замовчуванням

Best Latency. Краща латентність за рахунок зниження продуктивності

Good Latency. Баланс між латентністю і продуктивністю

Best Performance. Краща продуктивність за рахунок погіршення латентності.

Max Virtual Voices. Кількість віртуальних голосів, якими керує аудіосистема. Ця величина завжди повинна бути більшою, ніж кількість голосів у грі. Якщо ні, в консолі з'явиться попередження.

Max Real Voices. Кількість реальних голосів, які можна програвати одночасно. У кожному кадрі будуть вибрані найгучніші голоси.

Spatializer Plugin. Тільки для версії Unity Pro

Ambisonic Decoder Plugin. Тільки для версії Unity Pro

Disable Audio. Відключає аудіосистему в standalone збірках. Врахуйте, що це також впливає на звуки MovieTexture. У редакторі аудіо система все ще буде працювати і буде підтримувати попередній аудіо кліпів, але виклики AudioSource.Play і playOnAwake не будуть оброблятися, щоб імітувати поведінку аудіо системи standalone збірки.

Disable Unity Audio. Відключення звуку в Unity.

Virtualize Effects. Віртуалізація звукових ефектів.

Активація звукових ефектів з коду

Ви можете програвати обраний аудіокліп, використовуючи **Play**, **Pause** і **Stop**. Ви також можете налаштовувати його гучність під час відтворення, використовуючи властивість **volume**, або перемотувати, використовуючи **time**. При використанні **PlayOneShot** на одному **AudioSource** можуть бути програні відразу кілька звуків. Ви можете програвати кліп в статичній точці в тривимірному просторі, використовуючи **PlayClipAtPoint**.

AudioSource.Play

public void Play (ulong delay = 0);
public void Play (ulong delay = 0);

Параметри:

delay @param delay Затримка, вимірюється в кількості семплів, використовуючи частотою дискретизації (частоту семплювання) 44100 Гц (маючи на увазі, що Play (44100) забезпечить затримку рівно в 1 секунду).

AudioSource.Play програє **clip** з затримкою перед початком відтворення (опціонально).

Слід зауважити, що параметр затримки **delay** застарів. Замість нього використовується більш нова функція **PlayDelayed**, яка задає затримку в секундах.

Для досягнення точного відтворення AudioClip з відмінною від 44.1 кГц частотою дискретизації, слід самим робити відповідні обчислення. Затримка на джерелі аудіо з прив'язаним AudioClip з частотою дискретизації, скажімо, 32 кГц, з 16к семплами (0.5 секунд) здійснюється за допомогою Play (22050). ((44100/32000) * 16000 = 22050

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(AudioSource))]
public class ExampleClass : MonoBehaviour {
   void Start() {
      AudioSource audio = GetComponent<AudioSource>();
```

```
        audio.Play();
        audio.Play(44100);
    }
}
```

Слід також зауважити, що **AudioSource.PlayScheduled API** забезпечить більш точний контроль під час програвання звуку.

AudioSource.PlayScheduled

public void PlayScheduled(double time);

Параметри:

time @param time Час в секундах на абсолютній часовій шкалі, на який посилається **AudioSettings.dspTime** як на час початку відтворення звуку.

AudioSource.PlayScheduled програє clip в заданий час на абсолютній часовій шкалі з **AudioSettings.dspTime**.

Це кращий спосіб розміщення **AudioClips** в музичних програвачах, оскільки він не залежить від частоти кадрів і дає аудіосистемі досить часу, щоб підготувати відтворення звуку при отриманні його з джерела, де відкриття і буферизація займають багато часу (потокове відтворення) без раптових пікових навантажень на CPU.

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(AudioSource))]
public class ExampleClass : MonoBehaviour {
    public float bpm = 140.0F;
    public int numBeatsPerSegment = 16;
    public AudioClip[] clips = new AudioClip[2];
    private double nextEventTime;
    private int flip = 0;
    private AudioSource[] audioSources = new AudioSource[2];
    private bool running = false;
    void Start() {
        int i = 0;
        while (i < 2) {
            GameObject child = new GameObject("Player");
            child.transform.parent = gameObject.transform;
            audioSources[i] = child.AddComponent<AudioSource>();
```

```
            i++;
        }
        nextEventTime = AudioSettings.dspTime + 2.0F;
        running = true;
    }
    void Update() {
        if (!running)
            return;
        double time = AudioSettings.dspTime;
        if (time + 1.0F > nextEventTime) {
            audioSources[flip].clip = clips[flip];
            audioSources[flip].PlayScheduled(nextEventTime);
            Debug.Log("Scheduled source " + flip + " to start at time " +
nextEventTime);
            nextEventTime += 60.0F / bpm * numBeatsPerSegment;
            flip = 1 - flip;
        }
    }
}
```

