



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



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

ЛЯШЕНКО О.М.

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

*Для підготовки студентів
спеціальності 121 «Інженерія програмного забезпечення»*

ХЕРСОН-2017



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

НАВЧАЛЬНИЙ МОДУЛЬ

«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 module 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.



УДК 004.4

Навчальний модуль «Розробка комп'ютерних ігор за допомогою Unity 3D» для підготовки студентів спеціальності 121 «Інженерія програмного забезпечення»/О.М. Ляшенко. – Херсон: ХНТУ, 2017. – 220 с.

Укладачі:

Ляшенко О.М., к.т.н., доцент кафедри Інформаційних 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 «Інформаційних технологій»,
Протокол №11 від 23 травня 2017 року.

Розглянуто на засіданні Вченої ради ХНТУ,
Протокол №9 від 02 червня 2017 року.



Цей матеріал ліцензовано на умовах
Ліцензії Creative Commons CC BY-
NC-SA

Із Зазначенням Авторства —
Некомерційна — Поширення На Тих
Самих Умовах 4.0 Міжнародна



ЗАГАЛЬНИЙ ВСТУП

Навчальний модуль 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-SBHE-JP «GameHub: Співробітництво між університетами та підприємствами в сфері ігрової індустрії в Україні».

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

Навчальний модуль 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 модуля містить довідник дисципліни.

Другий розділ навчального модуля 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 модуля 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 модуль 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 модуля 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

Вступ	6
1. Опис навчальної дисципліни	7
2. Перелік компетентностей та результати навчання	7
3. Міждисциплінарні зв'язки	9
4. Мета та передбачувані результати вивчення навчальної дисципліни	10
5. Календарний план семестру і структура навчальної дисципліни	12
6. Форми навчання	13
7. Порядок проведення атестації	14
8. Зворотній зв'язок	16
9. Викладацький склад та допоміжні джерела	17
10. Навчальна програма і матеріали	18

ЛЕКЦІЇ ТА МЕТОДИКА ЇХ ПРОВЕДЕННЯ

Вступ	30
1. Лекція №1. Основи роботи в Unity 3D	32
2. Лекція №2. Фізика 3D - об'єктів	48
3. Лекція №3. Розробка та використання скриптів в Unity 3D. середовище розробки «Monodevelop». Система подій «Eventsystem»	61
4. Лекція №4. Аудіо-компоненти Unity 3D. Імпорт і налаштування звуку	78
5. Лекція №5. Анімація об'єктів в Unity 3D	96
6. Лекція №6. Технологія розробки крос-платформних комп'ютерних ігор в Unity 3D	116

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОНИХ РОБІТ

Вступ	130
Обладнання	132
1. Лабораторна робота № 1.Режими роботи та інтерфейс Unity 3D. Робота з графікою	134
2. Лабораторна робота № 2.Компоненти Unity 3D для роботи з фізикою 3D-об'єктів	154
3. Лабораторна робота № 3.Робота в інтегрованому середовищі розробки Monodevelop	169
4. Лабораторна робота № 4.Імпорт аудіокліпів в ігрові додатки. Робота з диспетчером Audiomanager в Unity 3D	185

5.	Лабораторна робота № 5. Анімація спрайтами. Робота з системою анімації Mecanim	199
6.	Лабораторна робота № 6. Розробка крос-платформних комп'ютерних ігор (Windows, Android).....	210

ЛІТЕРАТУРА

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 ІТ – 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	<i>ЗК-4, ЗК-5, ЗК-6</i>
Спеціальні (фахові) компетентності	<i>ФК-1, ФК-2, ФК-3, ФК-5, ФК-6</i>
Програмні результати навчання	<i>ПРН-4, ПРН-7, ПРН-8, ПРН-10</i>

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 вивчення
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-платформність на рівні апаратної платформи. Крос-платформність 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 поняттю «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 системі Unity 3D використовуються Joint – компоненти?
- Назвіть опції Joint - компонента, які можна включити для різних ефектів.
- Для чого в системі 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 компонентами, 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 інспекторіві 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 BundleIdentifier?
- 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 3D» 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 вибудований, 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, невідповідність 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, інтонаційною 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 браузері 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.

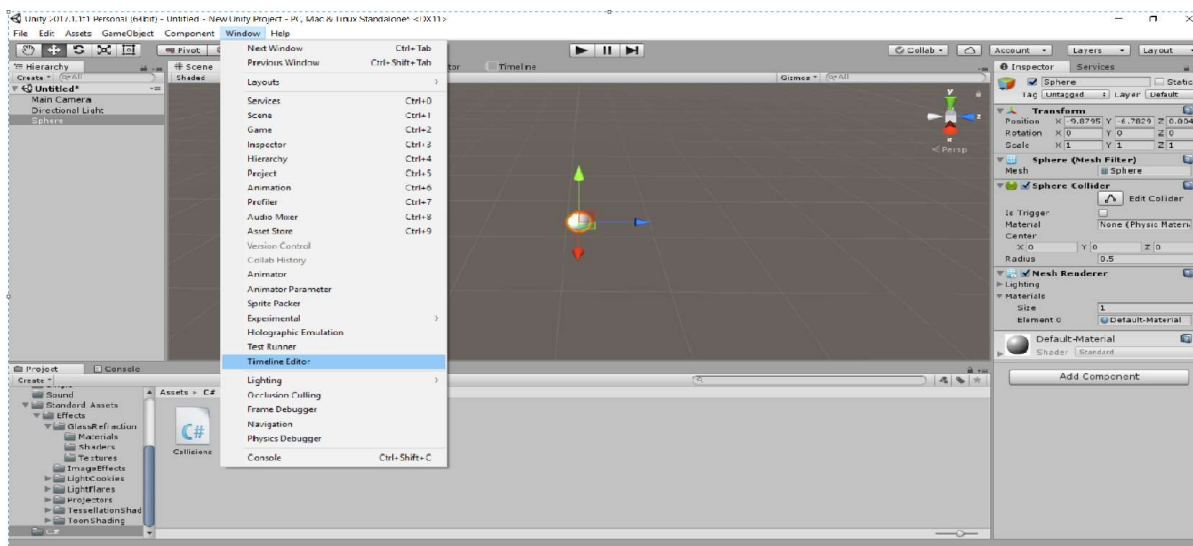


Рисунок 1.1 – Меню «Window→ Timeline Editor»

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**).

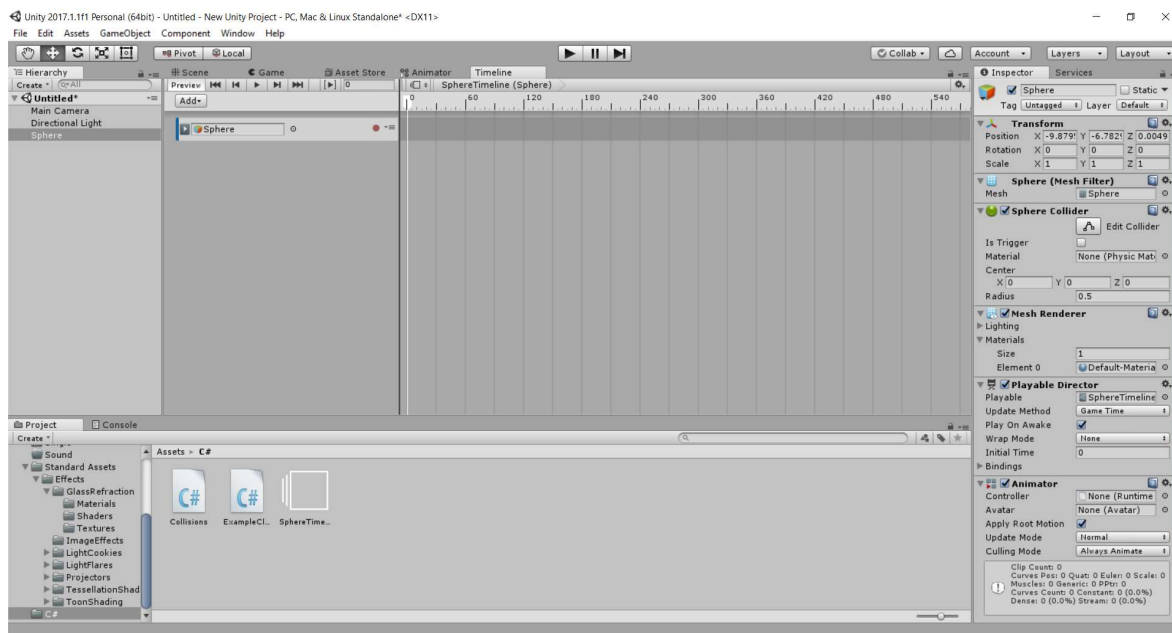


Рисунок 1.2 – Timeline Editor для об'єкту Sphere

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

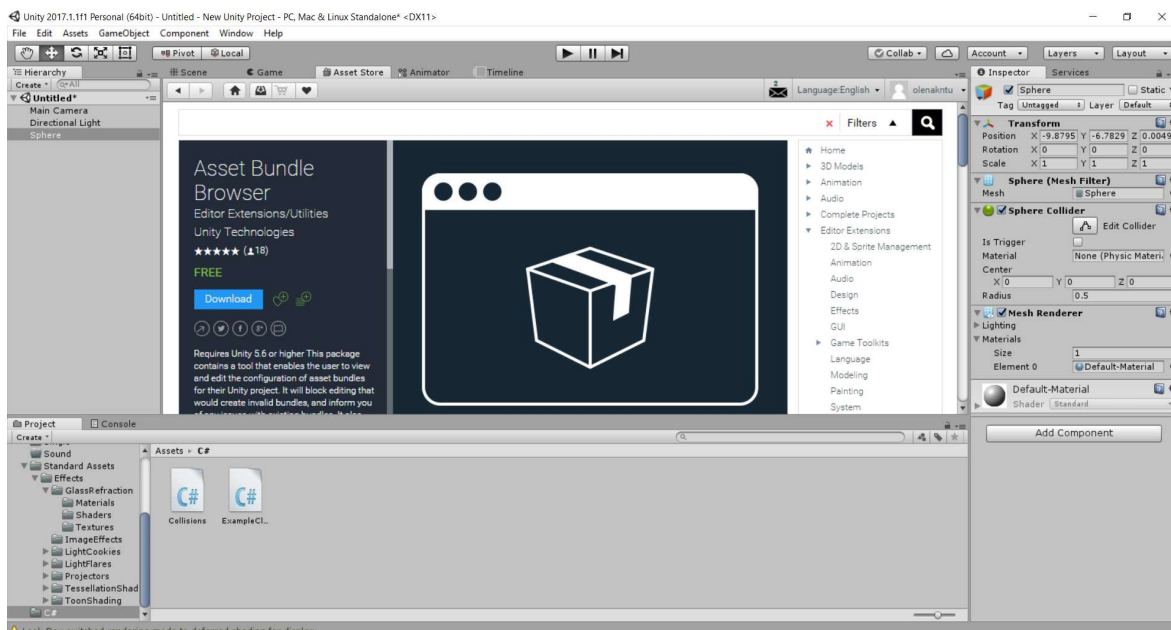


Рисунок 1.3 – Asset Store браузер Asset Bundle Browser

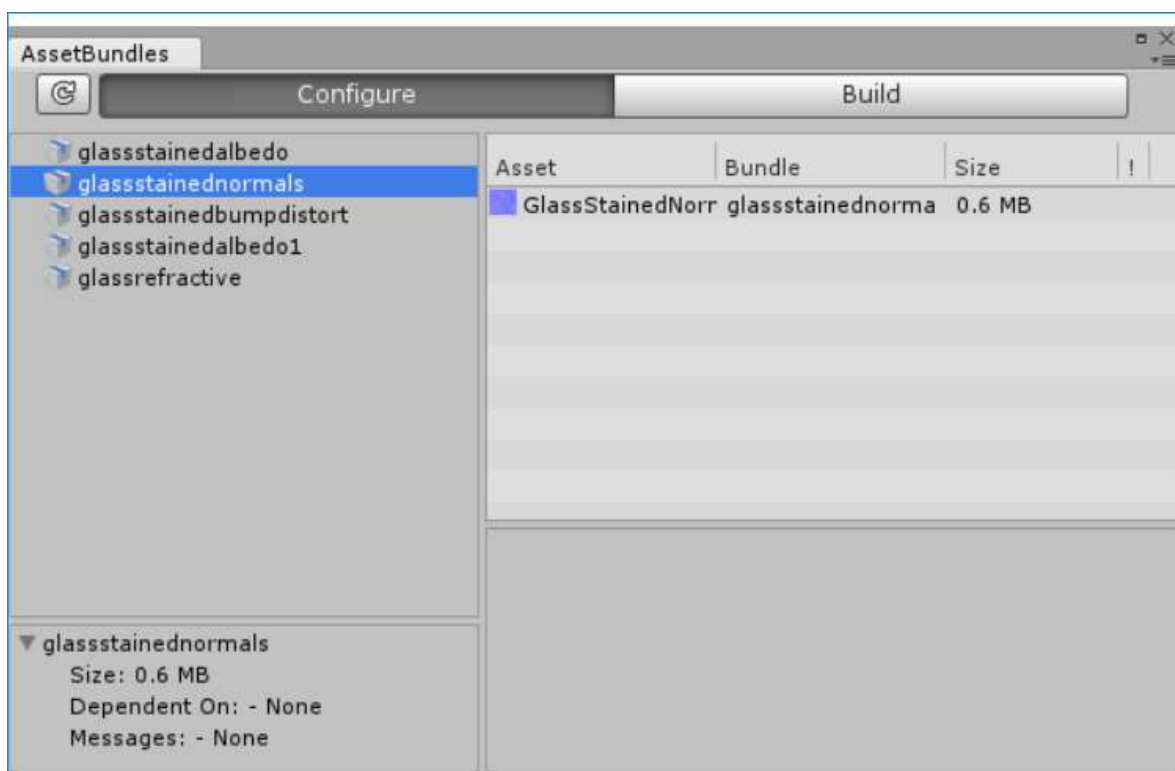


Рисунок 1.4 – Asset Bundle Browser

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 Explorer**.

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 проєкт 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).

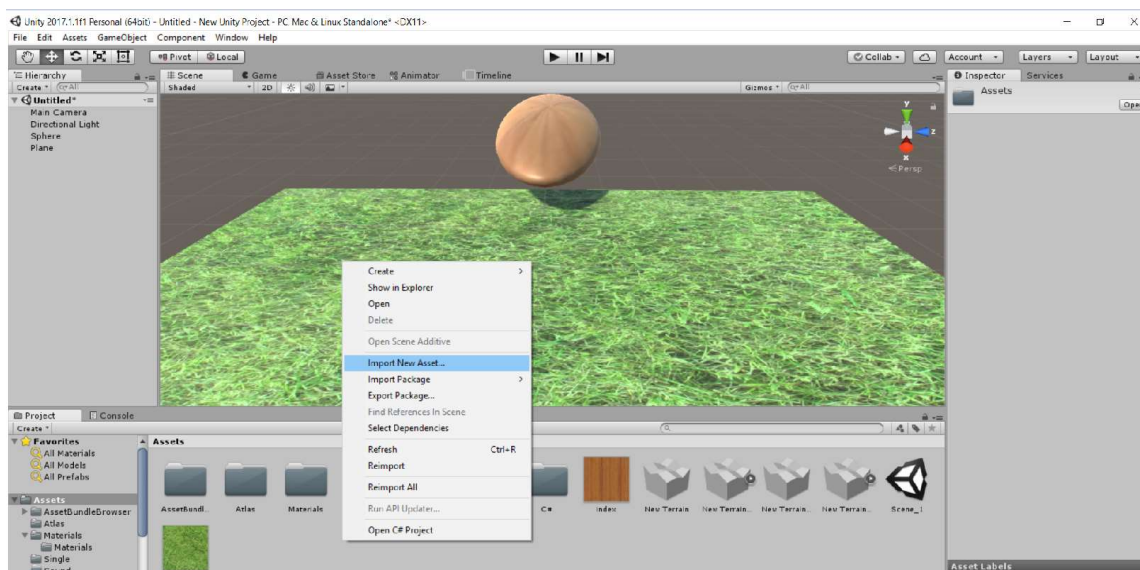


Рисунок 1.5 – Import New Assets

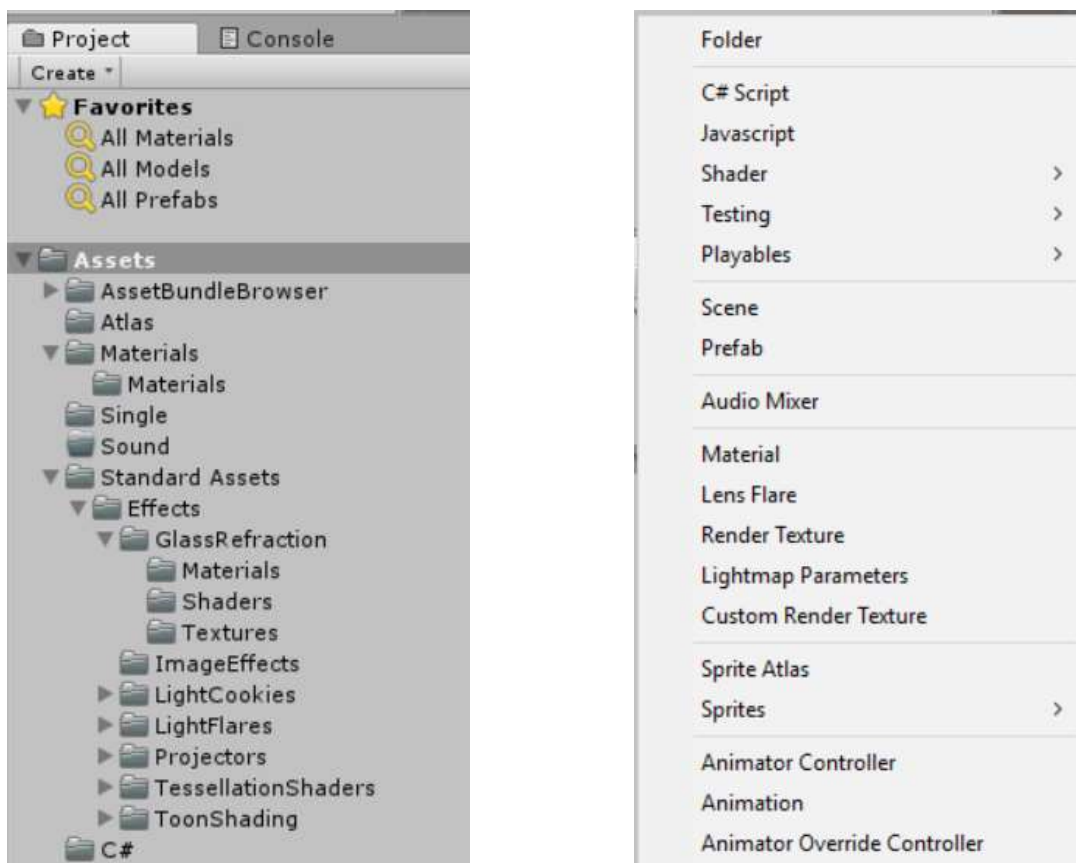
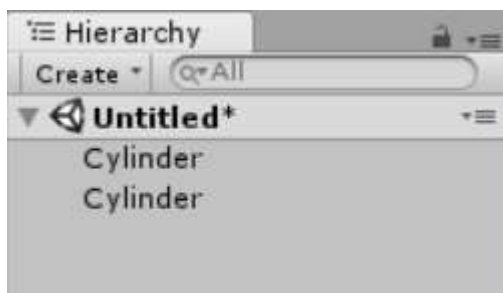


Рисунок 1.6 – Меню **Create** в **Project View**

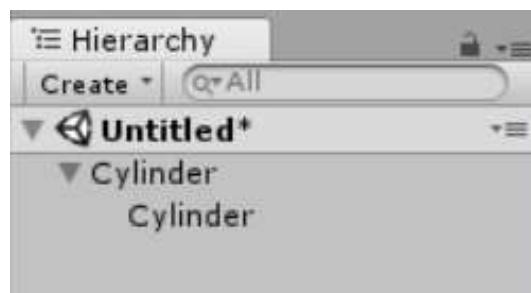


Рисунок 1.7 – Панель **Hierarchy**

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.

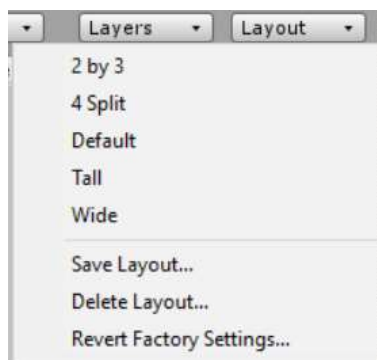


Рисунок 1.11 – **Layout Drop-down**

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.

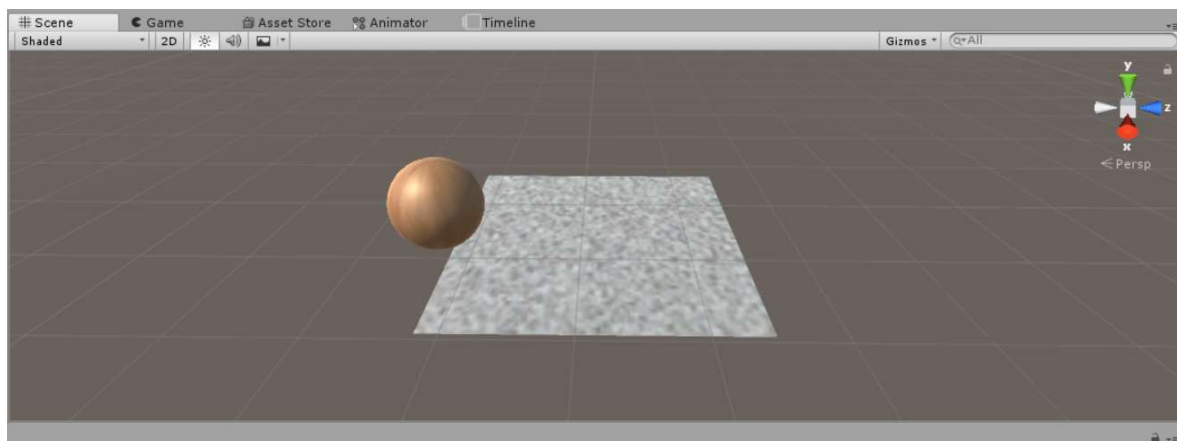


Рисунок 1.12 – **Scene View**

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 Inspector View.

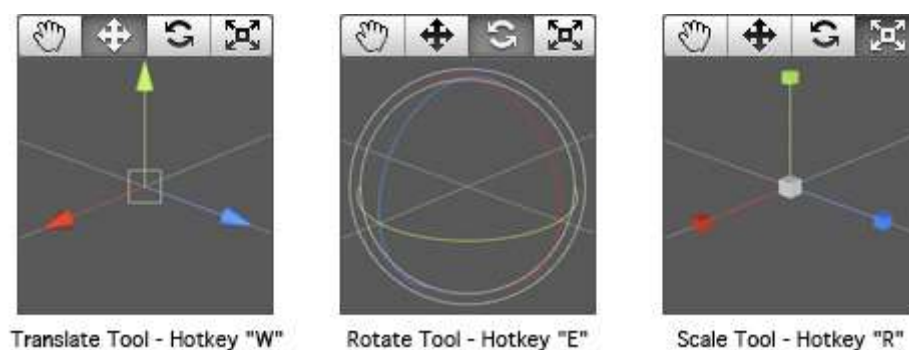


Рисунок 1.13 – Positioning GameObjects

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'єкта.

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.

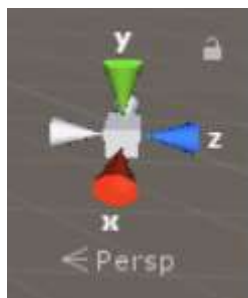


Рисунок 1.14 – Scene Gizmo

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.

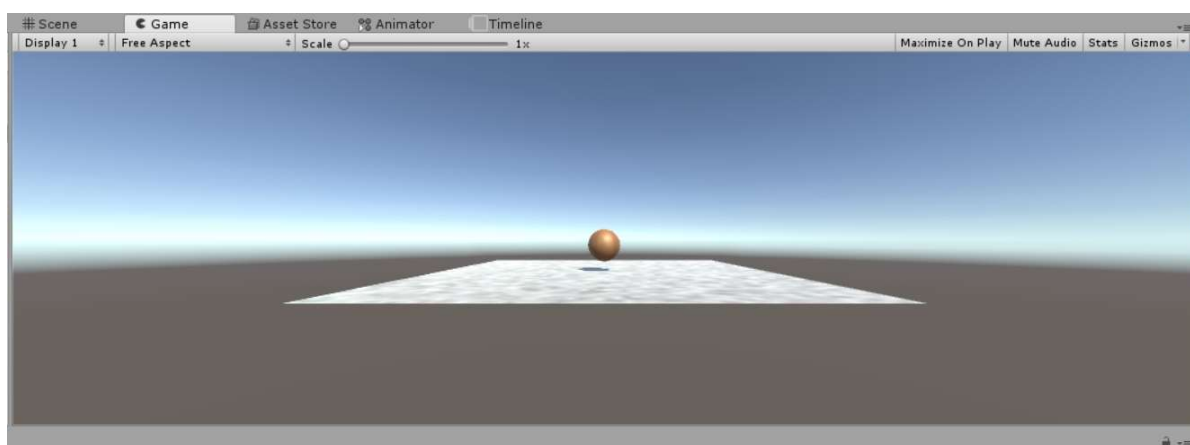


Рисунок 1.16 – Game View

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 статистику 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 компонентів (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.

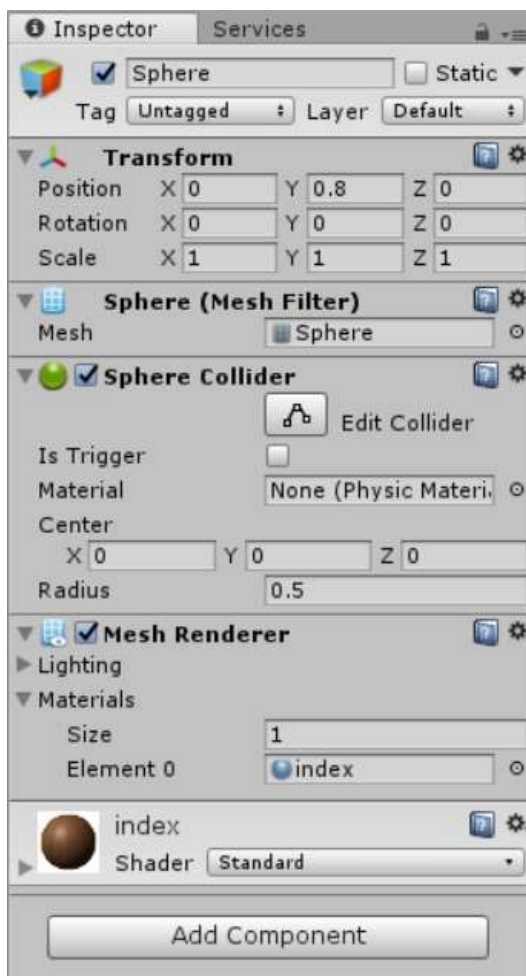


Рисунок 1.18 – Inspector

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 i Load)**.

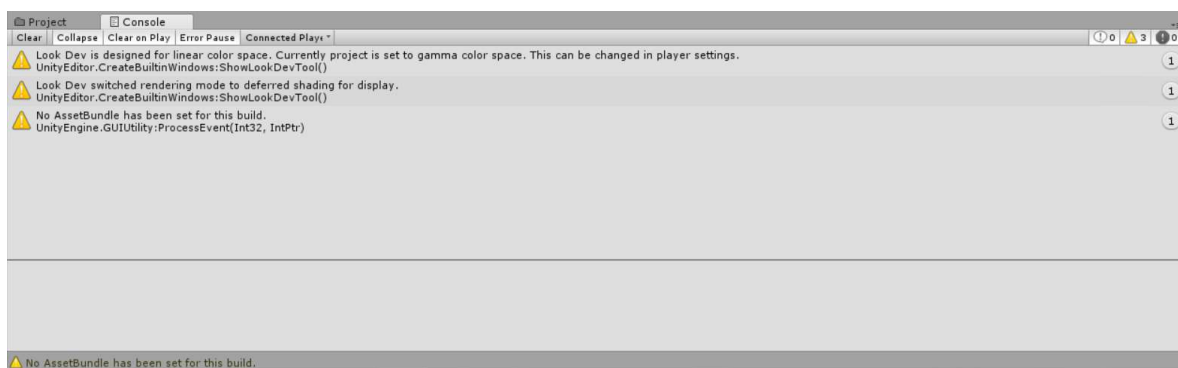


Рисунок 1.19 – Console

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.

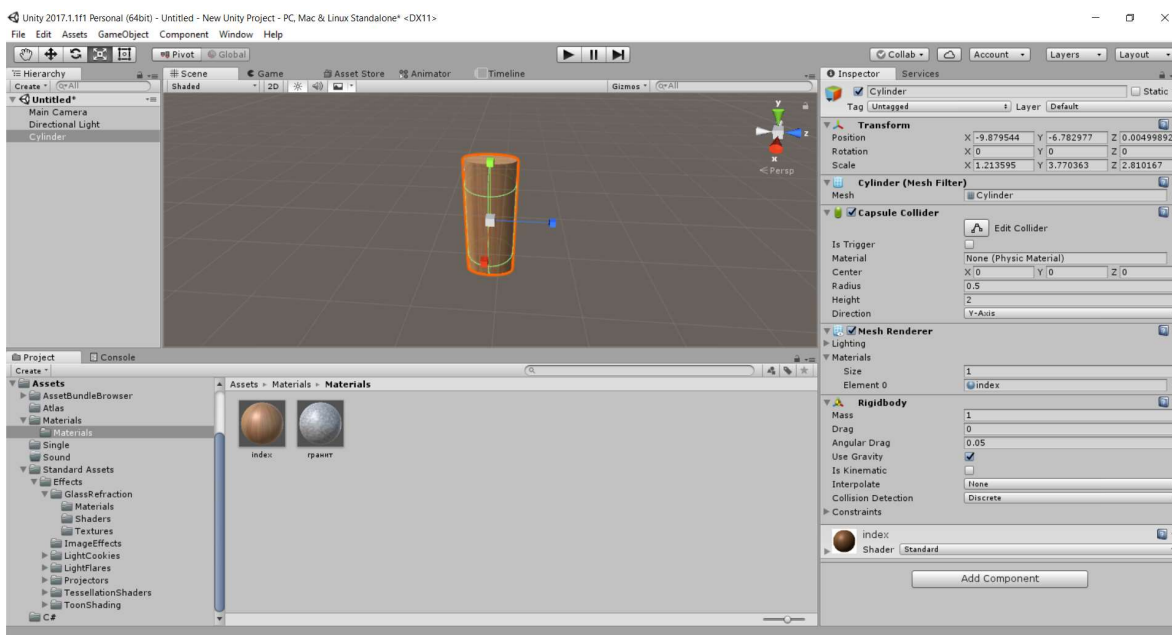


Рисунок 2.1 – Властивості компоненту **Rigidbody**

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.

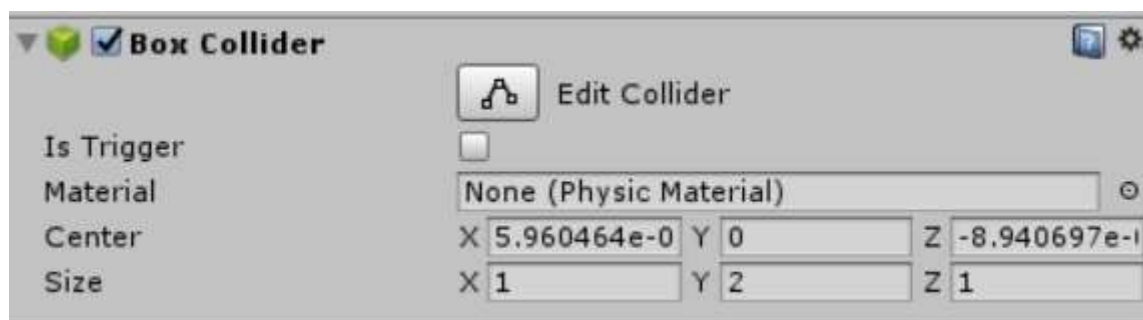


Рисунок 2.2 – Box Collider

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.

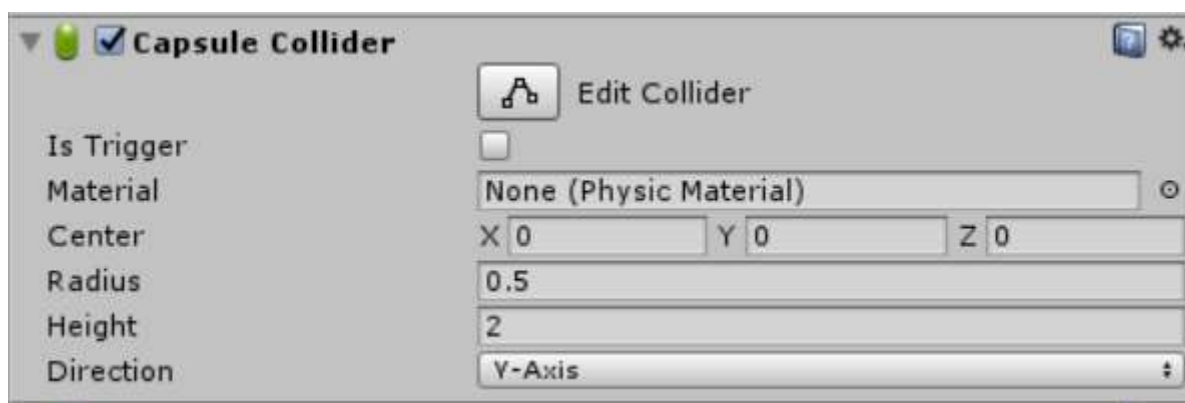


Рисунок 2.3 – Властивості **Capsule Collider**

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.

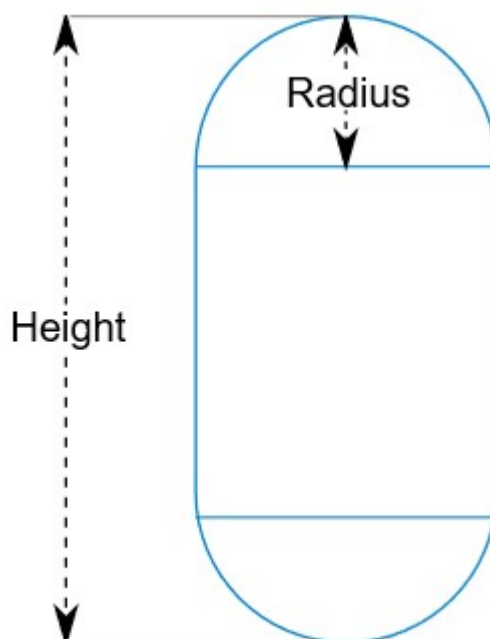


Рисунок 2.4 – Capsule Collider

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 слабе 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.

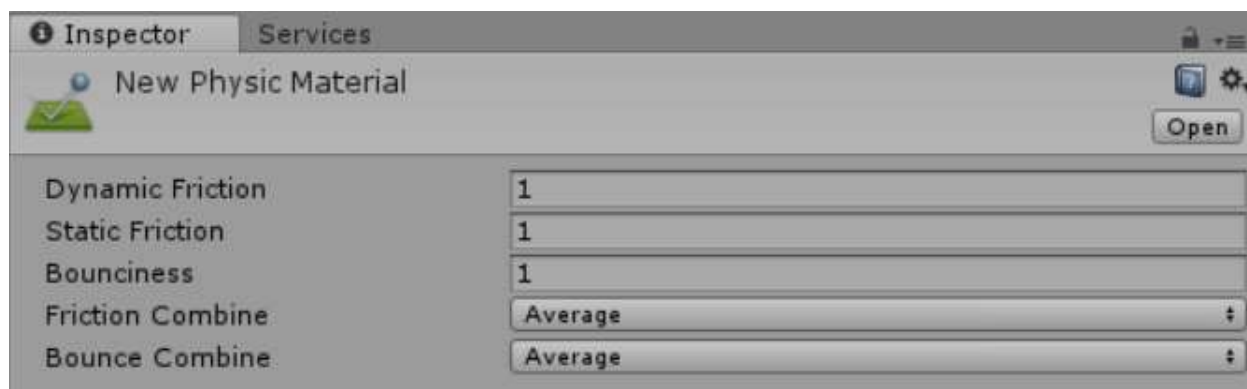


Рисунок 2.5 – Властивості **Physics Material**

Sphere Collider

Sphere Collider - базовий колайдер в формі сфери.

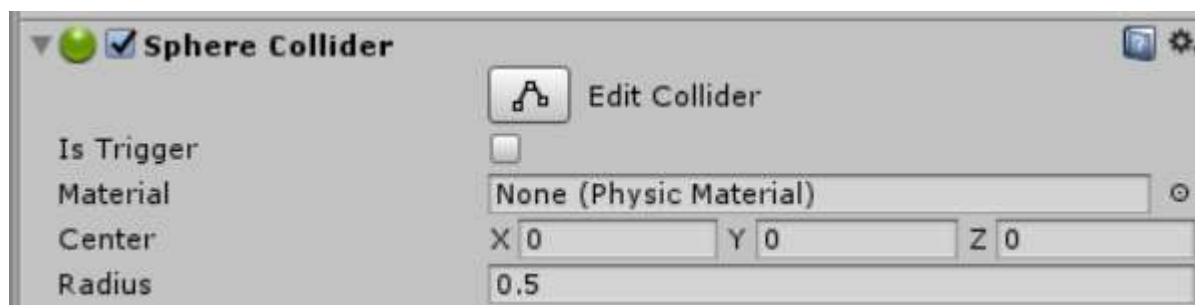


Рисунок 2.6 – Властивості **Physics Material**

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 врзатися 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 зміните 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 вмикатися/вимикатися, інакше вони будуть вести себе як статичні колайдери.

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

Навіть коли вони нерухомі, кінематичні Rigidbody колайдери поведуться 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 напрямку 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 **Character Controller**, необхідно створити C# -скрипт.

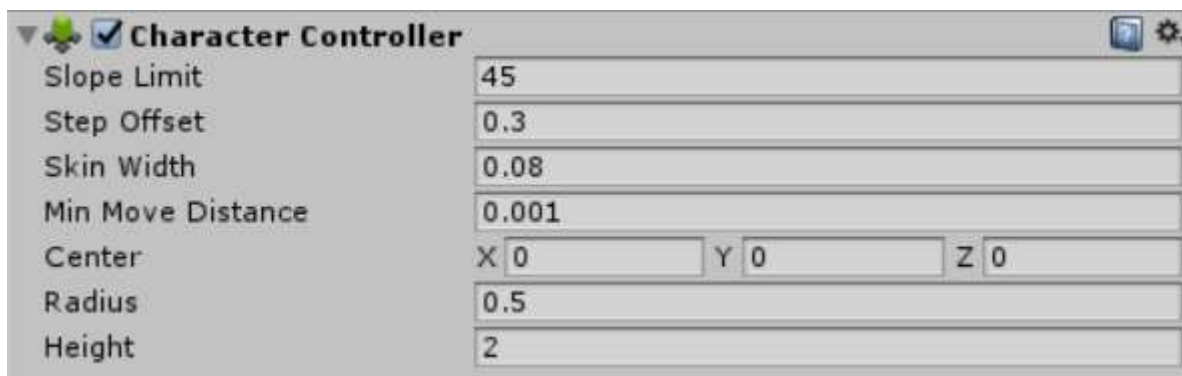


Рисунок 2.7 – Компонент **Character Controller**

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 **Rigidbody** на об'єкти, які потрібно з'єднати за допомогою **Fixed Joints**.

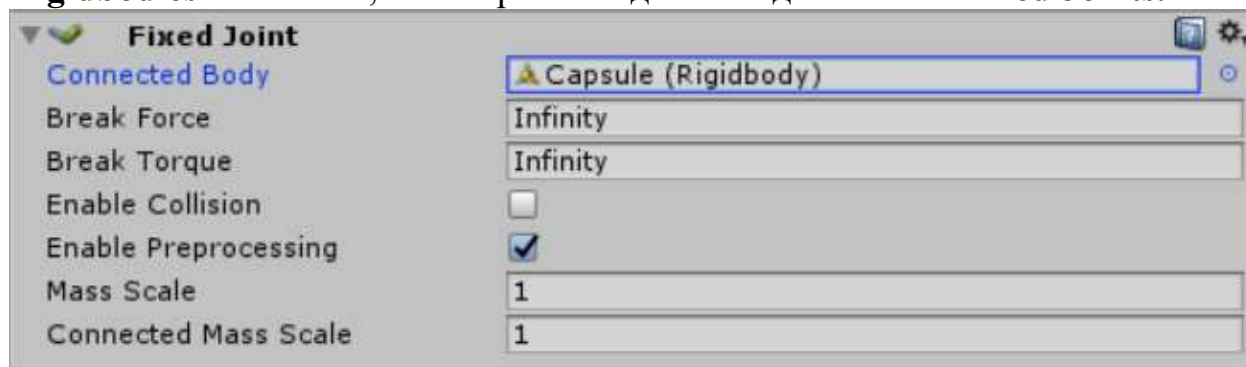


Рисунок 2.8 – Компонент **Fixed Joint**

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'єданого 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 системі Unity 3D використовуються Joint – компоненти?
 - Назвіть опції Joint - компонента, які можна включити для різних ефектів.
 - Для чого в системі 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**.

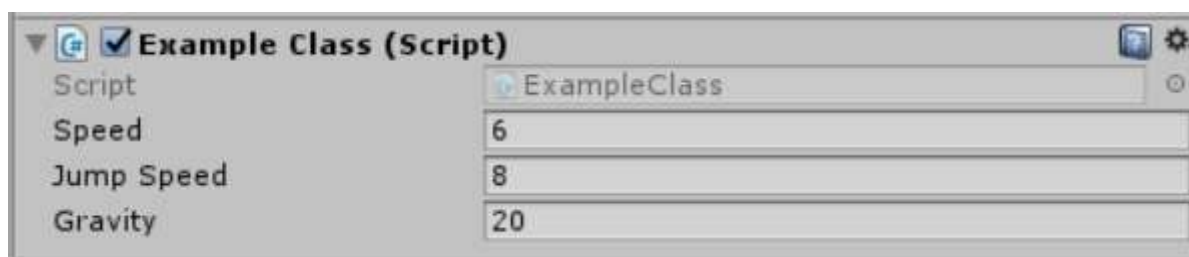


Рисунок 3.1 – Скрипт на панелі Inspector

Після приєднання скрипт почне працювати, коли ви натиснете **Play** і запустить 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**».

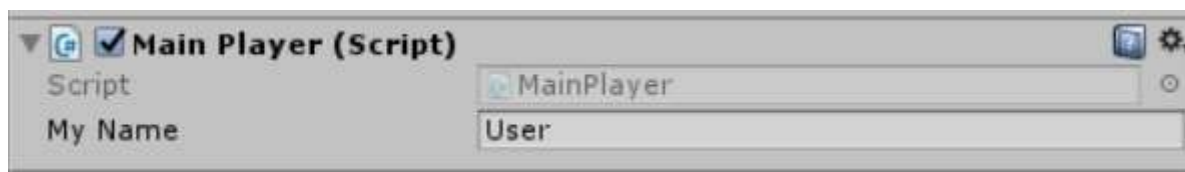


Рисунок 3.2 – Редагування змінної за допомогою панелі Inspector

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.