Приклад **AudioSource.SetScheduledEndTime** демонструє, як можна програвати звукозаписи без переходів між ними. Підхід полягає в тому, щоб мати два джерела аудіо з прив'язаними кліпами і складати чергу для кожного кліпу, використовуючи його джерело.

AudioSource.SetScheduledEndTime

public void SetScheduledEndTime(double time);

Параметри:

time @param time Час в секундах.

Змінює час, в який звук, вже запланований на відтворення, закінчить відтворення. Зверніть увагу, що в залежності від часу, не кожен запит на перепланування може бути виконаний.

Слід пам'ятати, що заданий час це все ще час на абсолютній часовій шкалі, що означає зупинку відтворення звуку при досягненні цього часу, незалежно від того, коли він почав відтворюватися. Так, якщо у вас є звук довжиною 5 секунд і ви хочете програти його в момент T і зупинити через 3 секунди (тобто заглушити останні 2 секунди звуку), вам слід задати час кінця відтворення як T + 3. Ця функція може бути корисна в музичних системах для подолання розривів у сигналах, викликаних кодеками з втратами, які працюють по кадрам.

```
using UnityEngine;
using System.Collections;
 [RequireComponent(typeof(AudioSource))]
public class ExampleClass : MonoBehaviour
{
    public AudioClip sourceClip;
    private AudioSource audio1;
    private AudioSource audio2;
    private AudioClip cutClip1;
    private AudioClip cutClip2;
    private float overlap = 0.2F;
    private int len1 = 0;
    private int len2 = 0;
    void Start()
    {
        GameObject child;
        child = new GameObject("Player1");
        child.transform.parent = gameObject.transform;
        audio1 = child.AddComponent<AudioSource>();
        child = new GameObject("Player2");
        child.transform.parent = gameObject.transform;
        audio2 = child.AddComponent<AudioSource>();
        int overlapSamples;
        if (sourceClip != null)
        {
            len1 = sourceClip.samples / 2;
            len2 = sourceClip.samples - len1;
            overlapSamples = (int)(overlap * sourceClip.frequency);
            cutClip1 = AudioClip.Create("cut1", len1 + overlapSamples,
sourceClip.channels, sourceClip.frequency, false, false);
            cutClip2 = AudioClip.Create("cut2", len2 + overlapSamples,
sourceClip.channels, sourceClip.frequency, false, false);
            float[] smp1 = new float[(len1 + overlapSamples) *
sourceClip.channels];
            float[] smp2 = new float[(len2 + overlapSamples) *
sourceClip.channels];
            sourceClip.GetData(smp1, 0);
            sourceClip.GetData(smp2, len1 - overlapSamples);
            cutClip1.SetData(smp1, 0);
            cutClip2.SetData(smp2, 0);
        }
```

```
            else
            {
                overlapSamples = (int)overlap * cutClip1.frequency;
                len1 = cutClip1.samples - overlapSamples;
                len2 = cutClip2.samples - overlapSamples;
            }
        }
        void OnGUI()
        {
            if (GUI.Button(new Rect(10, 50, 230, 40), "Trigger source"))
                audio1.PlayOneShot(sourceClip);

            if (GUI.Button(new Rect(10, 100, 230, 40), "Trigger cut 1"))
                audio1.PlayOneShot(cutClip1);
            if (GUI.Button(new Rect(10, 150, 230, 40), "Trigger cut 2"))
                audio1.PlayOneShot(cutClip2);

            if (GUI.Button(new Rect(10, 200, 230, 40), "Play stitched"))
            {
                audio1.clip = cutClip1;
                audio2.clip = cutClip2;
                double t0 = AudioSettings.dspTime + 3.0F;
                double clipTime1 = len1;
                clipTime1 /= cutClip1.frequency;
                audio1.PlayScheduled(t0);
                audio1.SetScheduledEndTime(t0 + clipTime1);
                Debug.Log("t0 = " + t0 + ", clipTime1 = " + clipTime1 + ",
cutClip1.frequency = " + cutClip1.frequency);
                Debug.Log("cutClip2.frequency = " + cutClip2.frequency +
", samplerate = " + AudioSettings.outputSampleRate);
                audio2.PlayScheduled(t0 + clipTime1);
                audio2.time = overlap;
            }
        }
    }
```