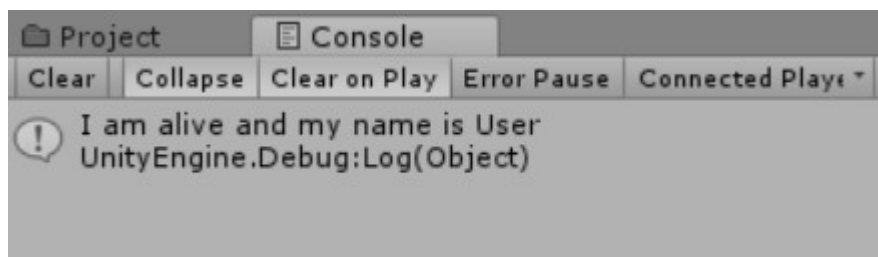


Рисунок 3.3 – Повідомлення в вікні **Console**

Unity дозволяє змінювати значення змінних скрипта в запущеній грі. Це дуже корисно і надає можливість подивитися всі ефекти від змін відразу ж, без зупинки і перезавпуску. 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 кожен 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**:

```
GameObject 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).

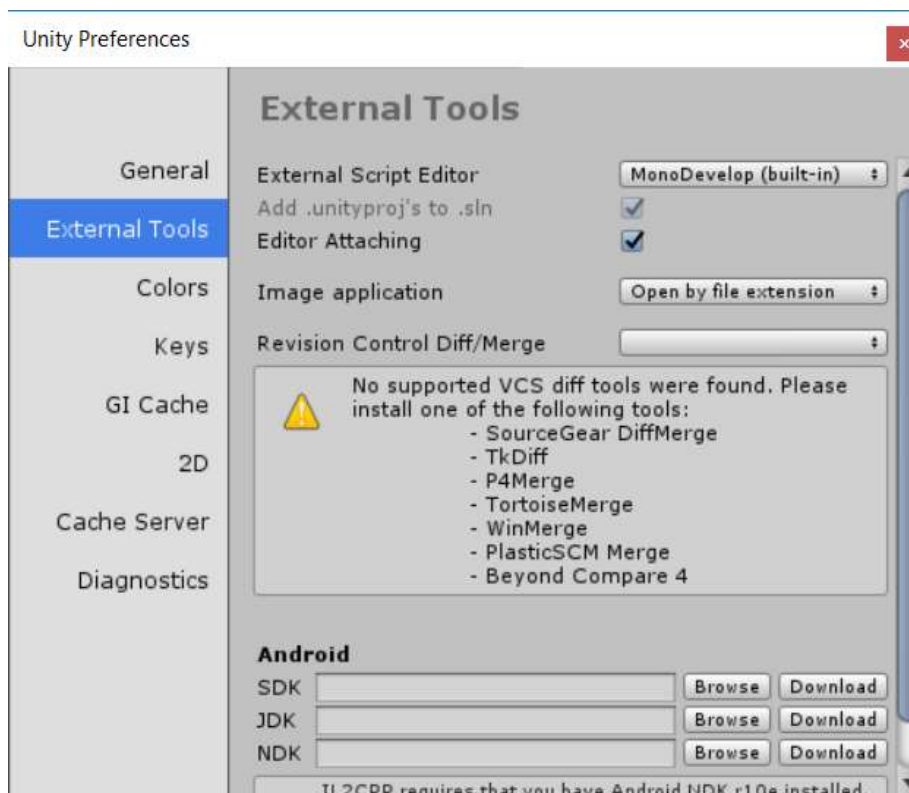


Рисунок 3.5 - Вкладка External Tools

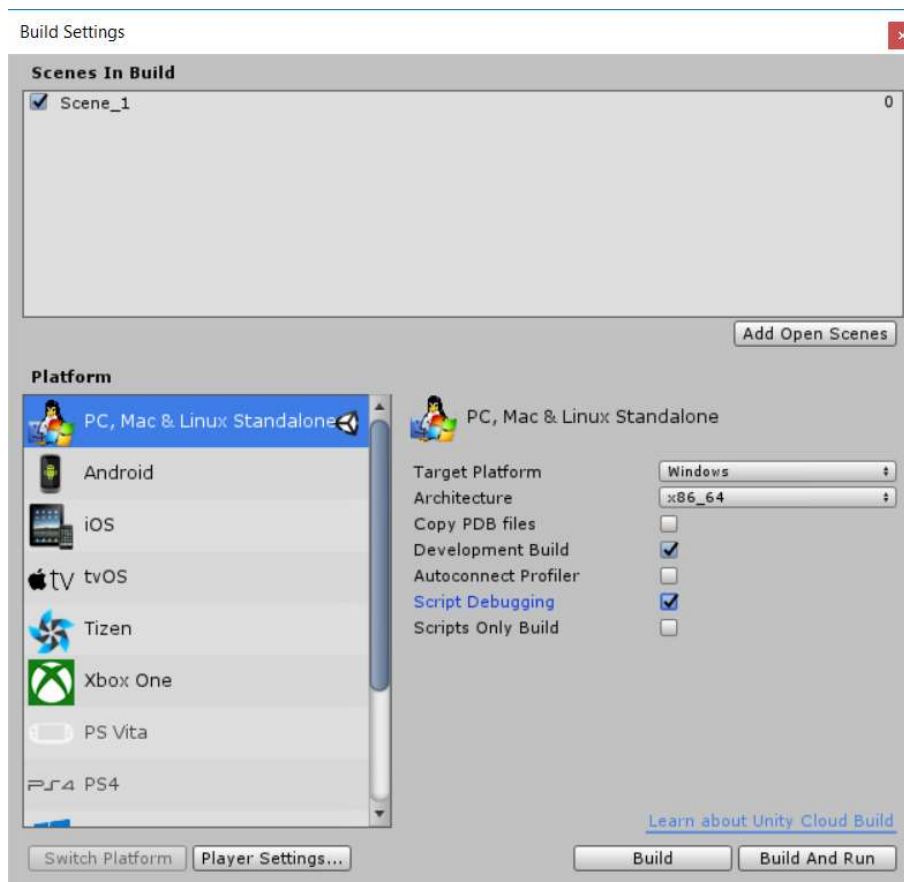


Рисунок 3.6 - Вікно BuildSettings

Створення проекту в MonoDevelop

Для створення нового проекту в меню «**F**ile» вибираємо пункт «**N**ew→**W**orkspace». Відкриється майстер проекту (рис. 3.7).

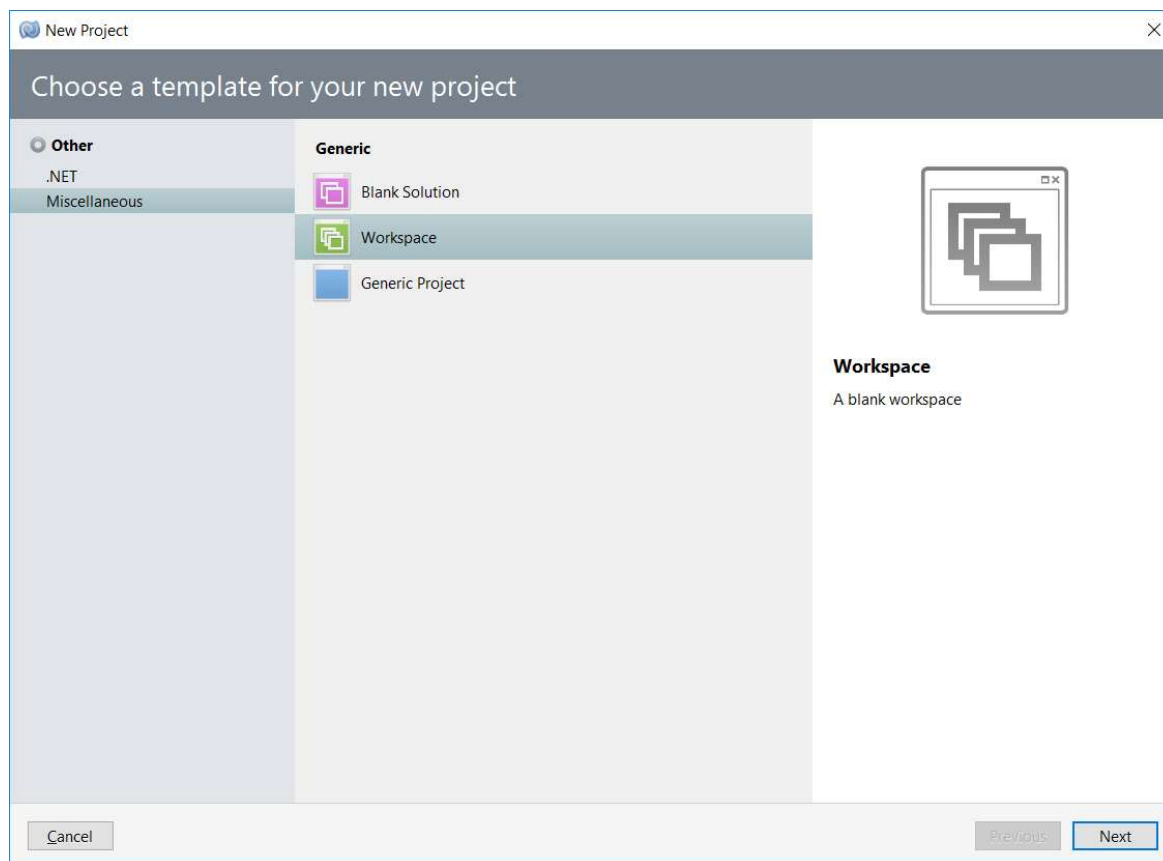


Рисунок 3.7 - Майстер проекту

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).

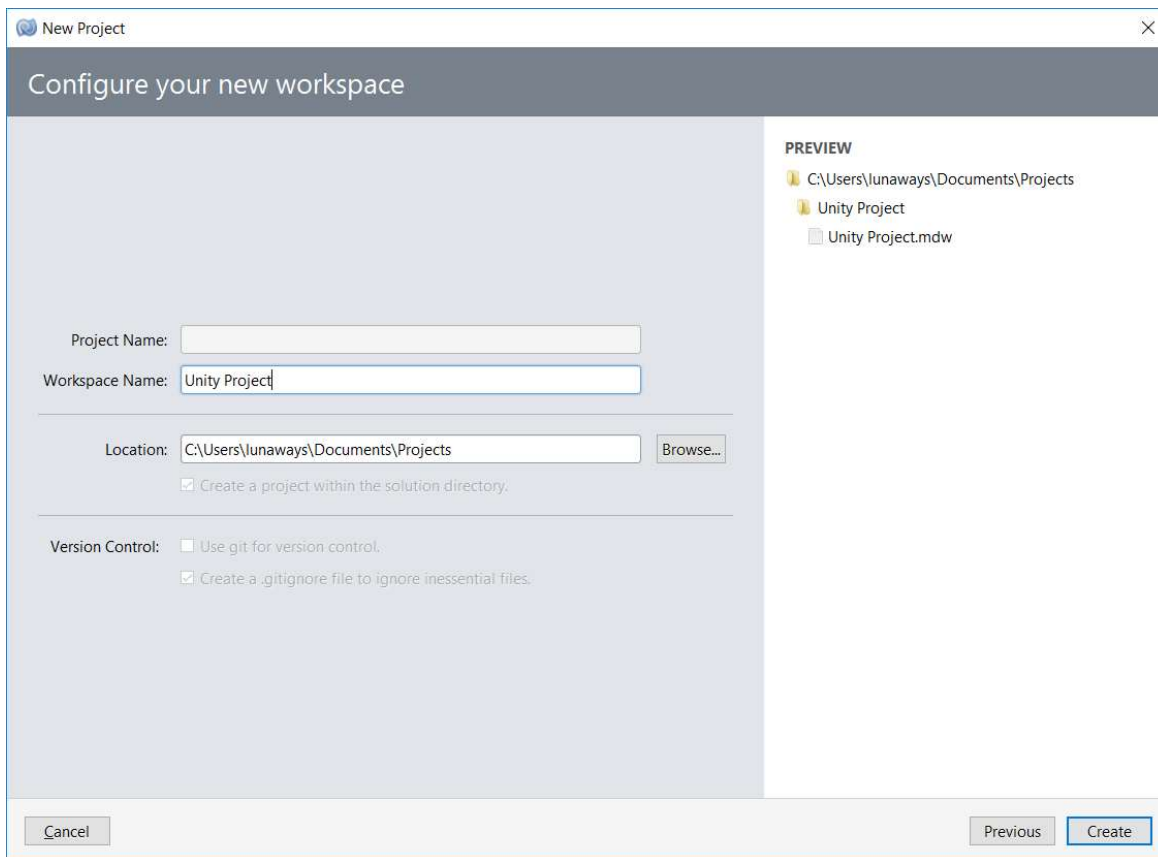


Рисунок 3.8 - Створення проекту

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

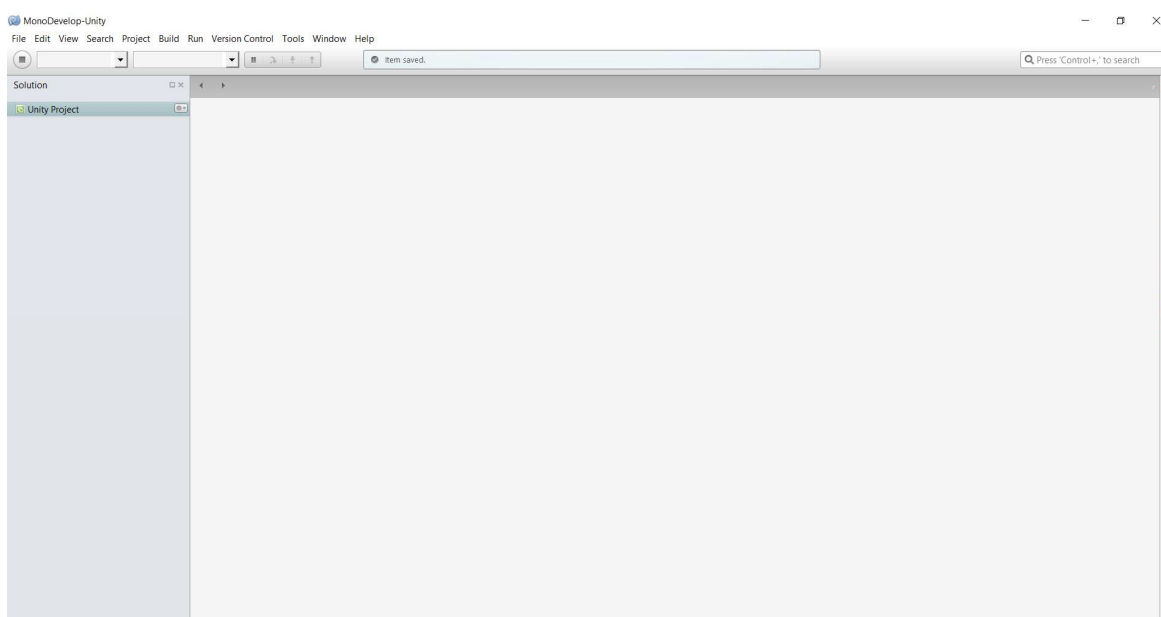


Рисунок 3.9 - Папка проекту

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

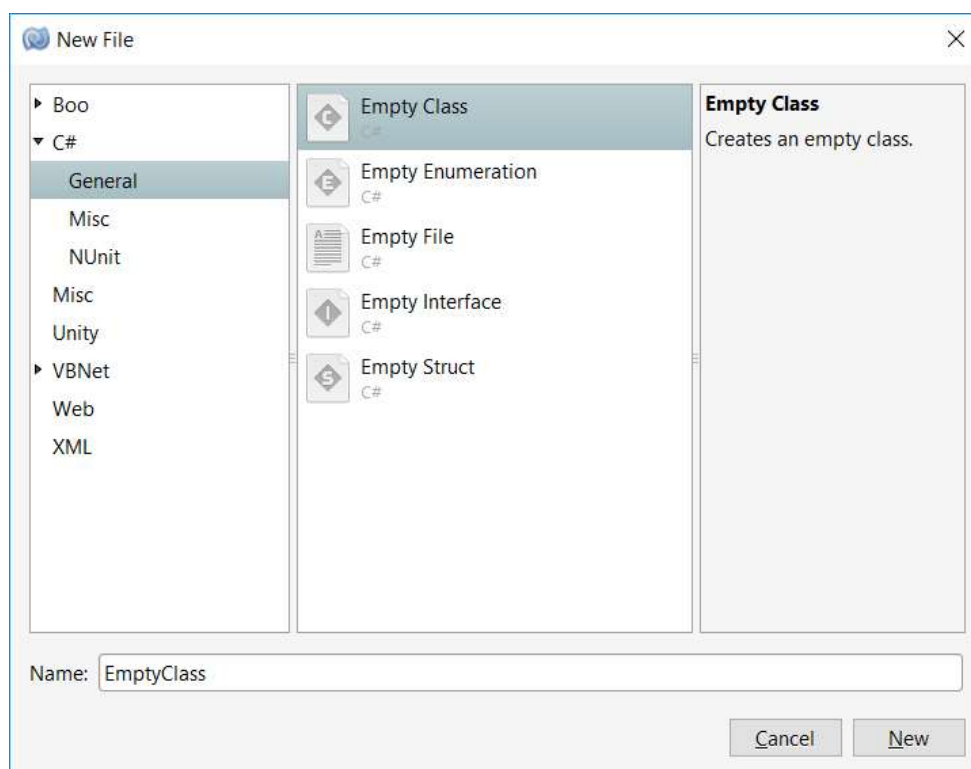


Рисунок 3.10 - Створення нового C# - файлу

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 райкастингом (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 користувацького 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 користувацького 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 **IEventHandler** за допомогою якого здійснюється **Message**.

```
public interface ICustomMessageTarget : IEventHandler
{
    // 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-кодovu 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 мультимедіаконтейнера, який є основним файловим і потоковим форматом для мультимедіакодеків 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, налаштовується) 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 мультіканальні 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** поводить себе 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.

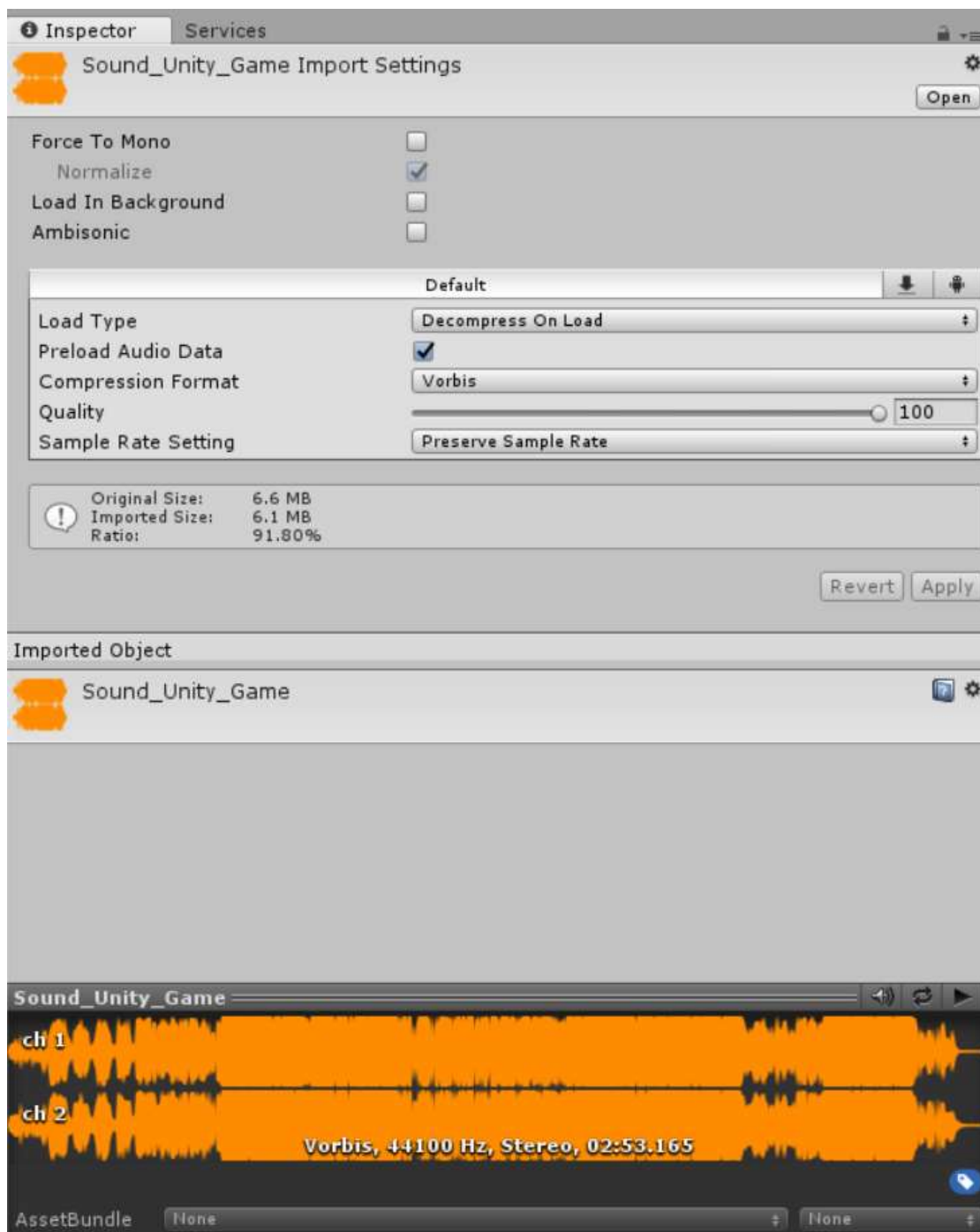


Рисунок 4.1 – Імпорт аудіофайлу

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**.

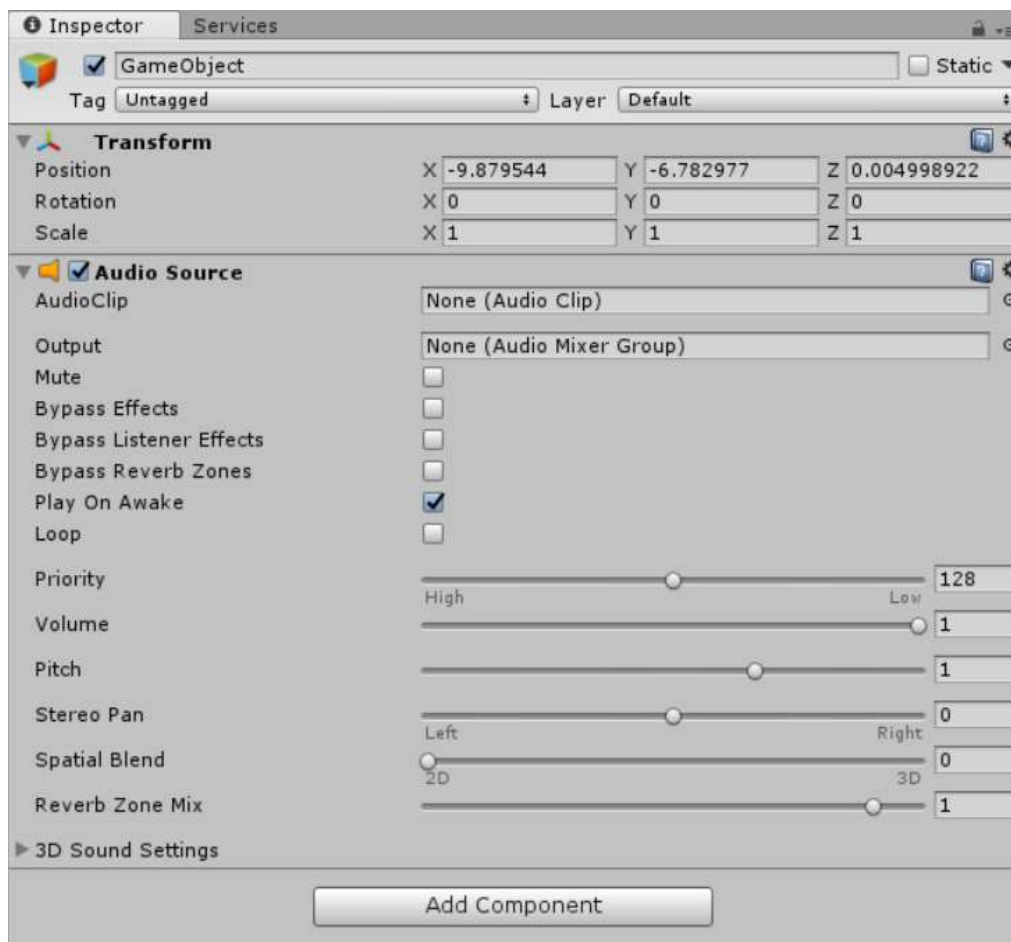


Рисунок 4.2 – Властивості компоненту **Audio Source**

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**».

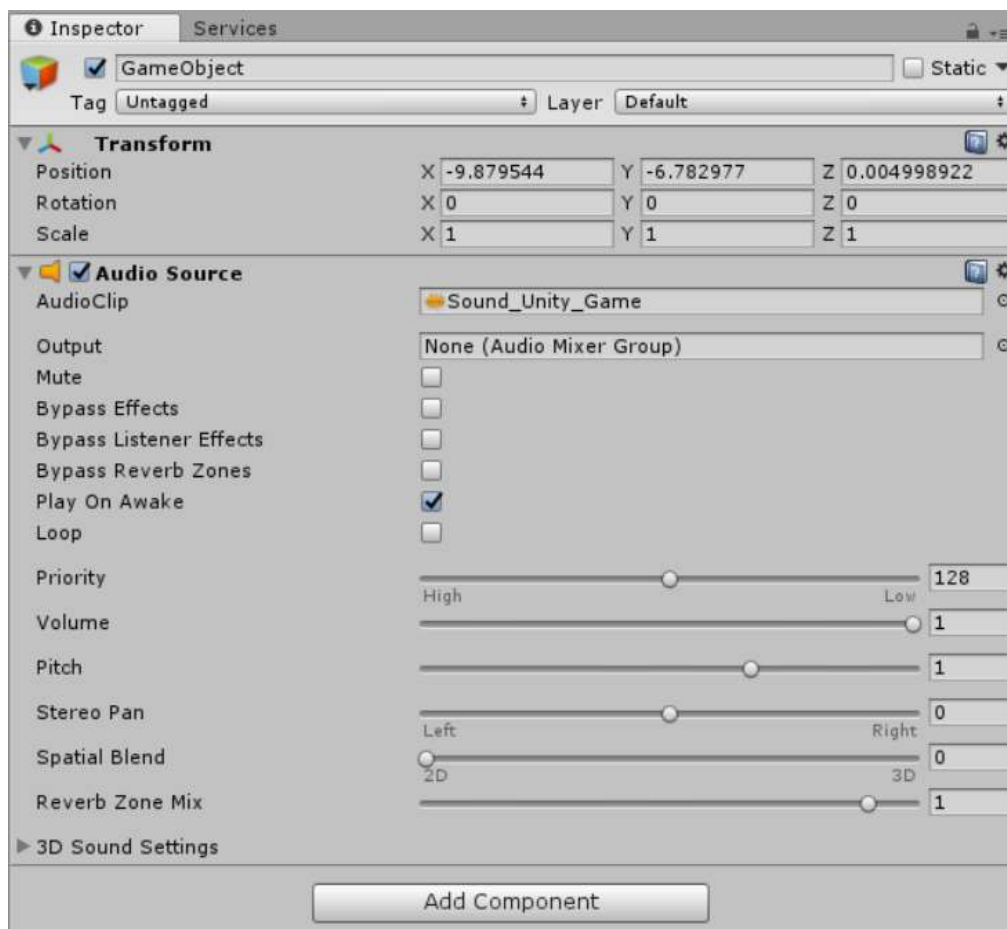


Рисунок 4.3 – Властивість **AudioClip** компонента **Audio Source**

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 чути 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.