AudioSource.time

```
public float time;
```

Позиція відтворення, в секундах. Використовується для зчитування поточного часу відтворення або для перемотування на новий час відтворення.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
   AudioSource audio;

   void Start() {
      audio = GetComponent<AudioSource>();
   }

   void Update() {
      if (Input.GetKeyDown(KeyCode.Return)) {
         audio.Stop();
         audio.Play();
      }
      Debug.Log(audio.time);
   }
}
```

AudioSource.clip

```
public AudioClip clip;
```

Обраний за замовчуванням AudioClip для програвання.

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(AudioSource))]
public class ExampleClass : MonoBehaviour {
   public AudioClip otherClip;
   IEnumerator Start() {
      AudioSource audio = GetComponent<AudioSource>();
      audio.Play();
      yield return new WaitForSeconds(audio.clip.length);
      audio.clip = otherClip;
      audio.Play();
   }
}
```

AudioSettings class in UnityEngine

Контролює загальні налаштування аудіо з скрипта. Встановлює вихід для динаміків.

Статичні змінні:

driverCapabilities. Повертає можливі конфігурації динаміків для поточного аудіо драйвера.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
   void Example() {
      AudioSettings.speakerMode = AudioSettings.driverCapabilities;
   }
}
```

dspTime. Повертає поточний час аудіосистеми.

Ця величина задається в секундах і ґрунтується на реальному числі семплів (кроків дискретизації), які оброблюються аудіосистемою і, відповідно, вона набагато точніша, ніж час, отриманий за допомогою властивості Time.time.

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(AudioSource))]
public class ExampleClass : MonoBehaviour
{
      public double bpm = 140.0F;
      public float gain = 0.5F;
      public int signatureHi = 4;
      public int signatureLo = 4;
      private double nextTick = 0.0F;
      private float amp = 0.0F;
      private float phase = 0.0F;
      private double sampleRate = 0.0F;
      private int accent;
      private bool running = false;
      void Start()
      {
                  accent = signatureHi;
```

```
                double startTick = AudioSettings.dspTime;
                sampleRate = AudioSettings.outputSampleRate;
                nextTick = startTick * sampleRate;
                running = true;
        }
        void OnAudioFilterRead(float[] data, int channels)
        {
                if (!running)
                        return;

                double samplesPerTick = sampleRate * 60.0F / bpm *
4.0F / signatureLo;
                double sample = AudioSettings.dspTime * sampleRate;
                int dataLen = data.Length / channels;
                int n = 0;
                while (n < dataLen)
                {
                        float x = gain * amp * Mathf.Sin(phase);
                        int i = 0;
                        while (i < channels)
                        {
                                data[n * channels + i] += x;
                                i++;
                        }
                        while (sample + n >= nextTick)
                        {
                                nextTick += samplesPerTick;
                                amp = 1.0F;
                                if (++accent > signatureHi)
                                {
                                        accent = 1;
                                        amp *= 2.0F;
                                }
                                Debug.Log("Tick: " + accent + "/"
+ signatureHi);
                        }
                        phase += amp * 0.3F;
                        amp *= 0.993F;
                        n++;
                }
        }
    }
```

outputSampleRate. Зчитує або встановлює поточну швидкість виведення мікшера.

speakerMode. Встановлює або зчитує поточну конфігурацію динаміків.

Контрольні запитання

- Для чого в Unity 3D використовуються компоненти AudioClip, AudioSource, AudioListener?
- Яким чином в Unity 3D здійснюється активації звукових ефектів?
- Назвіть компоненти Unity 3D для роботи зі звуком.
- Для чого в Unity 3D використовується AudioManager?
- Назвіть основні етапи налаштування центрального диспетчера управління звуком в Unity 3D.

ЛЕКЦІЯ №5
ТЕМА: «АНІМАЦІЯ ОБ'ЄКТІВ В UNITY 3D»

Анотація

Лекція знайомить з основними поняттями анімації та типами анімації об'єктів в Unity 3D.

Мета лекції

Ознайомити студентів з основними типами анімації об'єктів в Unity 3D. Розглянути методи анімація твердого тіла, анімація на основі скелета, анімації спрайтами (окремі спрайти, атлас спрайтів), анімації, що заснована на фізиці (використання фізичної системи Unity), відеоанімації (відтворення відеофайлів у вигляді анімованих текстур), анімації частинками, анімація за допомогою системи Mecanim, анімації персонажів.

Очікувані результати

Сформувати у студентів знання щодо реалізації в ігрових додатках основних типів анімації об'єктів, анімації персонажів, застосування анімації предметів за допомогою системи Mecanim.

Типи анімації об'єктів в Unity 3D

Unity 3D підтримує наступні типи анімації: анімація твердого тіла, анімація на основі скелета, анімації спрайтами (окремі спрайти, атлас спрайтів), анімація, що заснована на фізиці (використання фізичної системи Unity), відеоанімація (відтворення відеофайлів у вигляді анімованих текстур), анімація частинками, анімація персонажів за допомогою системи Mecanim.

Анімація твердого тіла

Анімація твердого тіла використовується для створення готових послідовностей анімації, які переміщують або змінюють властивості об'єктів, з урахуванням того, що об'єкти є єдиним цілим.

При анімації твердого тіла зміни в ключових кадрах поширюються тільки на цілі об'єкти і їх властивості високого рівня. Вони не стосуються під властивостей і внутрішніх компонентів і не змінюють суті або власної форми об'єктів.

Анімація на основі скелета

Цей тип анімації змінює в ключових кадрах не становище, обертання або масштаб об'єкта, а рух і деформацію його внутрішніх частин. Як правило, анімація на основі скелета створюється як повна послідовність анімації в програмі 3D-моделювання та імпортується в Unity як частина файлу, який може бути оброблений і доступний за допомогою системи анімації Mecanim.

Анімація спрайтами

Анімація складається з послідовності зображень або кадрів і програє їх в потрібному порядку і з заданою швидкістю для отримання безперервної анімації.

Анімація, що заснована на фізиці (використання фізичної системи Unity)

Тип анімації, який дозволяє керувати поведінкою об'єктів за допомогою фізичної системи Unity. Наприклад, падіння об'єкта на землю під дією сили тяжіння.

Відеоанімація (відтворення відеофайлів у вигляді анімованих текстур)

Можливість відтворювати відеофайли у вигляді анімованих текстур як на настільних платформах, так і на мобільних пристроях, таких як iOS і Android.

Анімація частинками

Використання системи частинок сюрікен (**Shuriken particle system**) для створення дощу, снігу, феєрверків, іскор і в іншої нематеріальної анімації з безліччю рухомих частинок.

Анімація персонажів за допомогою системи Mecanim

Можливість створювати анімовані персонажі за допомогою системи анімації Mecanim.