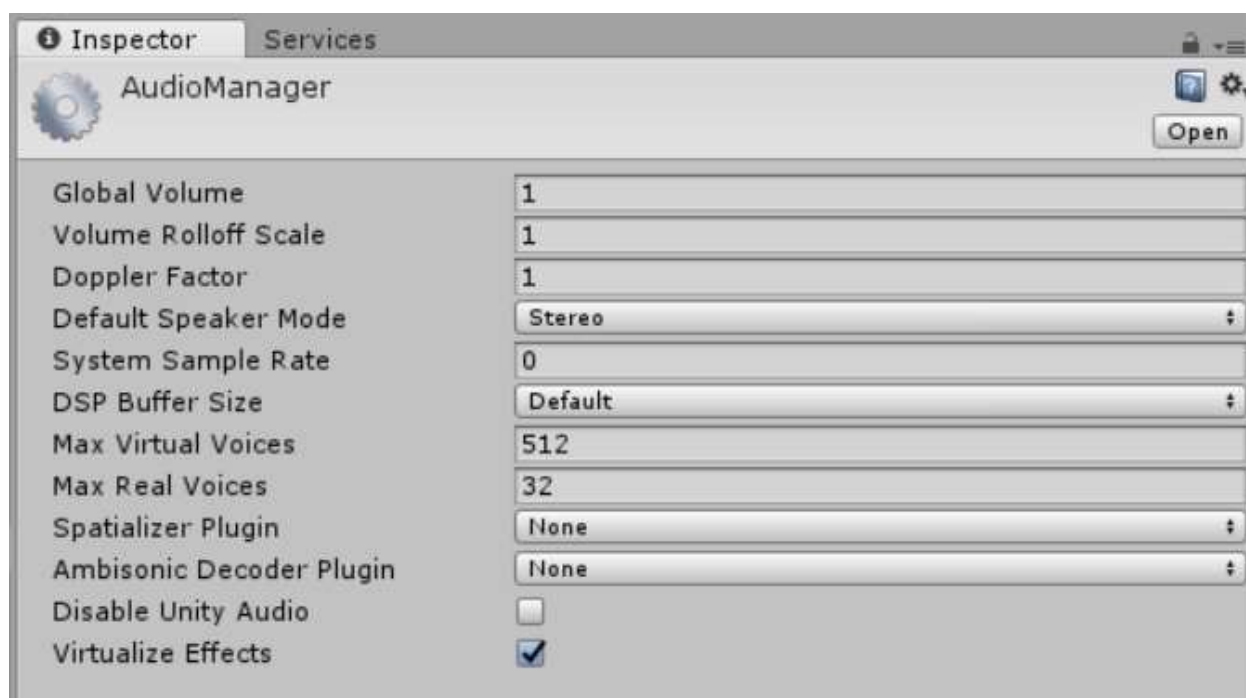


Рисунок 4.4 – Диспетчер **AudioManager**

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.

Активация звукових ефектів з коду

Ви можете програвати обраний аудіокліп, використовуючи **Play**, **Pause** і **Stop**. Ви також можете налаштувати 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 іншій нематеріальній 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 іншій нематеріальній 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**».

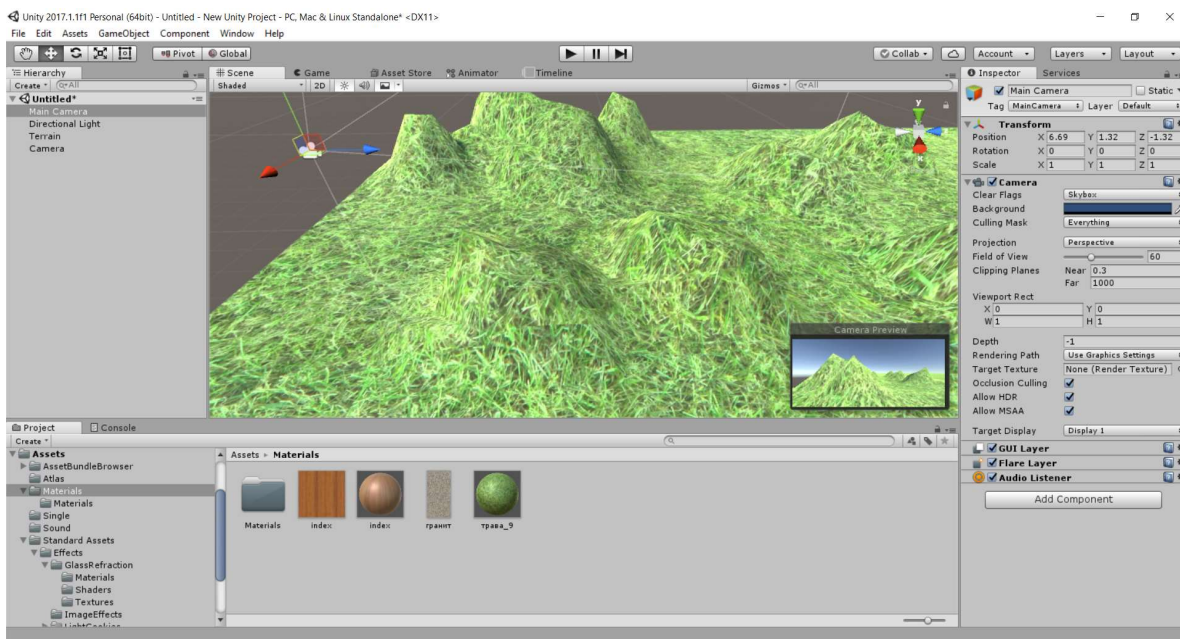


Рисунок 5.1 – Створення ландшафту та камери

Щоб відкрити вікно анімації для створення ключових кадрів анімації скористайтеся меню **Window** → **Animation** (або натисніть комбінацію клавіш **Ctrl + 6** на клавіатурі).

Для створення ключових кадрів анімації об'єкта гри виберіть об'єкт на сцені або в панелі ієрархії. Потім натисніть на кнопку **Add Property** у вікні анімації. Для створення анімованої літаючої камери виберіть камеру сцени, а потім натисніть **Add Property** у вікні анімації. Далі **Unity** виведе діалогове вікно для зазначення місця розташування в папці проекту даних анімації (кліпу анімації), завдання його імені і підтвердження. Задайте ім'я анімації **Animation.anim**.

Після цього **Unity** створить два активи: актив анімаційного кліпу (який містить дані всіх ключові кадрів) і актив контролера анімації (**Animation Controller**), який є прив'язкою до системи **Mecanim**.

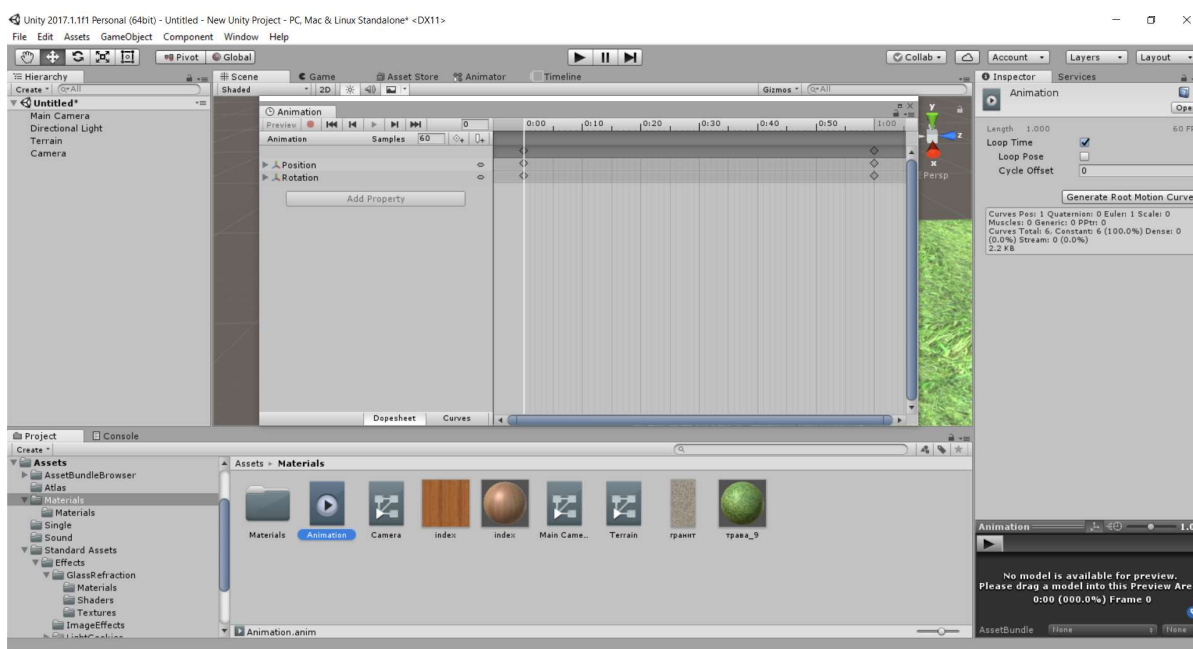


Рисунок 5.2 – Створення кліпу анімації **Animation.anim**

Щоб анімувати політ камери, натисніть на кнопку **Add Property** у вікні анімації і з контекстного меню виберіть канали для анімації. Зокрема, натисніть кнопку «+» і перейдіть до **Transform**. Потім ви можете перейти або до каналу положення (**Position**), або до каналу обертання (**Rotation**). При цьому у вікні анімації будуть додані два канали (положення і обертання), які тепер можуть бути анімовані.

Канал - це просто властивість, значення яке ми можемо прив'язати до ключових кадрів.

За замовчуванням для обох каналів положення і повороту автоматично генеруються два ключових кадра: один кадр на початку анімації (час: 0 секунд) і один в кінці (час: 1 секунда).

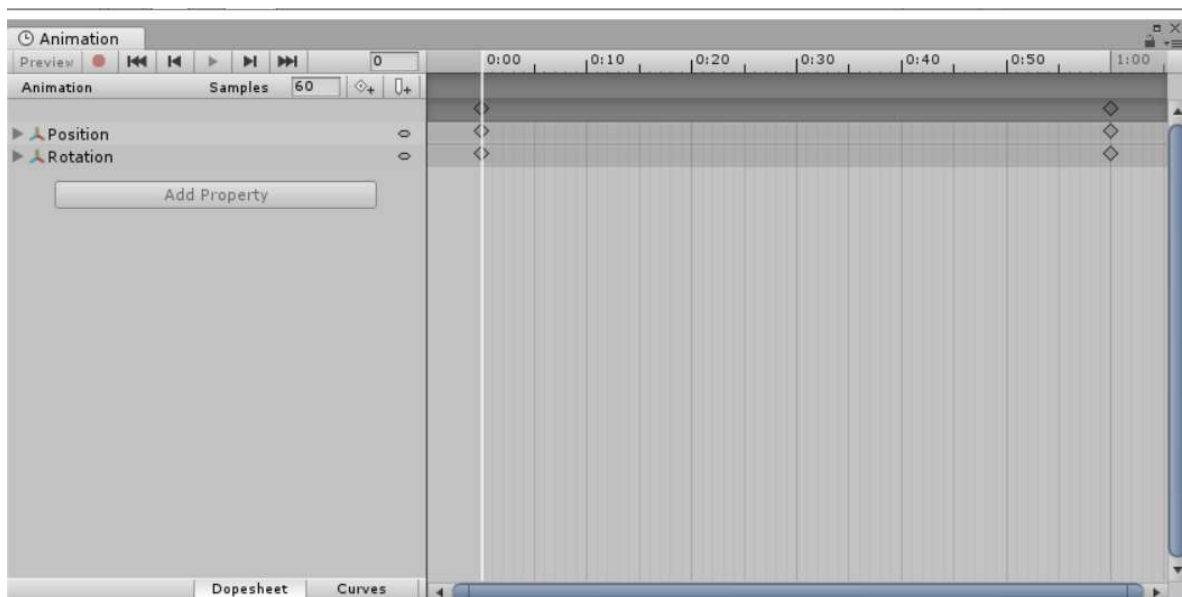


Рисунок 5.3 – Створення каналів анімації

Можна вручну створити ключовий кадр для запису стану будь-якого каналу (або всіх каналів) для камери. Для цього необхідно перетягнути ромбовидні значки ключових кадрів на шкалі часу, змінюючи для них час. Також можна продублювати їх за допомогою комбінації клавіш **Ctrl + C** і **Ctrl + V** і видалити вибрані ключові кадри за допомогою клавіш **Backspace** або **Delete**.

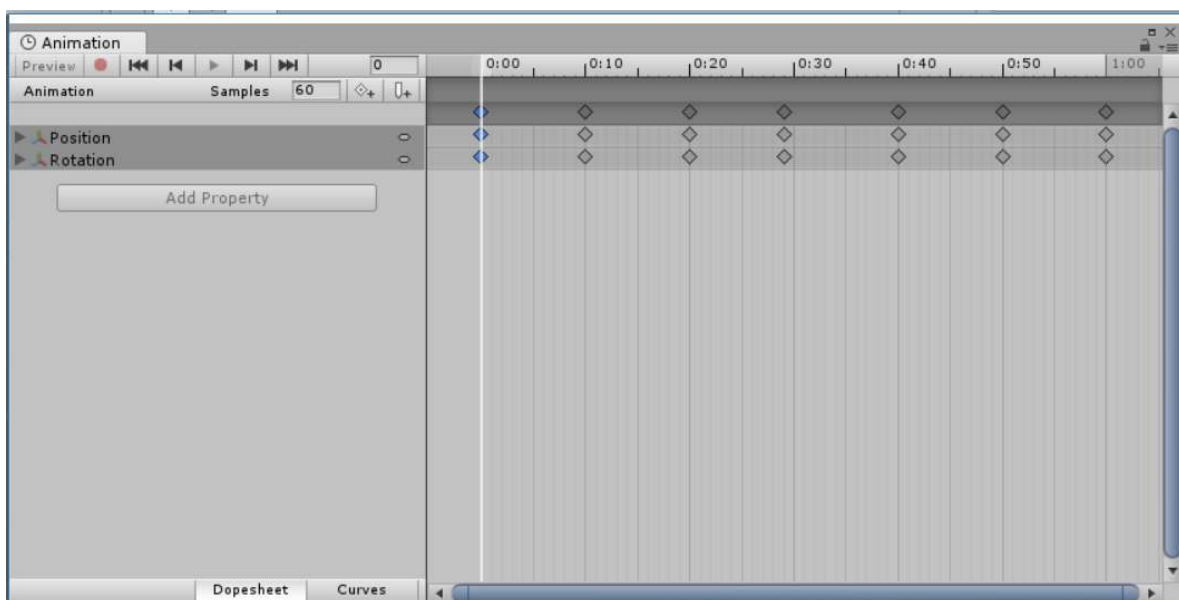


Рисунок 5.4 – Створення ключових кадрів анімації

В Unity існує можливість генерувати ключові кадри автоматично, ґрунтуючись на рухах об'єктів в сцені.

Для цього необхідно помістити повзунок анімації у вікні анімації в положення 0, а потім повернути об'єкт камери в сцені в її бажане вихідне положення і орієнтацію. Unity автоматично запише положення і поворот

камери для каналів на обраний вами момент часу. Повторіть цей процес генерації ключів кадрів, що визначають положення і поворот для камери, для кожного з моментів часу: 0:10, 0:20, 0:30, 0:40, 0:50, 1:00.

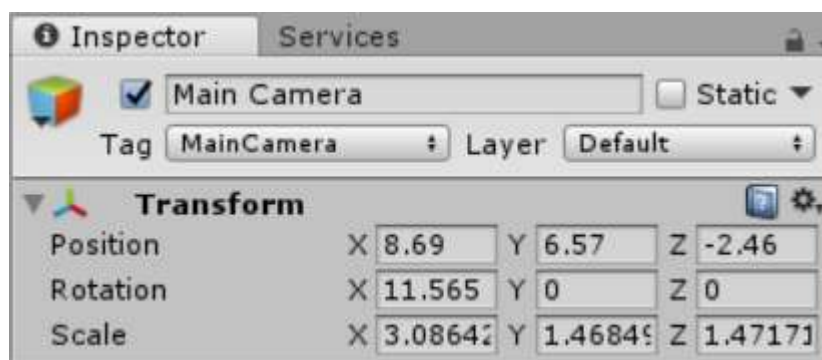


Рисунок 5.5 – Положення і поворот камери для каналів на обраний момент часу

Unity має широкий діапазон попередньо підготовлених систем частинок для створення найбільш поширених ефектів, таких як вибух, вогонь, пар або дим. Для отримання доступу до цих ефектів їх необхідно імпортувати до Unity-проекту за допомогою меню **Assets → Import Package → Particle Systems**.



Рисунок 5.6 – Діапазон попередньо підготовлених систем частинок

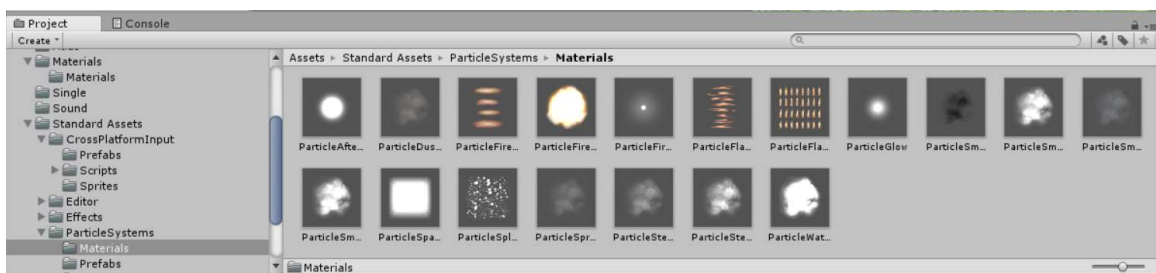


Рисунок 5.7 – Імпорт найбільш поширених ефектів

Всі системи частинок мають три спільні риси. По-перше, це наявність емітера (**emitter**), який, випускає частинки. По-друге, самі частинки - дрібні предмети, які генеруються емітером. По-третє, частинки мають термін служби – період часу, протягом якого вони існують, а також траєкторію або швидкість, яка визначає, що станеться з частинками протягом їх терміну служби (як вони будуть рухатися, як швидко переміщатися і т.д.).

Для роботи з частинками в Unity використовується система частинок Сюрікен (**Shuriken particle system**).

Створимо нову систему частинок в ігровій сцені. Для цього скористаємося меню **Game Object → Effects → Particle System**.

Після вибору меню система частинок автоматично почне функціонувати в вікні сцени.

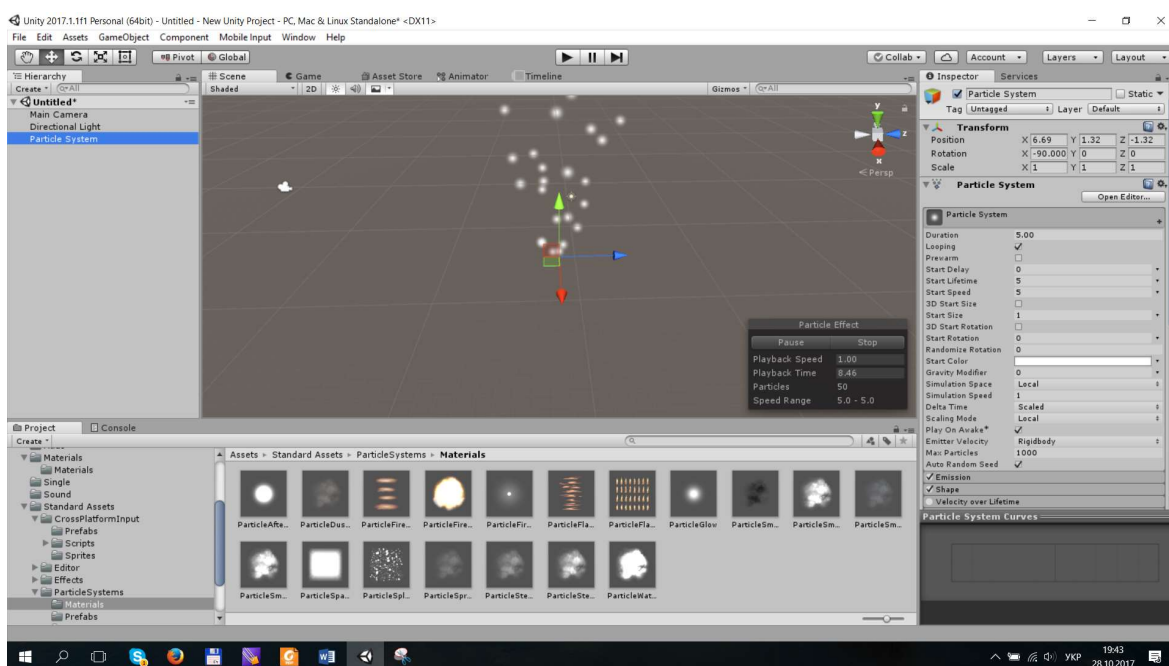


Рисунок 5.8 – Автоматичне функціонування системи частинок

Властивості системи частинок доступні на панелі «**Inspector**».



Рисунок 5.9 – Властивості системи частинок

Компонент **Particle System** має безліч налаштувань і для зручності, інспектор впорядковує їх у секції або «модулі». Кожна секція може бути розгорнута і згорнута при натисканні на заголовок. З лівого боку розташовані чекбокси, які можна відзначати для включення або відключення тих чи інших налаштувань. Наприклад, якщо ви не хочете змінювати розмір часток під час їх випромінювання, ви можете просто зняти галочку з параметра «**Size over lifetime**».

Крім модулів, в інспекторі є ще кілька інших елементів управління. Кнопка **Open Editor** відображає опції в окремому вікні редактора, яке також дозволяє редагувати одночасно кілька систем. А чекбокс **Resimulate**

відповідає за те, будуть чи не будуть застосовуватися зміни властивостей до щойно створених системою частинок. Кнопка **Wireframe** відображає контури меш-об'єктів, щоб показати частинки, які знаходяться за ними.

Значення поля тривалості **Duration** визначає загальну тривалість анімації системи частинок, перш ніж вона почнеться спочатку і зациклиться.

Значення поля початкового розміру **Start Size** управляє розміром згенерованих частинок.

Кожна система частинок має **емітер** – місце відображення частинок, які генеруються. Сам емітер має форму. Ця форма може бути площиною, точкою, кубом, кулею або звичайним мешем, і вона є поверхнею або тілом, всередині якого генеруються частинки.

Для управління формою емітера використовується закладка **Shape** в **Shuriken particle system**.

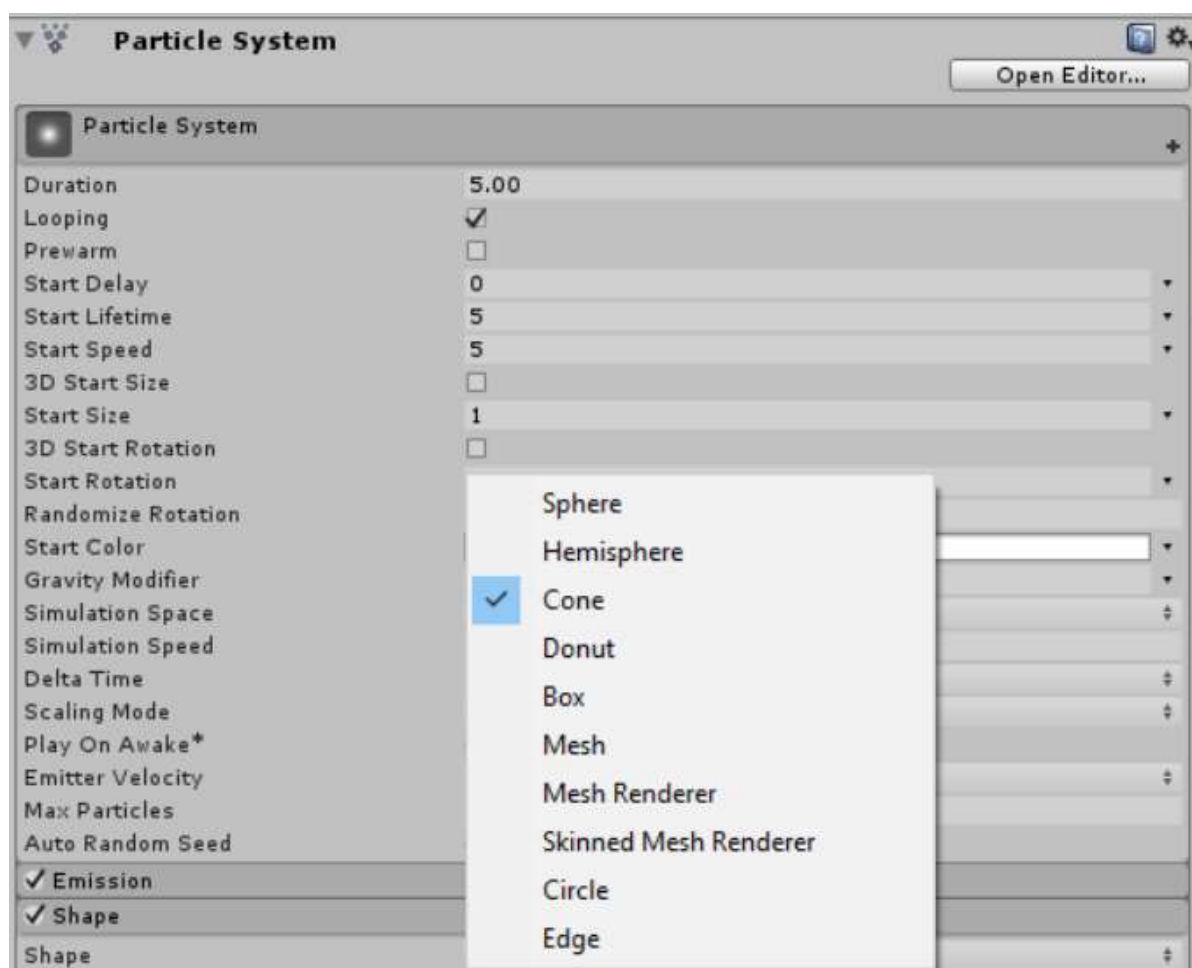


Рисунок 5.10 – Закладка **Shape**

За частоту або швидкість генерації нових частинок з емітера в секунду відповідає секція **Emission**.

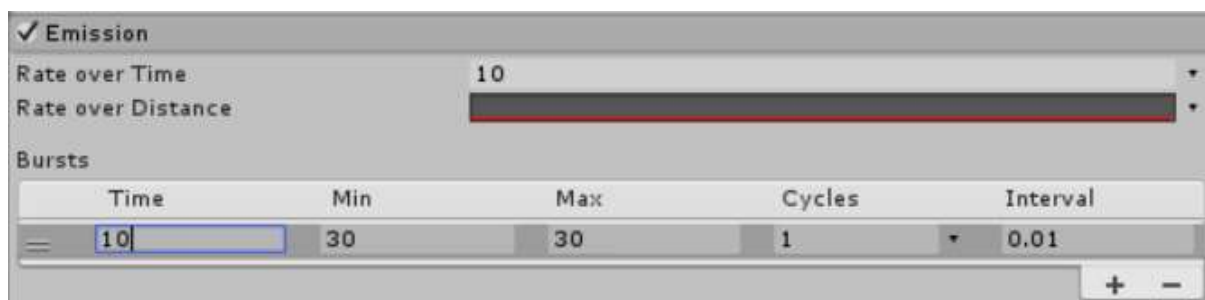


Рисунок 5.11 – Секція **Emission**

Параметри, що керують зовнішнім виглядом частинок, знаходяться в секції **Renderer**.

Поле **Render Mode** визначає, як візуалізуються частинки. Режимів всього два: режим білборда **Billboard** і режим мешів **Mesh**.

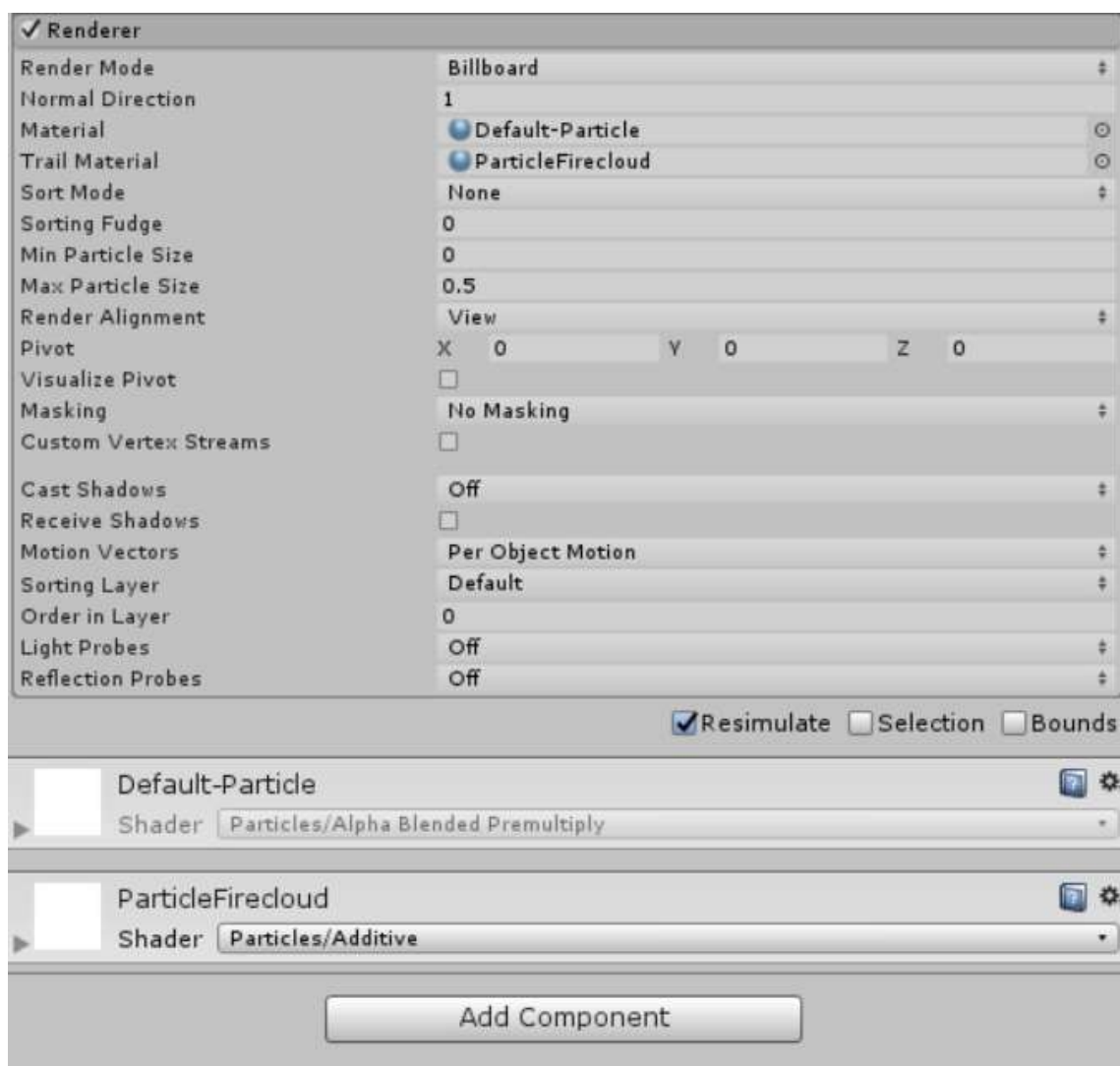


Рисунок 5.12 – Секція **Renderer**

Крім налаштування режиму рендеру, частки потребують в матеріалі, який буде призначений Мешу.

Можна використовувати попередньо підготовлені матеріали для систем частинок, що поставляються в складі активів **Unity**. Для цього необхідно перетягнути вибраний матеріал в чарунку матеріалу **Material** в інспекторі об'єктів.

Швидкість частинки в період її існування визначається в секції **Velocity over Lifetime**.

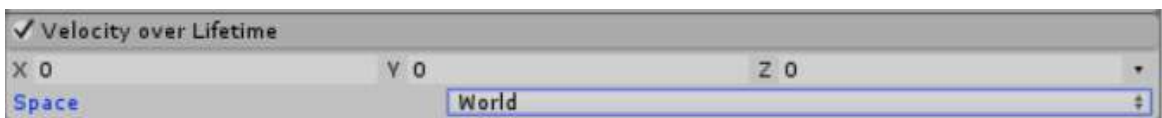


Рисунок 5.13 – Секція **Velocity over Lifetime**

Секцію зміни кольору протягом існування частинок **Color over Lifetime** можна використовувати для зміни кольору частинок в часі.

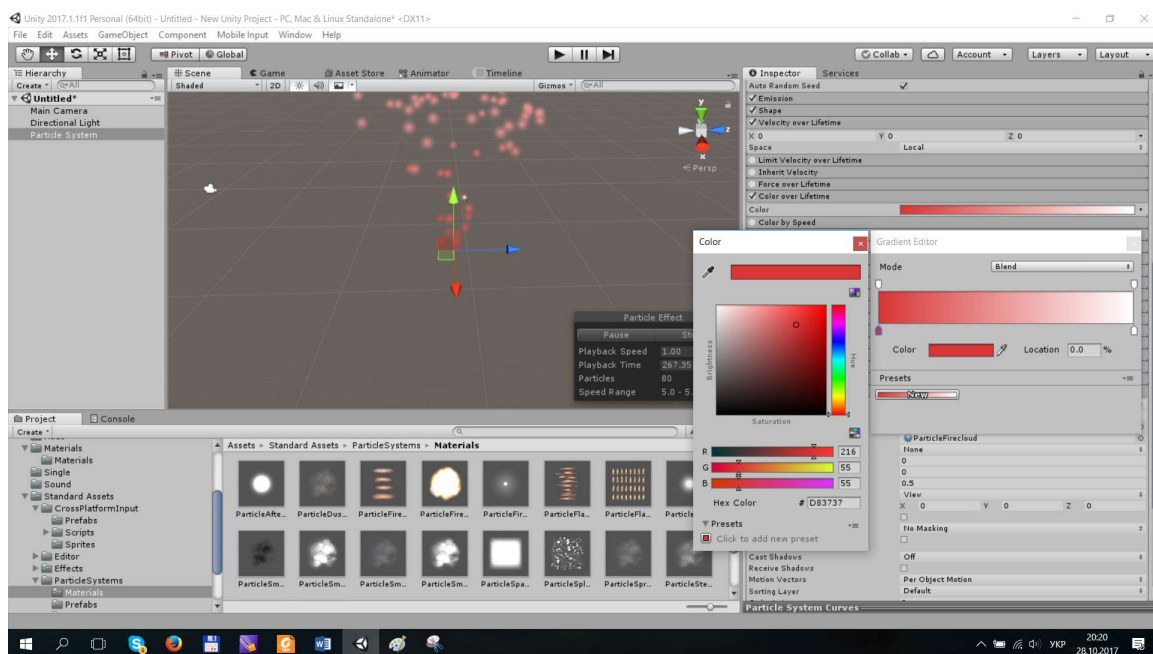


Рисунок 5.14 – Секція **Color over Lifetime**

Анімація за допомогою спрайтів

Перед початком створення анімації необхідно здійснити імпорт спрайтів. Після того як ви імпортували набір об'єктів спрайтів (не має значення, як окремі зображення або як текстуру атласу), ви готові запустити анімацію з їх участю.

На панелі проекту Unity виберіть всі спрайти, що входять в повну послідовність анімації, а потім перетягніть їх всі разом на панель ієрархії сцени. Якщо ви зробите це, з'явиться діалогове вікно збереження **Save**, пропонуючи вам створити нову послідовність анімації (файл з розширенням **.anim**). Задайте ім'я файлу анімації (наприклад, **PlayerRun.anim**) і виберіть кнопку **Save**.

У сцені буде створено об'єкт спрайту, він повинен стати видимим на вкладках сцени і гри. Якщо ви протестуєте вашу сцену, то ваш персонаж буде анімаційним, ви повинні побачити всі зображення послідовності. Зазначимо, що Unity створило новий актив Animation Clip (.anim), який визначає ключову послідовність кадрів. Крім того, Unity створило контролер Mecanim для запуску відтворення анімації при завантаженні рівня і для управління швидкістю відтворення, додало компонент аніматора до об'єкта спрайту в сцені, щоб зв'язати об'єкт з його даними по анімації.

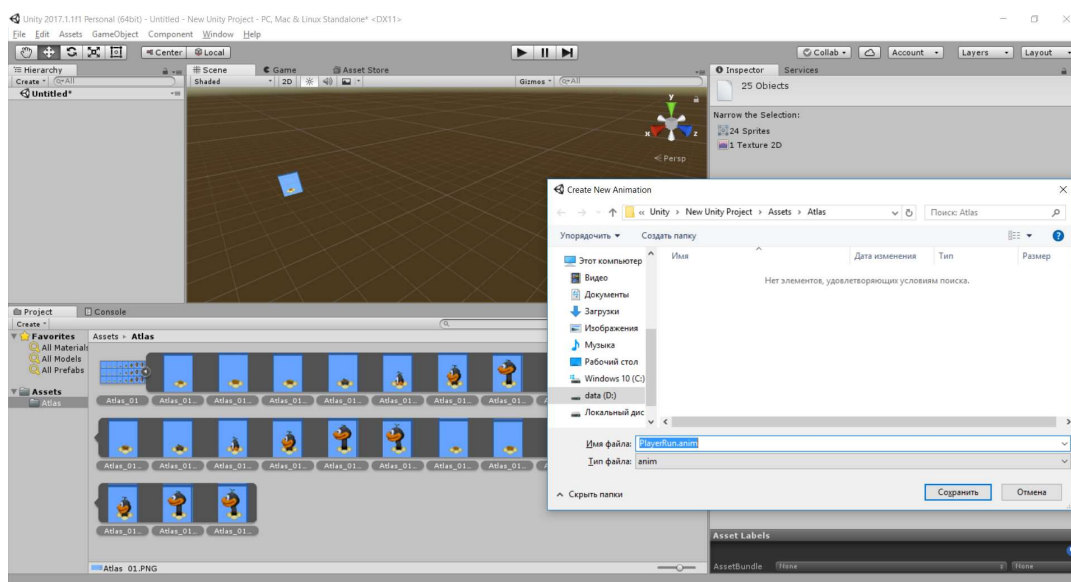


Рисунок 5.15 – Створення файлу анімації

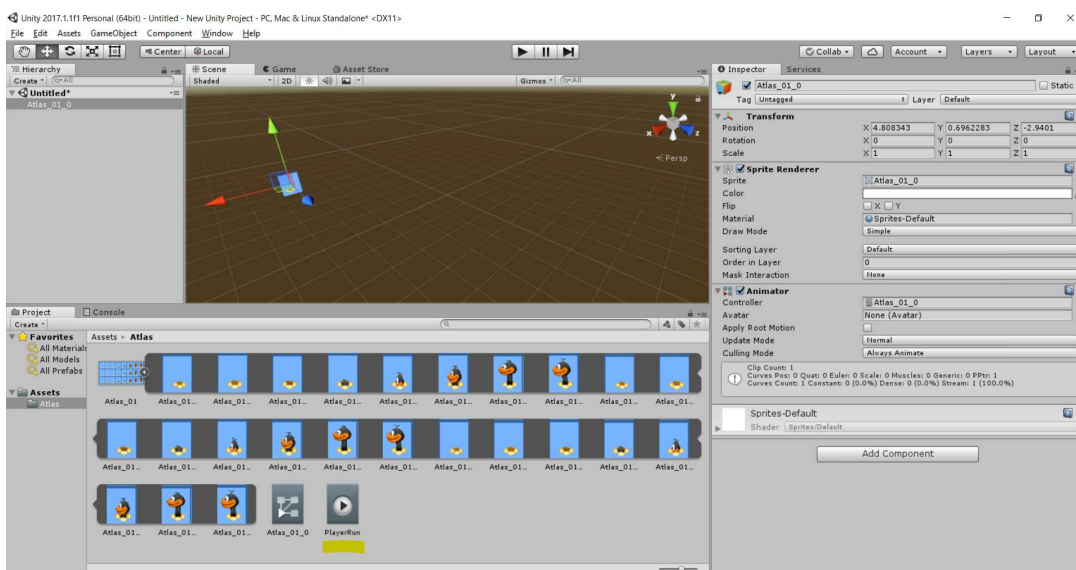


Рисунок 5.16– Створення нового активу Animation Clip (.anim) та контролеру Mecanim

Налаштування анімації за допомогою інструменту Mecanim

Якщо анімація спрайтами відтворюється занадто швидко або занадто повільно, то ви повинні будете відредагувати діаграму спрайту інструменту Mecanim.

Для досягнення цієї мети виберіть об'єкт спрайту в сцені. В інспекторі об'єктів двічі клацніть на актив **Animation Controller** всередині слота **Controller** компонента **Animator**.

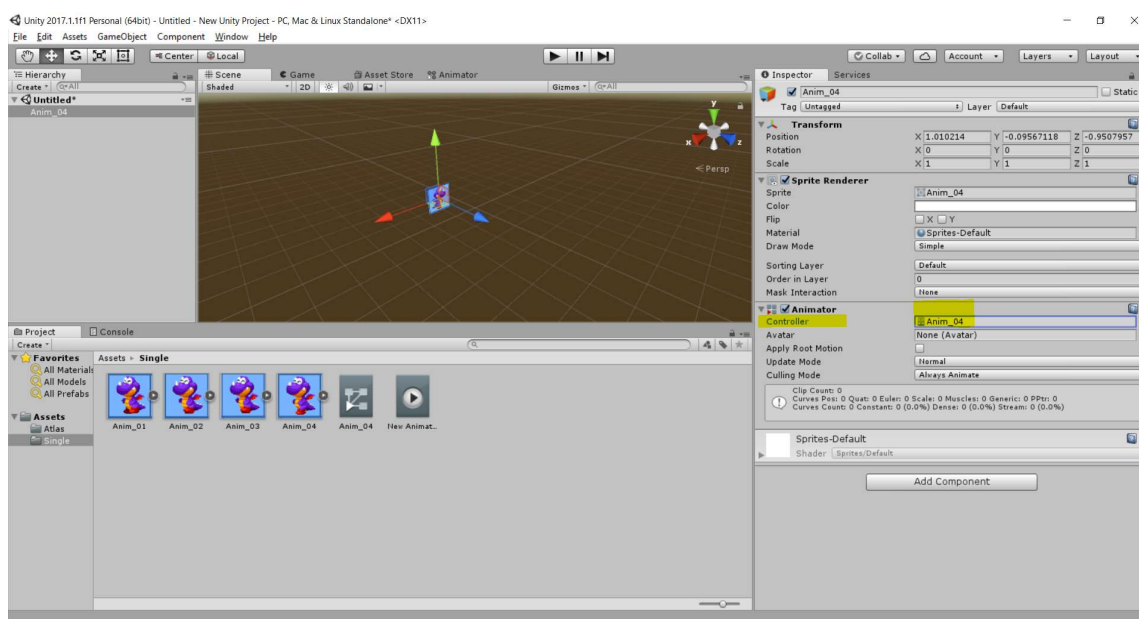


Рисунок 5.17– Налаштування анімації

Подвійний вибір активу **Animation Controller** спрайту викликає діаграму інструменту **Mecanim** для спрайту. З неї можна керувати швидкістю анімації. Ця діаграма містить кілька вузлів, пов'язаних між собою.

У діаграмі натисніть на вузол **Anim_04**, який представляє в діаграмі спрайт анімації, щоб вибрати його і переглянути його властивості в інспекторі об'єктів.

Швидкість анімації регулюється за допомогою параметра **Speed**. Значення 0 - зупинка, значення 1 - швидкість за замовчуванням, значення 0,5 - половинна швидкість, 2 - подвійна швидкість і т.д. Якщо анімація занадто повільна, то збільште значення швидкості, і якщо вона занадто швидка, то зменште значення швидкості. Після того як ви закінчите, просто запустіть гру, щоб побачити ефект.

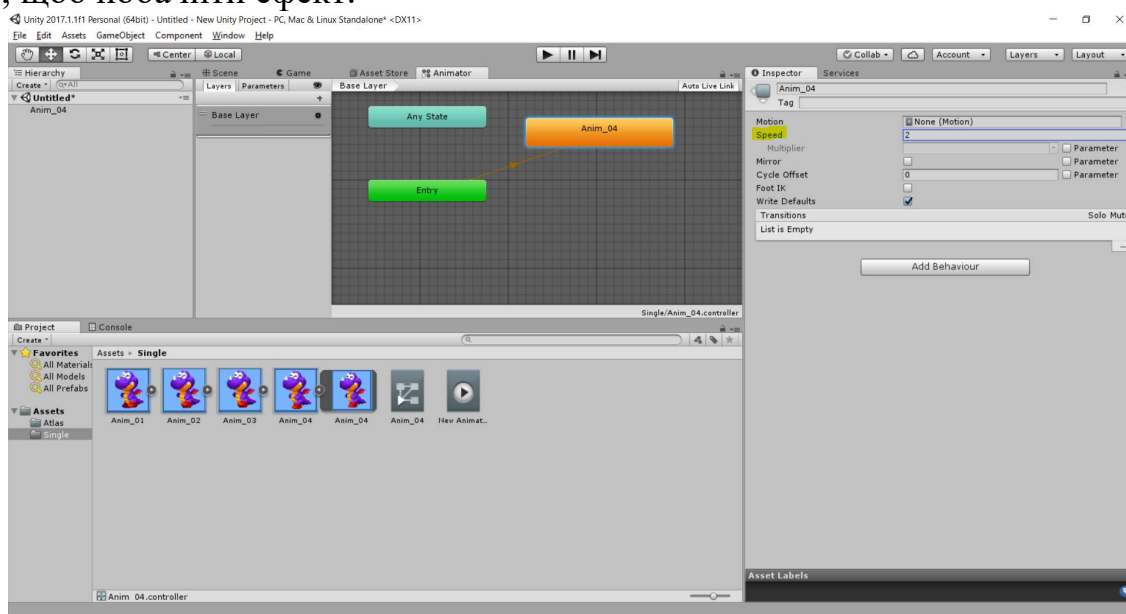


Рисунок 5.18– Діаграма інструменту **Mecanim**

За замовчуванням спрайт анімації зациклений, тобто відтворюється знову і знову без кінця. Коли відтворення анімації завершує цикл, воно просто повертається до початку, і відбувається нове відтворення.