Unity пропонує широкий спектр інструментів для створення анімації. Ці інструменти називаються вбудованими функціями анімації, вони включають в себе наступне:

1. Редактор анімації Unity (**Unity animation editor**) для анімації твердих тіл, наприклад для створення дверей, камери, що літає, платформи ліфта і іншого;

2. Система частинок сюрікен (**Shuriken particle system**) для створення дощу, снігу, феєрверків, іскор і в іншої нематеріальної анімації з безліччю рухомих частинок.

Вікно анімації Animation window

Вікно анімації (**Animation window**) - це повнофункціональний редактор анімації. Він дозволяє виконувати анімацію об'єкта гри в часі, зберігаючи дані про анімацію у вигляді окремого незалежного активу, який називається кліпом анімації (**animation clip**), в панелі проекту.

Для демонстрації роботи вікна анімації створимо літаючу над ландшафтом камеру.

Для створення ландшафту необхідно додати в вікно «Scene» 3D Object **«Terrain»** вибравши меню **«Game Object →3D Object → Terrain»**.

Далі необхідно задати йому текстуру. Для цього в інспекторі об'єктів натискаємо на пензлик а потім на кнопку **Edit Textures**. Обираємо пункт **Add Texture**.

Відкриється вікно додавання текстур до об'єкту **Terrain** «**Add Terrain Texture»**.

У вікні «**Add Terrain Texture»** натискаємо «**Select Texture 2D**» .

У вікні «**Select Texture 2D**» вибираємо текстуру, нажимаємо на кнопку **«Add»** та додаємо її до об'єкту **Terrain**.

Для створення камери скористаємося меню **«GameObject→Camera».**

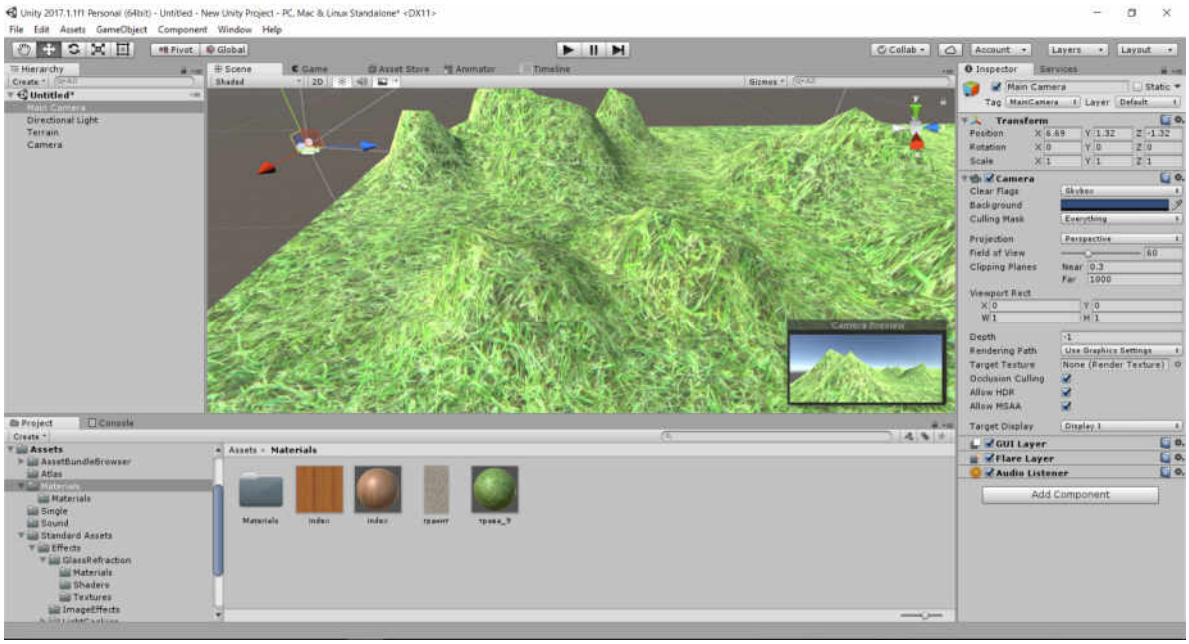

Рисунок 5.1 – Створення ландшафту та камери