Іноді потрібно відтворити анімацію тільки один раз, а потім зупинити. Для цього потрібно отримати доступ до даних анімації (вони знаходяться всередині активу .anim) і відкоригувати їх властивості. Щоб зробити це, виберіть актив спрайту анімації в панелі проекту. Активи анімації помічені іконкою **Play** і мають ім'я, яке ви присвоїли їм при їх створенні. Після вибору активу скиньте прапорець **Loop Time** в інспекторі об'єктів. Тепер анімацію буде відтворено тільки один раз.

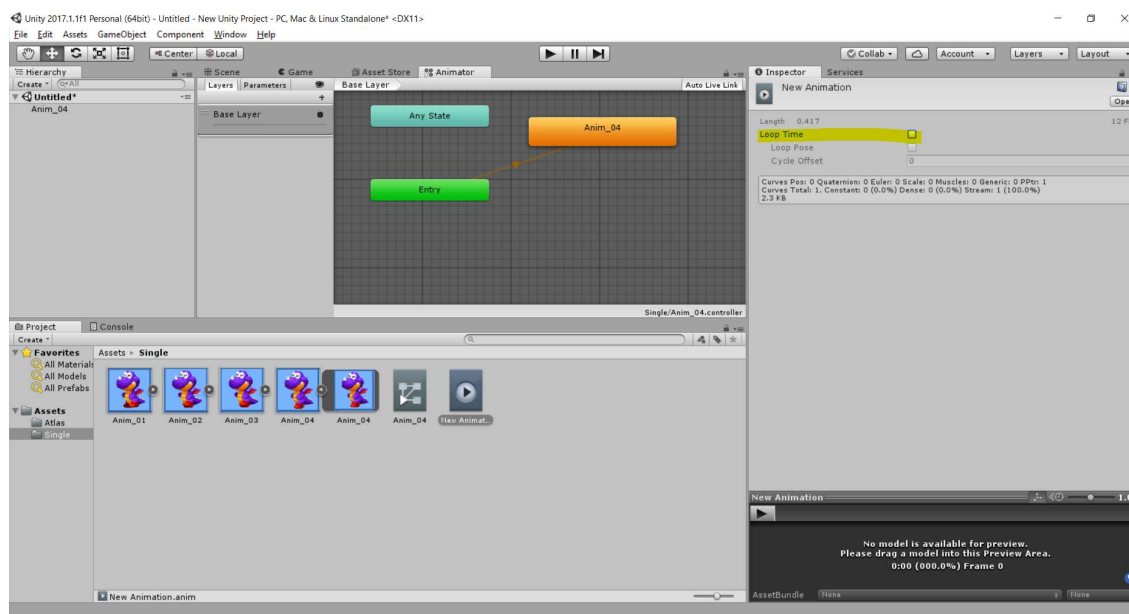


Рисунок 5.19– Налаштування анімації. Властивість **Loop Time**

Редагування кадрів анімації. Вікно **Animation**.

Якщо спрайт анімації має багато кадрів, то, можливо, при генерації анімації вони будуть розташовані в неправильному порядку, в результаті чого деякі кадри стануть з'являтися раніше або пізніше потрібного моменту.

Редагувати кадри анімації можна у вікні **Animation**. Це вікно можна отримати, перейшовши до **Window** → **Animation** в головному меню.

При відкритому вікні **Animation** виберіть об'єкт спрайту в сцені, і його анімаційні дані автоматично будуть відображені на часовій шкалі. Графік містить в собі весь період анімації, від початку і до кінця. Ромбовидні символи, рівномірно розподілені по шкалі, є ключові кадри, в яких змінюються зображення спрайтів. Ви можете вибрати конкретний ключовий кадр на часовій шкалі, обравши його. Потім ви можете переглянути його спрайтові властивості в інспекторі об'єктів.

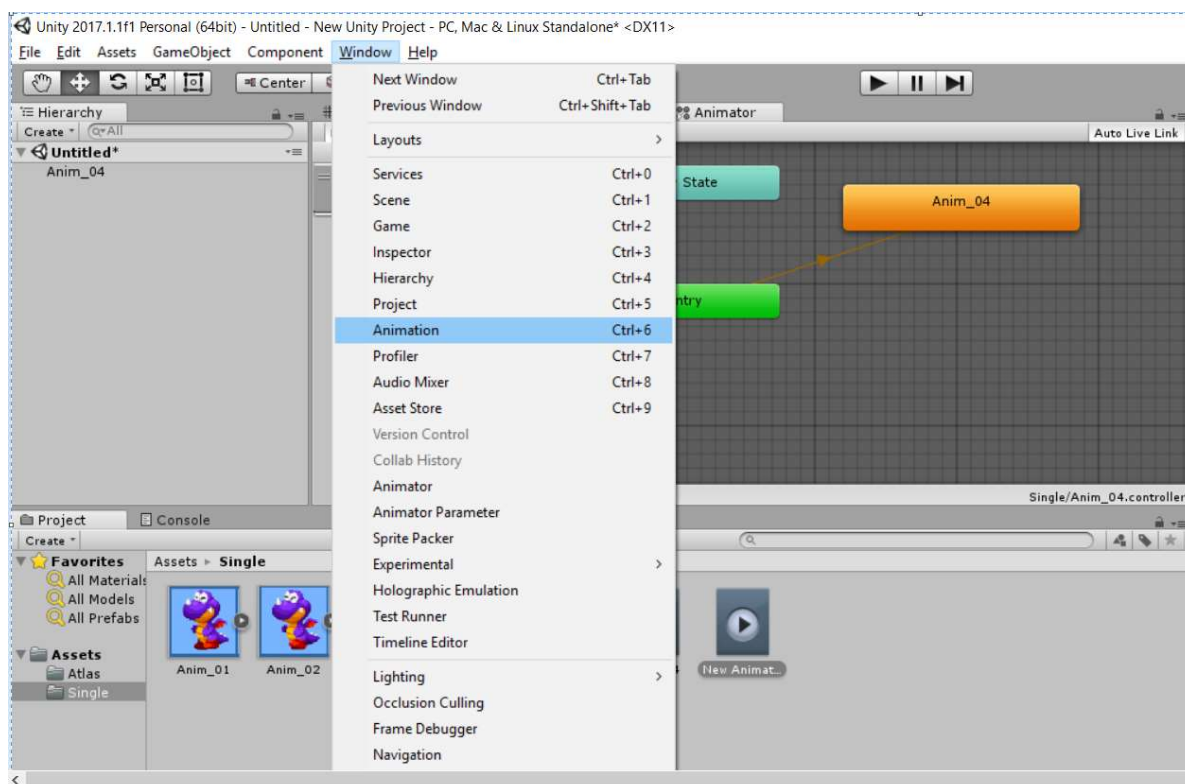


Рисунок 5.20– Вікно Animation

Вибравши ключовий кадр графіка, в якому знаходиться те зображення, яке не повинно бути показано в цей час, ви можете це легко виправити, натиснувши на поле **Sprite** в інспекторі об'єктів і вибравши новий спрайт в браузері спрайтів. Unity автоматично внесе потрібні зміни, і обраний вами спрайт буде відображатися в цьому ключовому кадрі.

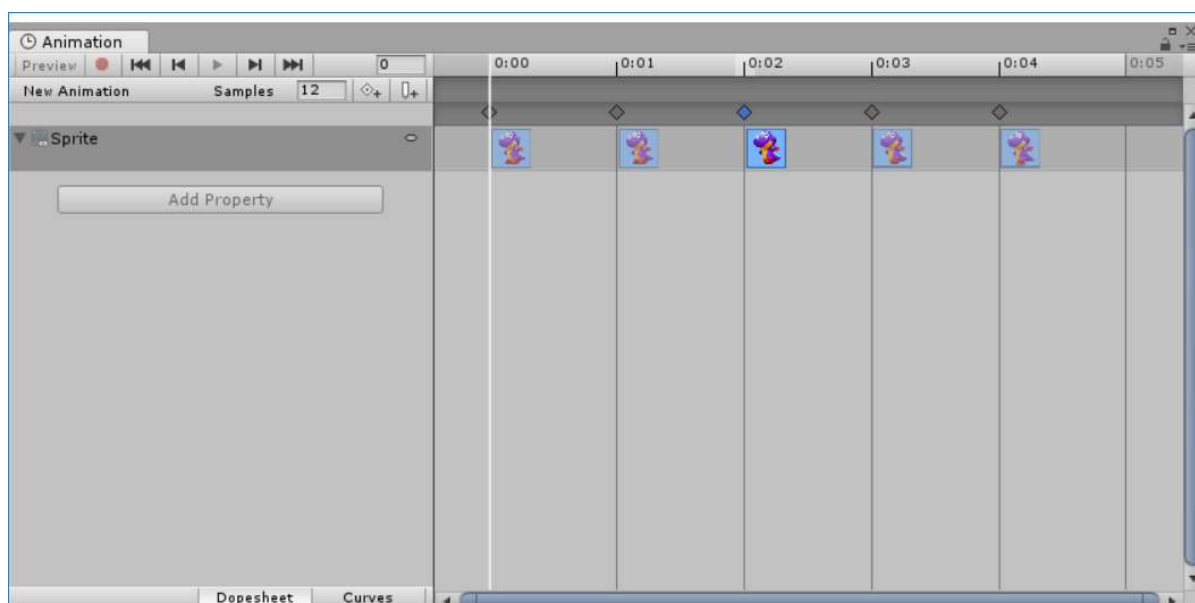


Рисунок 5.21– Ключові кадри анімації

Анімація персонажів

Імпортуємо заздалегідь підготовленого персонажа з «Asset Store» за допомогою меню «**Window** → **Asset Store**» (Ctrl + 9).

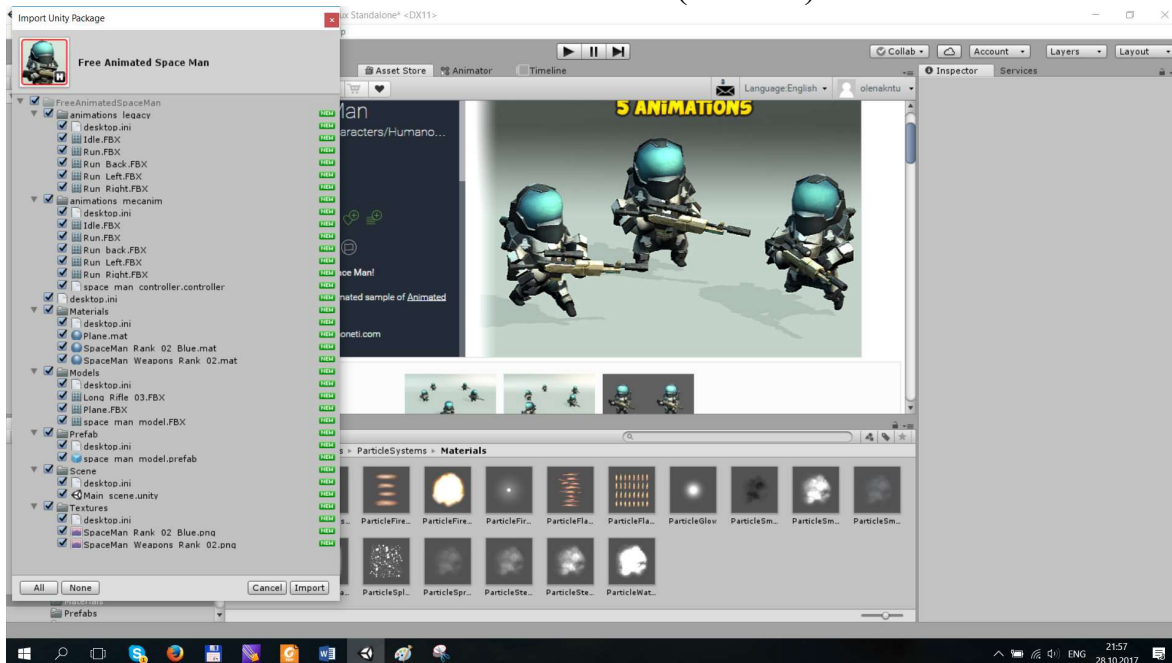


Рисунок 5.22– Імпорт персонажа з «Asset Store»

Після імпорту знаходимо у вікні «**Project**» префаб персонажа «FreeAnimatedSpaceMan». Перетягуємо цей префаб на сцену, і бачимо наш персонаж.

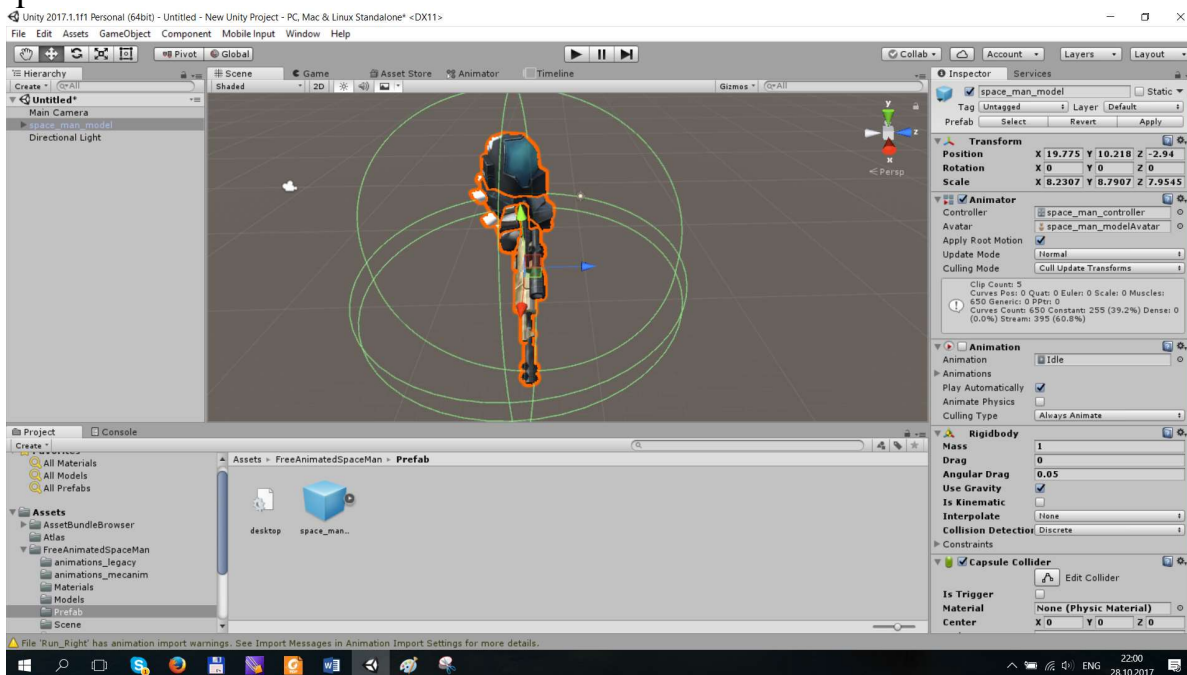


Рисунок 5.23– Персонаж «FreeAnimatedSpaceMan»

Натискаємо кнопку «**Add component → Physics → Rigidbody**» і «**Add component → Physics → Capsule Collider**». У «**Capsule Collider**» виставляємо розміри капсули у відповідності з розмірами персонажа.

У компонента «**Rigidbody**» активуємо властивість «**Use gravity**». Розкриваємо вкладку «**Constraints**» і вибираємо всі 3 галочки на «**Freeze Rotation**». Це потрібно для того що б персонаж не провалювався крізь землю.

Далі необхідно створити скрипт переміщення персонажа.

```
using UnityEngine;  
using System.Collections;  
public class Move: MonoBehaviour {  
public GameObject player;  
public int SRotation = 5;  
public int sp = 7;  
public AnimationClip SpaceMan;  
public int JSpeed = 30;  
void Start () {  
    player = (GameObject)this.gameObject;  
    animation.AddClip(SpaceMan, " FreeAnimatedSpaceMan");  
    }  
    void Update(){  
    if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))  
    {  
        player.transform.position += player.transform.forward * sp * Time.deltaTime;  
        animation.CrossFade("FreeAnimatedSpaceMan");  
    }  
    if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow))  
    {  
        player.transform.position -= player.transform.forward * sp * Time.deltaTime;  
    }  
    if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))  
    {  
        player.transform.Rotate(Vector3.down * SRotation);  
    }  
    if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))  
    {  
        player.transform.Rotate(Vector3.up * SRotation);  
    }  
    if (Input.GetKeyDown(KeyCode.Space))  
    {
```

```
player.transform.position += player.transform.up * JSpeed *  
Time.deltaTime;  
}  
}  
}
```

Після створення скрипта його необхідно додати до 3D-об'єкту шляхом перетягування скрипта на об'єкт в сцені.

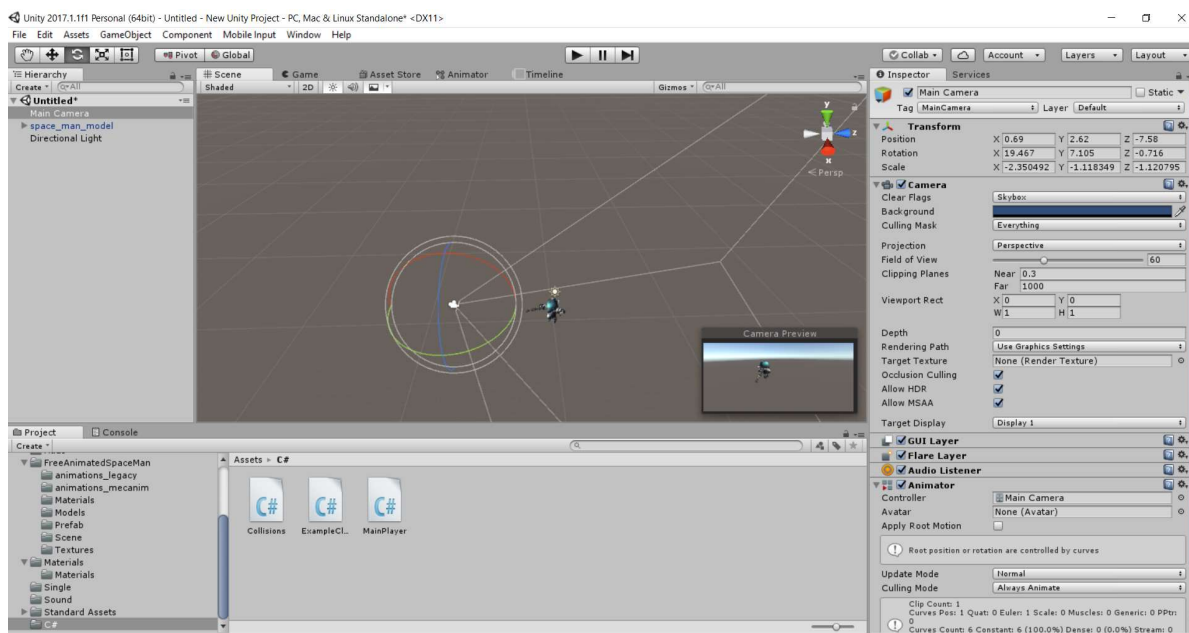


Рисунок 5.24– Персонаж «FreeAnimatedSpaceMan»

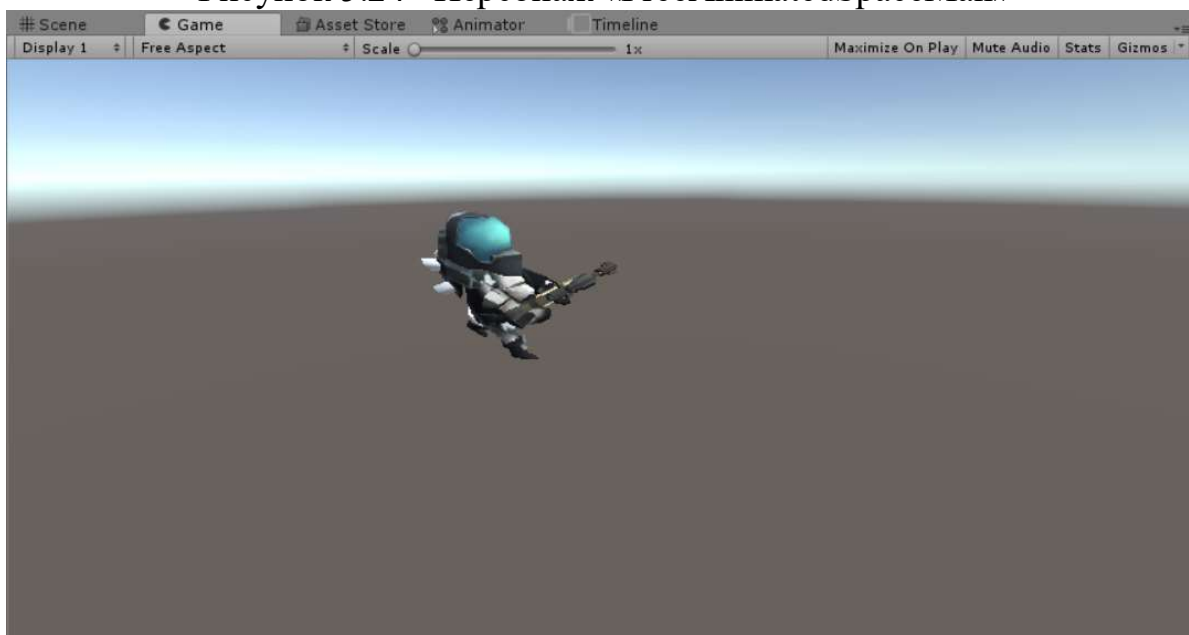


Рисунок 5.25– Анімація персонажу «FreeAnimatedSpaceMan»

Анімована текстура (Movie Texture)

Movie Textures це анімовані **Textures**, створені з відеофайлів. Переміщаючи відео файл в папку Assets Folder вашого проекту, ви можете використовувати імпортоване відео як звичайну текстуру.

Відео імпортуються через **Apple QuickTime**. Формати файлів, які може відтворювати **QuickTime** (зазвичай .mov, .mpg, .mpeg, .mp4, .avi, .asf). При імпортуванні відео в Windows, потрібно мати встановлений в системі **Quicktime** програвач.

При додаванні в проект відеофайлу, він буде автоматично імпортований і перетворений в формат **Ogg Theora format**. Відразу, після імпорту анімованої текстури, ви можете призначити її будь-який **GameObject** або **Material**, точно також як і будь-якій звичайній текстурі.

Програвання ролика

Анімована текстура не буде автоматично програтися при запуску гри. Для цього потрібно створити скрипт.

```
((MovieTexture)GetComponent<Renderer>().material.mainTexture).Play();
```

Далі необхідно призначити необхідний скрипт параметру «Включити відтворення текстури» (**toggle Movie playback**) після натискання пробілу:

```
public class PlayMovieOnSpace : MonoBehaviour {  
    void Update () {  
        if (Input.GetButtonDown ("Jump")) {  
            Renderer r = GetComponent<Renderer>();  
            MovieTexture movie = (MovieTexture)r.material.mainTexture;  
            if (movie.isPlaying) {  
                movie.Pause();  
            }  
            else {  
                movie.Play();  
            }  
        }  
    }  
}
```

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?

ЛЕКЦІЯ №6

ТЕМА: «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.

Поняття крос-платформної комп'ютерної 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 браузерях 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.

Створення ігрового проекту для ОС Windows

Під створенням крос-платформних додатків мається на увазі генерація прикладних пакетів, які запускаються на різних платформах. На кожній платформі (Windows, Android та ін.) своя форма пакета, але як тільки ви згенерували виконуваний файл, з'являється можливість поширювати гру і грати в неї без прив'язки до Unity.

Один проект Unity можна розгорнути на різних платформах - його не потрібно кожного разу генерувати заново.

Подивитися всі варіанти платформ можна у вікні «**Build Settings**». Відкрити вікно «**Build Settings**» можна за допомогою меню «**File→Build Settings**» (рис. 6.1).

Вікно «**Build Settings**» містить мінімальні налаштування, необхідні для збірки проекту. Перше, що необхідно зробити - додати в проект ігрову сцену. Зробити це можна за допомогою кнопки «**Add Open Scenes**», або просто перемістити файл сцени з вікна проекту в поле «**Scenes In Build**».

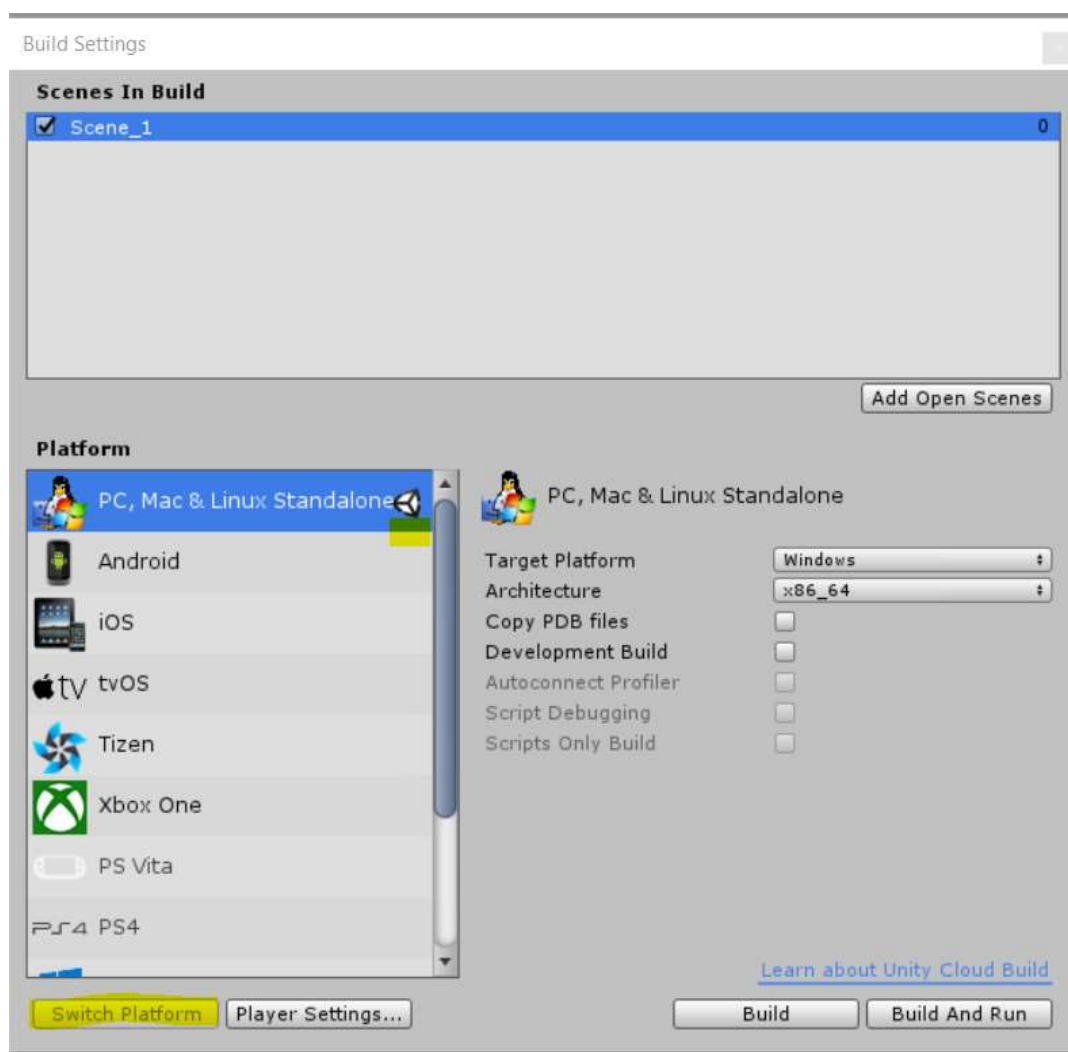


Рисунок 6.1– Вікно «**Build Settings**»

Зверніть увагу на цифру нуль, праворуч від назви файлу сцени. Це її порядковий номер. Якщо сцен в вашому проекті кілька - пам'ятаєте, що сцена, яка повинна буде завантажуватися першою, повинна отримати ідентифікатор нуль. Інакше, Unity автоматично завантажить замість вступного ролика сцену, що випадково отримала головний ідентифікатор.

Перелік платформ, які підтримує **Unity**, подано у лівій частині вікна. Активні платформи позначено значком **Unity**. Достатньо виділити платформу з цього переліку та натиснути кнопку «**Switch Platform**».

Далі праворуч на вкладці «**Target Platform**» вибираємо платформу «**Windows**», на вкладці «**Architecture**» вибираємо архітектуру, яка буде підтримуватися проектом.

Для генерації ігрового проекту у нижній частині вікна знаходяться кнопки «**Build**» і «**Build and Run**». Кнопка «**Build and Run**» відрізняється від кнопки «**Build**» тим, що автоматично запускає згенерований ігровий додаток.

Відкрити перелік налаштувань ігрового додатку на панелі «**Inspector**» можна за допомогою кнопки «**Player Settings**» (рис. 6.2). Ці налаштування контролюють різні аспекти готового додатка.

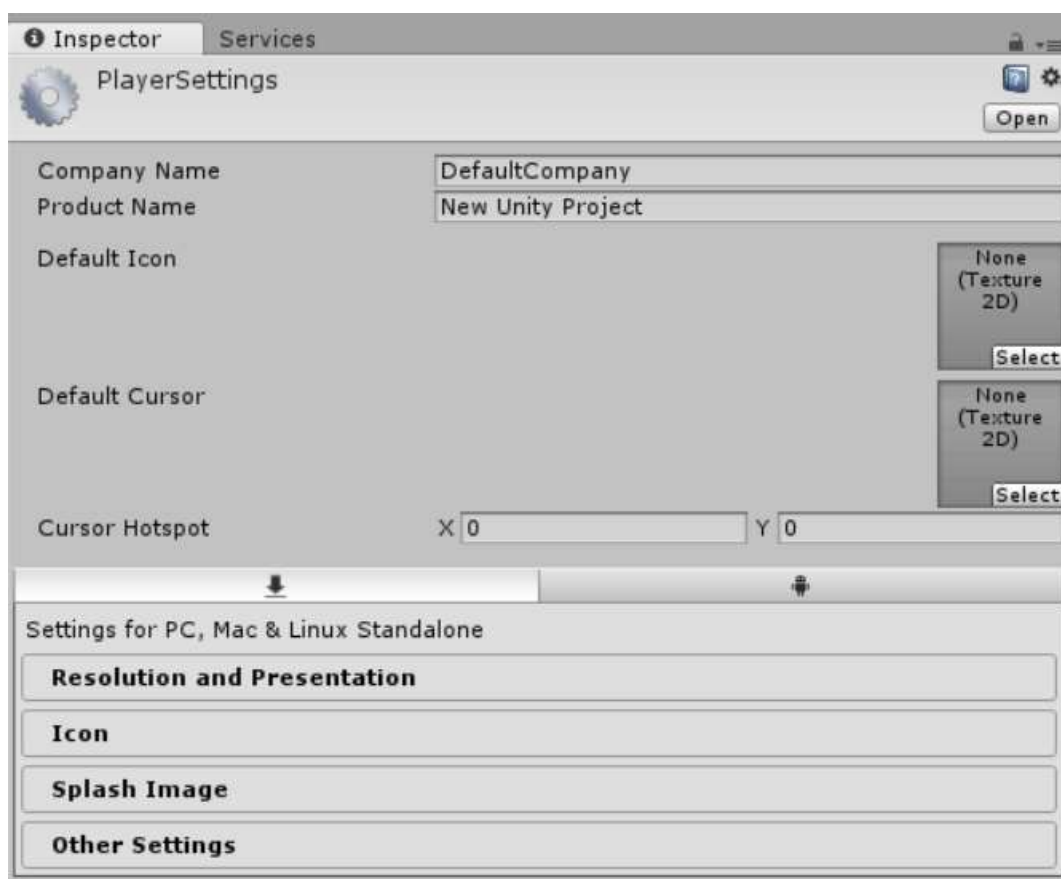


Рисунок 6.2– Вікно «**Player Settings**»

Поля «**Company Name**» і «**Product Name**» дозволять вказати назву гри і розробника. За допомогою «**Default Icon**» можна встановити іконку ігрового додатку. Для цього необхідно обрати зображення (при необхідності імпортувати його) на вкладці «**Project**» та перетягнути його у розділ «**Texture 2D**» на панелі «**Inspector**». Крім того, за допомогою «**Default Cursor**» можна також задати зображення курсора.

Після натискання на кнопку «**Build**» відкриється вікно вибору файла, в якому потрібно вказати адресу для генерації пакета ігрового додатку. Відразу після вказівки місця розташування почнеться процес побудови, після чого **Unity** створить виконуваний файл для активної в даний момент платформи. У нашому випадку це платформа «**Windows**».

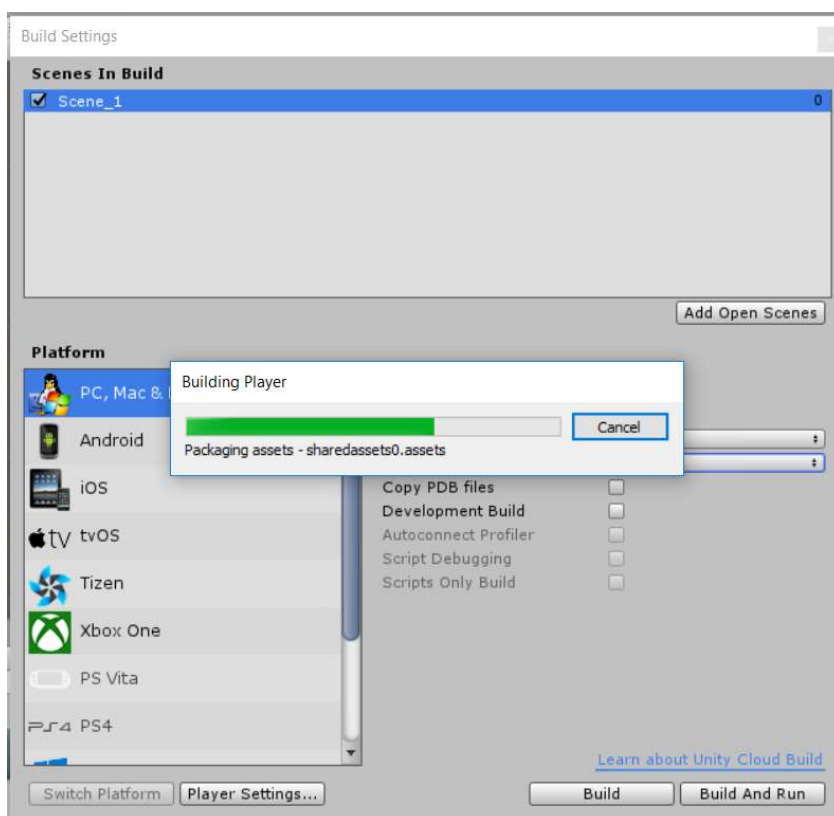


Рисунок 6.3– Процес побудови ігрового проекту

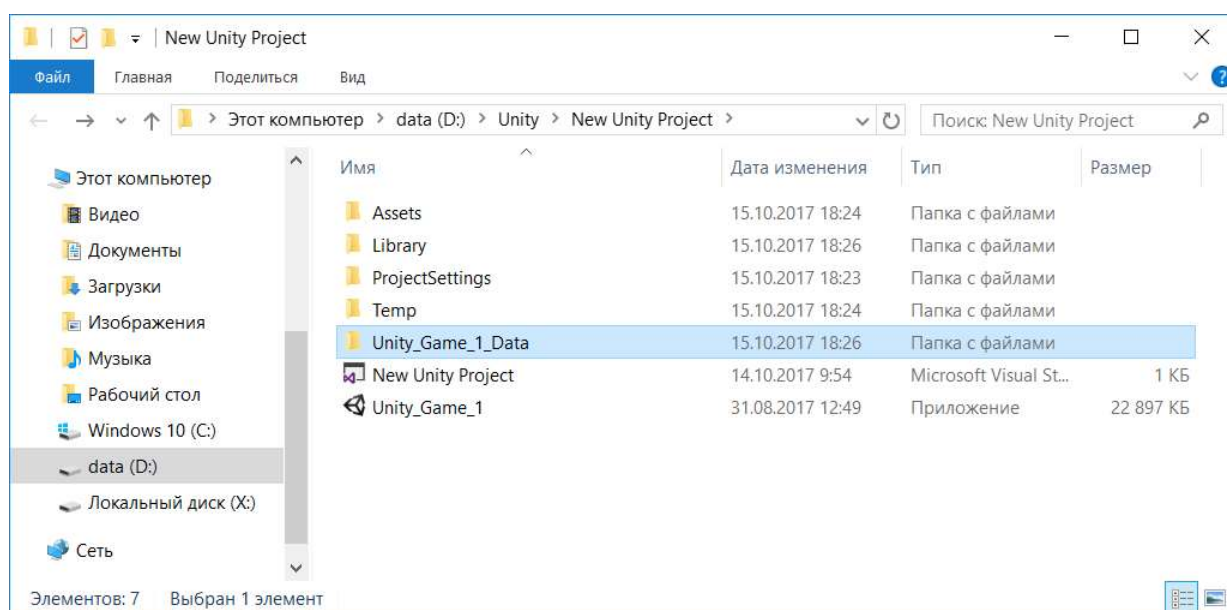


Рисунок 6.4– Файлы проекта для платформы «**Windows**»

На додаток також впливають налаштування, доступ до яких здійснюється за допомогою меню «**Edit**». Зокрема, саме тут налаштовується візуальна якість готового додатка.

Виберіть в меню «**Edit**» команду «**Project Settings**», а потім в додатковому меню - команду «**Quality**» (рис.6.5).

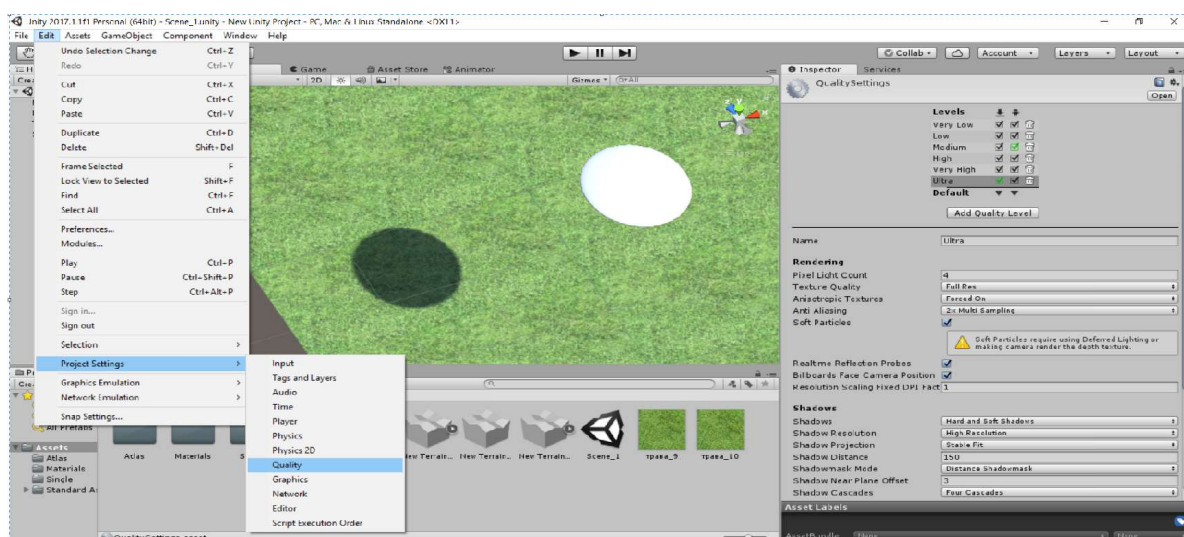


Рисунок 6.5– Меню «Edit»

На панелі «**Inspector**» з'являться елементи управління якістю, найбільш важливими з яких є прапорці в розташованій зверху групі. У верхньому рядку знаходяться значки можливих платформ, а збоку вказані варіанти налаштувань якості. Встановлені прапорці показують доступні для даної платформи налаштування, а ті, що виділені зеленим - поточні налаштування.

У більшості випадків за замовчуванням застосовується варіант «**Very Low**» (мінімальна якість), але можна вибрати варіант «**Very High**» або «**Ultra**» (максимальна якість) (рис. 6.6).



Рисунок 6.6– Елементи управління якістю

Створення ігрового проекту для ОС Android

Unity має можливість генерувати файли формату **APK** (Android Application Package). Для цього необхідно додати в **Unity** шлях до середовища розробки **Android SDK**, яке вже містить необхідний компілятор.

Android - операційна система (ОС) для смартфонів, планшетних комп'ютерів, електронних книг, цифрових програвачів, нетбуків, смартбуків, очок Google, телевізорів, систем автоматичного керування автомобілем та іншими пристроями. ОС базується на Linux-ядрі та власної реалізації віртуальної машини Java від Google.

На даний момент останньою версією є Android 7.1 Nougat, яка вийшла в жовтні 2016 року:

Таблиця 6.1 – Версії Android

Версія	Кодове ім'я	Дата випуску	Рівень API	Частка ринку (2017 рік)
7.0	Nougat	4 жовтня 2016	25	0,4%
7.0	Nougat	22 серпня 2016	24	2,4%
6.0	Marshmallow	5 жовтня 2015	23	31,3%
5.1	Lollipop	9 березня 2015	22	23,1%

5.0	Lollipop	3 листопада 2014	21	9,4%
4.4	KitKat	31 жовтня 2013	19	20,8%
4.3	Jelly Bean	24 липня 2013	18	1,5%
4.2	Jelly Bean	13 жовтня 2012	17	5,4%
4.1	Jelly Bean	9 липня 2012	16	3,7%
4.0	Ice Cream Sandwich	16 грудня 2011	15	1%
2.3	Gingerbread	9 лютого 2012	10	1%

Налаштування Android SDK

Android SDK – це середовище розробки додатків для операційної системи **Android**. **Android SDK** дозволяє створювати і тестувати **Android**-додатки, що використовують камеру мобільного пристрою, акселерометр, компас, дані GPS, доступ по Bluetooth, Wi-Fi, EDGE і 3G.

Android SDK підтримує роботу з мультимедійним контентом (аудіо, відео, зображення в форматах MPEG4, H.264, MP3, AAC, AMR, JPG, PNG і GIF), базами даних SQLite, інтегрованим браузером на движку WebKit, віртуальною машиною Dalvik, GSM телефонією і т.д. Користувачі **Android SDK** мають можливість тестувати створені ними додатки за допомогою вбудованого емулятора.

Завантажити **Android Studio** з **Android SDK** можна з офіційного сайту за посиланням <https://developer.android.com/studio/index.html>

Відкриємо **Android Studio**. Внизу стартового екрана програми знайдемо кнопку «Configure» і натиснемо на неї.

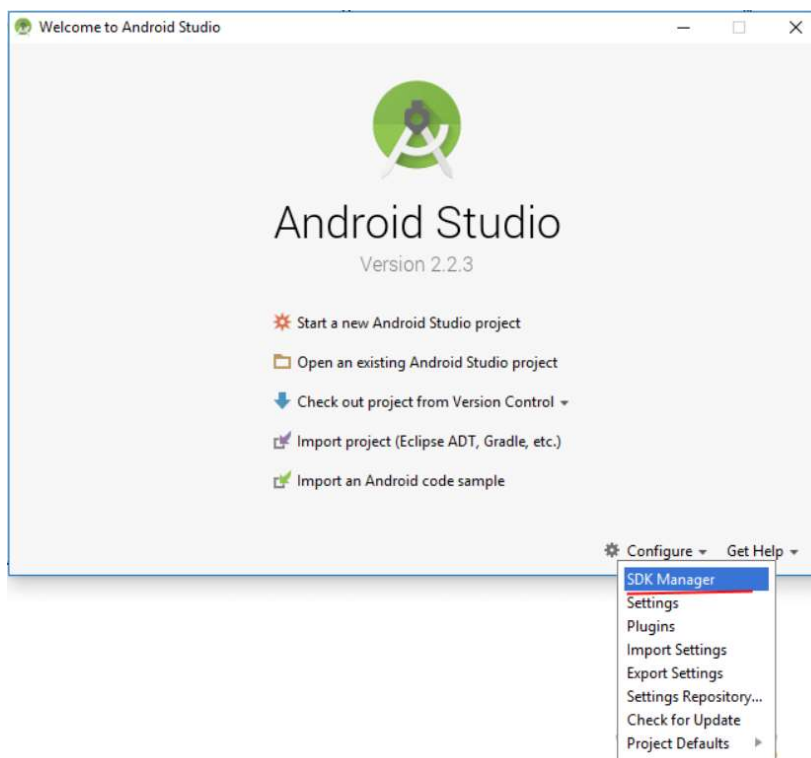


Рисунок 6.7 - Стартовий екран програми **Android Studio**

В меню натиснемо на пункт «SDK Manager». Після цього відкриється вікно з настройками для Android SDK Manager:

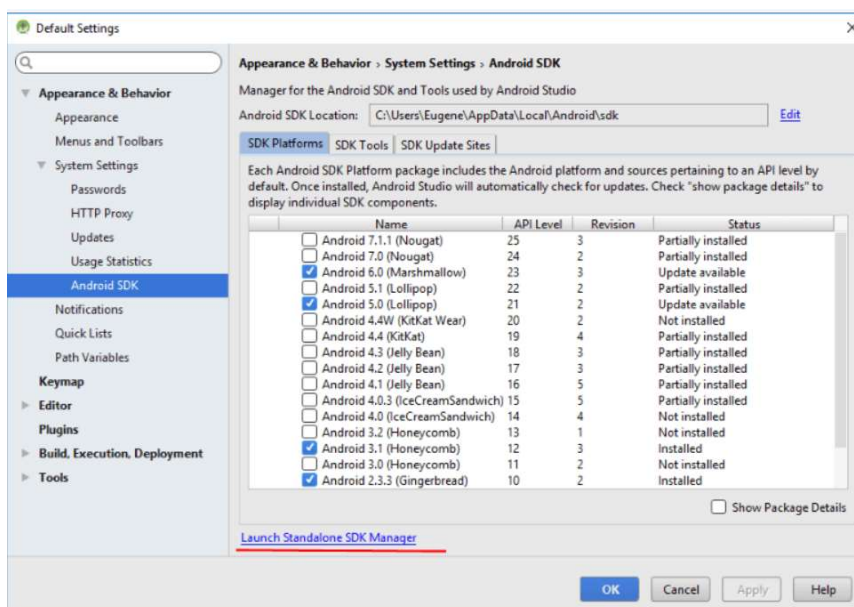


Рисунок 6.8 - Android SDK Manager

Тут ми можемо побачити, які версії **SDK** встановлені. Але кожна версія передбачає широке коло компонентів. Не всі ці інструменти можуть знадобитися, тому немає сенсу встановлювати всі версії SDK повністю.

Для більш детального перегляду всіх компонентів по кожній платформі натиснемо на посилання **Launch Standalone SDK Manager**, яке розташоване внизу вікна. Це посилання знову запустить **Android SDK Manager**, але в окремому вікні.

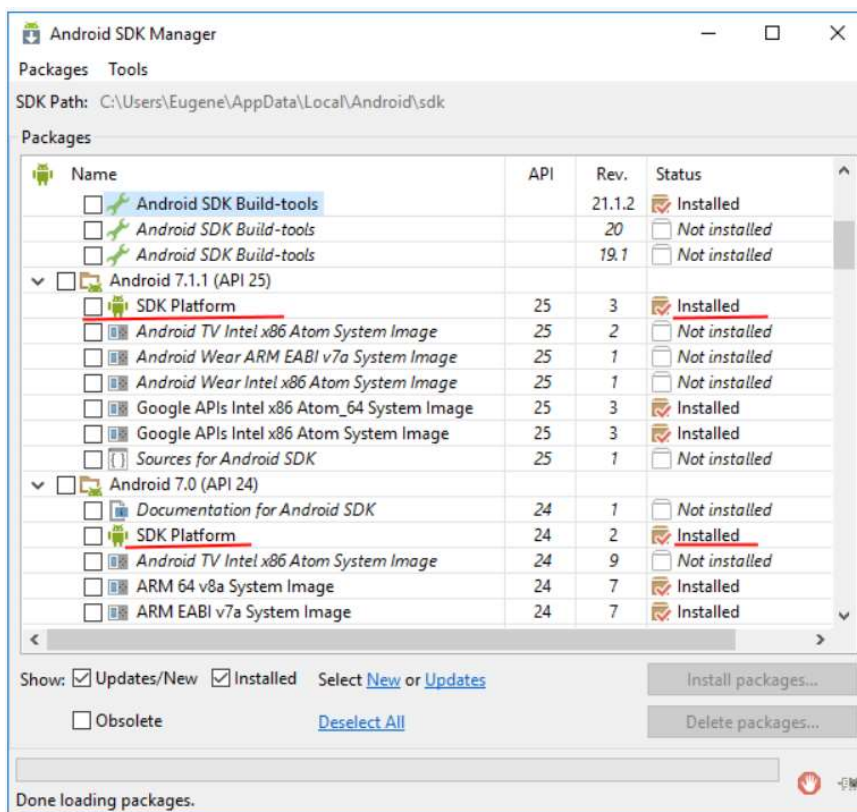


Рисунок 6.9 - Посилання Launch Standalone SDK Manager

Структурно **SDK Manager** побудований таким чином. Спочатку йде секція **Tools**. Вона містить пакети, необхідні для розробки програми.

Далі йдуть секції платформи **Android**. Тут ми детально можемо подивитися, які пакети для кожної платформи встановлені. В даному випадку нас перш за все буде цікавити пункт **SDK Platform**.

SDK Platform містить весь основний функціонал, який використовується при розробці. Даний пункт можна виділити для всіх тих платформ, під які ми збираємося компілювати Unity-додаток.

Крім **SDK Platform** кожна платформа, як правило, містить ще ряд компонентів:

Google API: різні API від Google, зокрема, інтерфейси по роботі з Google Maps, іншими сервісами Google.

ARM EABI v7a System Image: образ системи, що емулює роботу процесора ARM, необхідний для запуску емулятора.

Intel x86 Atom System Image / Intel x86 Atom_64 System Image: образ системи (для 86-х і 64-х платформ) від компанії Intel, також необхідний для запуску емулятора.

Google APIs ARM EABI v7a System Image: Google API для образу системи ARM EABI v7a System Image.

Google APIs Intel x86 Atom System Image / Intel x86 Atom_64 System Image: Google API для образу системи Intel x86 Atom System Image або Intel x86 Atom_64 System Image

Якщо тестування додатку буде відбуватися на емуляторі, то слід встановити для цього образ системи, який відповідає поточній робочій машині, і відповідне **Google API**.

Після всіх платформ в **SDK Manager** розташована секція **Extras**. Тут перераховані додаткові пакети, не прив'язані до платформ.

В секції **Extras** важливі такі пакети як:

Android Support Repository

Android Support Library

Google Play Services

Google Repository

Google Usb Driver

Ці пакети встановлюють репозиторії **Android** і сервіси **Google Play**. Крім того, якщо ви хочете використовувати для тестування додатків смартфон, то необхідно буде встановити USB-драйвер безпосередньо виробника смартфона.

Як правило, при підключенні смартфона система сама намагається встановити драйвер.

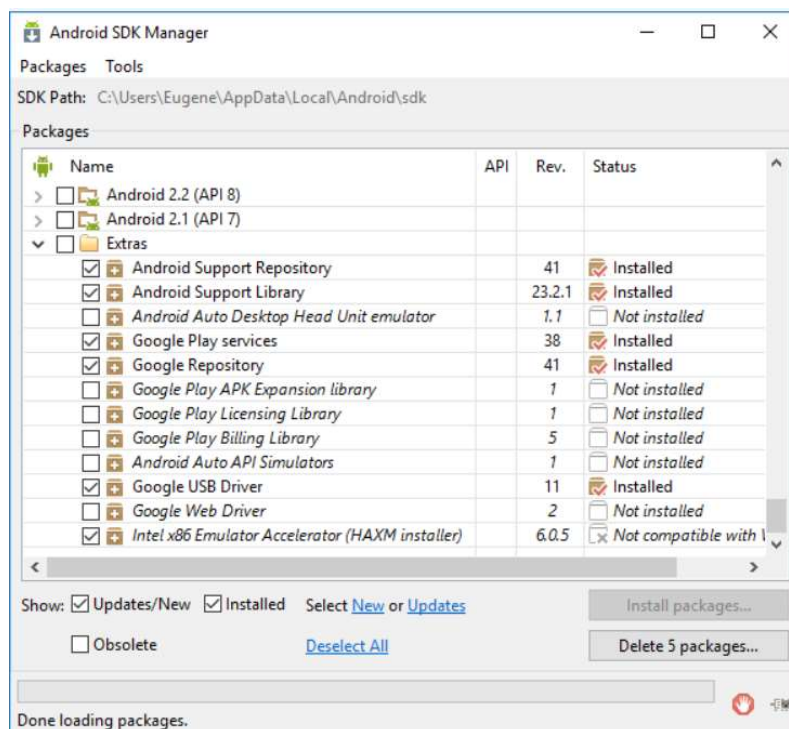


Рисунок 6.10 - Секція Extras

Вкажіть шлях до файлу **Android SDK** на вкладці «**External Tools**» у вікні «**Unity Preferences**».

Відкрити вікно «**Unity Preferences**» можна за допомогою меню «**Edit**→**Preferences**».

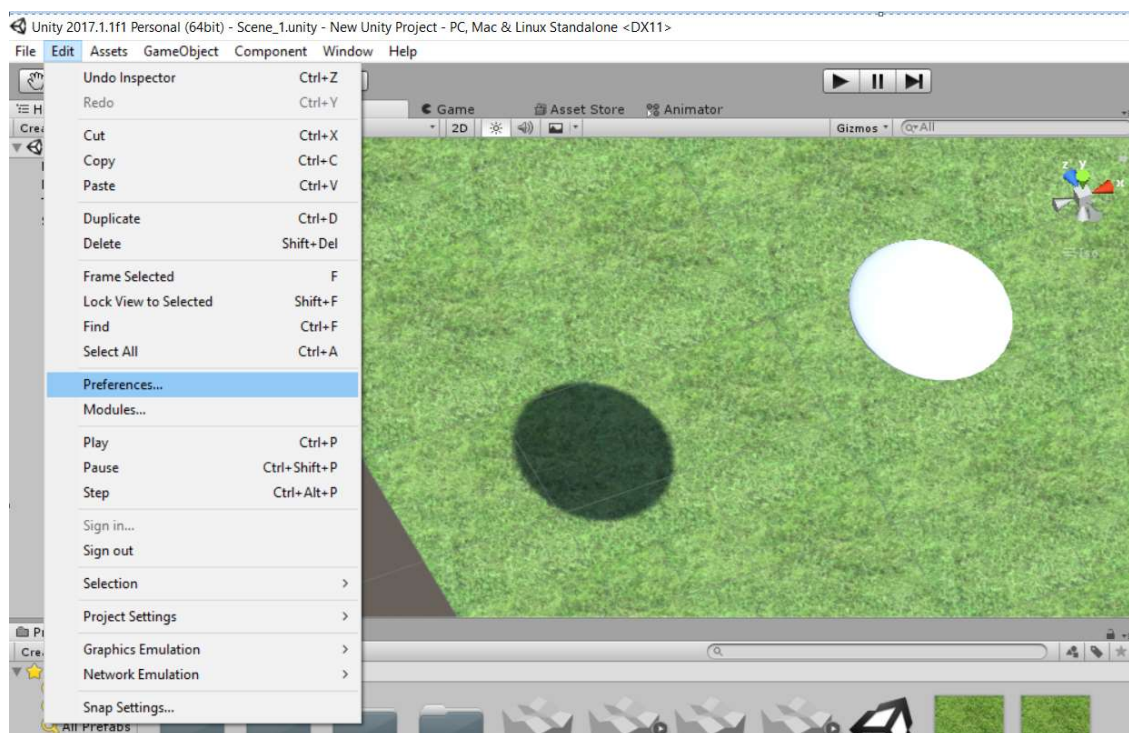


Рисунок 6.11– Меню «**Edit**→**Preferences**»

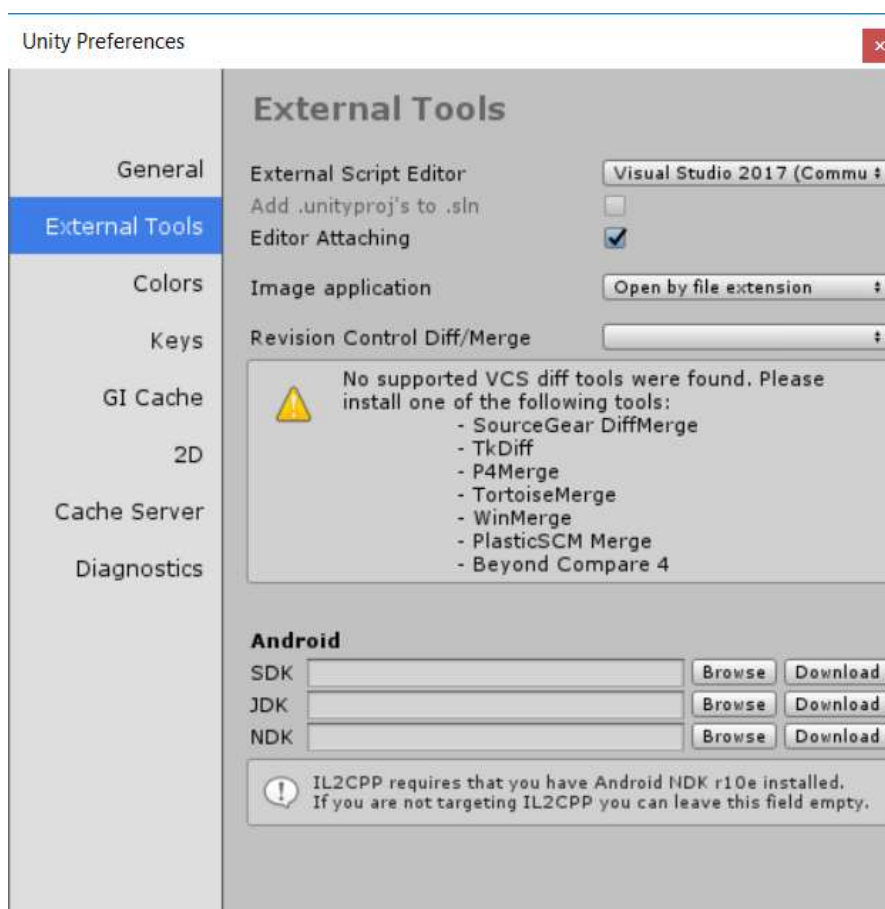


Рисунок 6.12– Вкладка «External Tools» у вікні «Unity Preferences»

Після цього необхідно задати параметр «**Package Name**» в розділі «**Other Settings**» на панелі «**Inspector**» у вигляді **com.назвакомпанії.назвапродукта** та запустить процес збірки ігрового проекту за допомогою кнопки «**Build**».

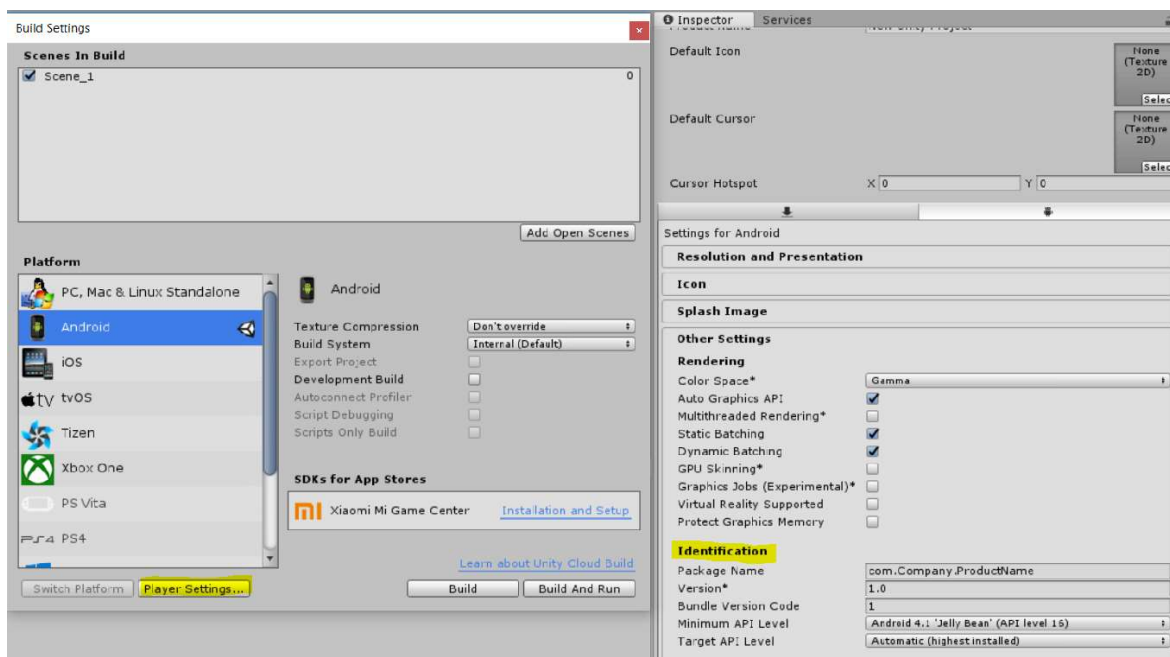


Рисунок 6.13– Налаштування параметру «Package Name»

Отриманий **APK** пакет ігрового додатку необхідно встановити на мобільний пристрій.

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 BundleIdentifier?
- Назвіть основні етапи налаштування інструментів збірки для Android?

Вступ

Комплекс лабораторних робіт модуля «Розробка комп'ютерних ігор за допомогою Unity 3D» виконується відповідно до програми навчальної дисципліни «Технології розробки крос-платформних комп'ютерних ігор» та передбачає поглиблення теоретичних знань з навчальної дисципліни та набуття практичних навичок роботи з Unity 3D за допомогою спеціалізованого ігрового обладнання.

До комплексу лабораторних робіт входять індивідуальні завдання, які 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.

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).

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 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.

Обладнання

Комплекс лабораторних робіт модуля «Розробка комп'ютерних ігор за допомогою Unity 3D» виконується в ігровій навчальній лабораторії «GameLab».

За призначенням ігрову навчальну лабораторію «GameLab» поділено на чотири робочі зони:

1. Зона розробки та тестування ігор в середовищі Android, яка містить робочі місця з наступною комп'ютерною технікою:

- 15 робочих місць студентів для розробки ігор, кожне з яких оснащено ноутбуком HP Pavilion 15-ab246ur, ігровою клавіатурою Logitech G103 та маніпулятором Logitech F310;

- 5 спеціалізованих робочих місць, кожне з яких оснащено планшетом Lenovo Tab 3 Business X70F, обладнанням ігрової віртуальної реальності Razer Virtual Reality (OSVR) ecosystem&headset, шоломами віртуальної реальності EMOTIV Insight 5 Channel Mobile EEG + USB Receiver та ігровою гарнітурою TrackIR 5 Premium Head Tracking for Gaming.

2. Зона розробки та тестування ігор в середовищі Apple iOS, яка містить 2 спеціалізованих робочих місця, кожне з яких оснащено комп'ютером Apple A1418 iMac 21.5" та планшетом Apple A1566 iPad Air 2.

3. Зона запису, обробки та прослуховування аудіоконтенту гри, яка оснащена звуковим USB-інтерфейсом 2x4 BEHRINGER U-Phoria UMC204HD з мікрофонами BEHRINGER, кабелями PROEL BULK250LU5, мікрофонними стійками PROEL RSM180 та навушниками Sennheiser HD558 для якісного запису аудіоігрового контенту.

4. Зона розробки фото- та відеоігрового контенту гри, яка оснащена зеркальною фотокамерою Canon EOS700D 18-55mm, сканером Epson Perfection V370 Photo, лазерним принтером Canon i-SENSYS LBP6030W для друку через WiFi та з мобільних пристроїв.

Сервер HP ProLiant ML350 Gen9 забезпечує безперебійне функціонування мережі Wi-Fi навчальної лабораторії «GameLab».

Крім того, ігрова лабораторія «GameLab» оснащена ігровими приставками для тестування ігор Sony PlayStation 4 500GB та XBOX 360 Slim 250Gb Kinect.

В ігровій навчальній лабораторії «GameLab» переважно використовується програмне забезпечення з відкритим кодом, що передається у вільне користування для студентів та викладачів для некомерційної експлуатації.

Це насамперед крос-платформні середовища та засоби розробки додатків для операційних систем Android, iOS, Windows з використанням мов програмування Java, JavaScript, Python, C++.

Але для створення якісного ігрового контенту залучено також комерційне програмне забезпечення, для якого відсутні якісні вільно розповсюджені аналоги, а саме:

- крос-платформні професійні графічні програми для обробки та покращення растрових і векторних зображень, фотографій, графічного дизайну (Adobe Photoshop CC ALL Multiple Platforms);
- професійні графічні програми моделювання високополігональних цифрових скульптур і текстурного фарбування 3D моделей для створення цифрових 3D об'єктів і 2D скетчів з широким спектром текстур і фарб (Autodesk Mudbox);
- спеціалізоване програмне забезпечення для якісної 3D-анімації, рендеринга, моделювання та візуалізації із застосуванням відеоефектів (Autodesk Maya 2016);
- спеціалізоване програмне забезпечення для підготовки, компонування, редагування, запису та завантаження аудіоконтенту високої якості (Sound Forge Audio Studio 10);
- спеціалізоване програмне забезпечення для підготовки, компонування, редагування, запису та завантаження відеоконтенту різної роздільної здатності та якості (Corel VideoStudio Pro X9 ML).

Таким чином, обладнання та програмне забезпечення ігрової навчальної лабораторії «GameLab» відповідає сучасним вимогам підготовки фахівців з розробки крос-платформних комп'ютерних ігор та ігрового контенту.

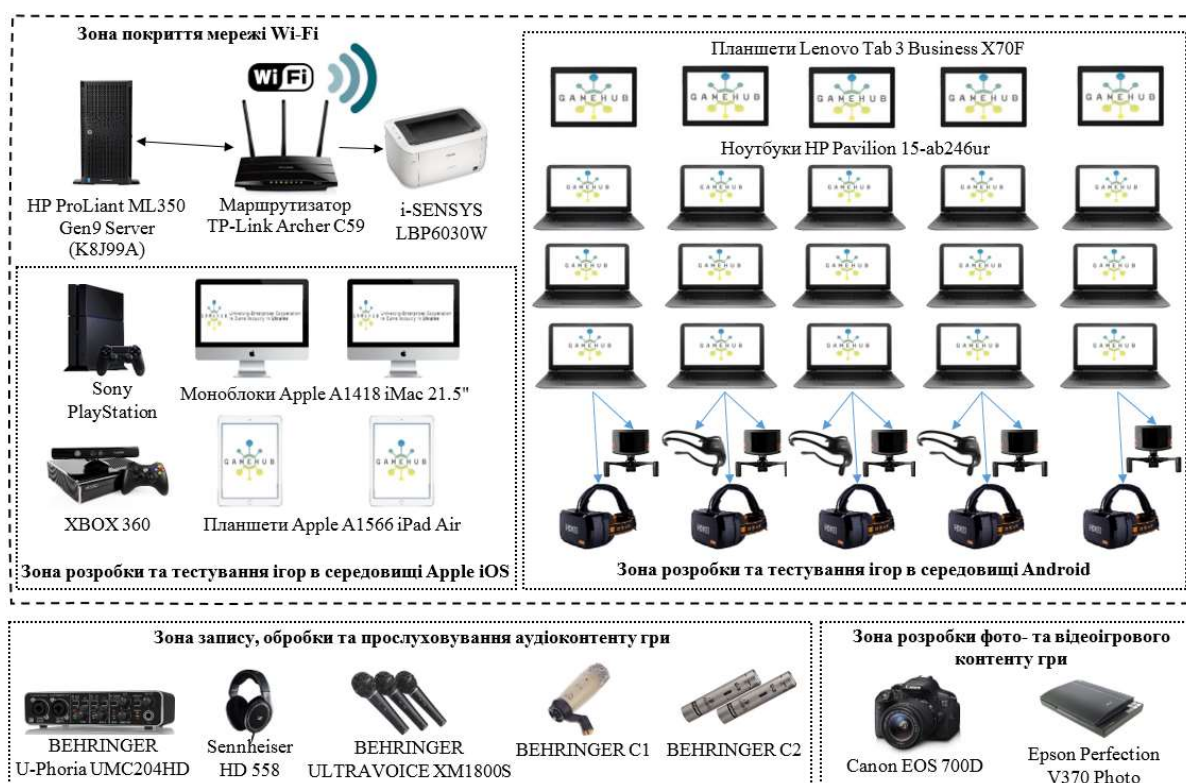


Рисунок 1.1 - Концепція ігрової навчальної лабораторії «GameLab»

ЛАБОРАТОРНА РОБОТА № 1

ТЕМА: «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.

Режими роботи та інтерфейс Unity 3D

Робоче вікно Unity 3D розбите на 6 взаємопов'язаних областей (рис. 1.1). (За замовчуванням в Unity 3D включено вид «**Default**» або «**Tall**». Якщо у вас інший вид робочого вікна, то можете переключити його в меню «**Layout**» у верхньому правому кутку вікна).

Перелік всіх робочих областей вікна Unity 3D подано у табл. 1.1.

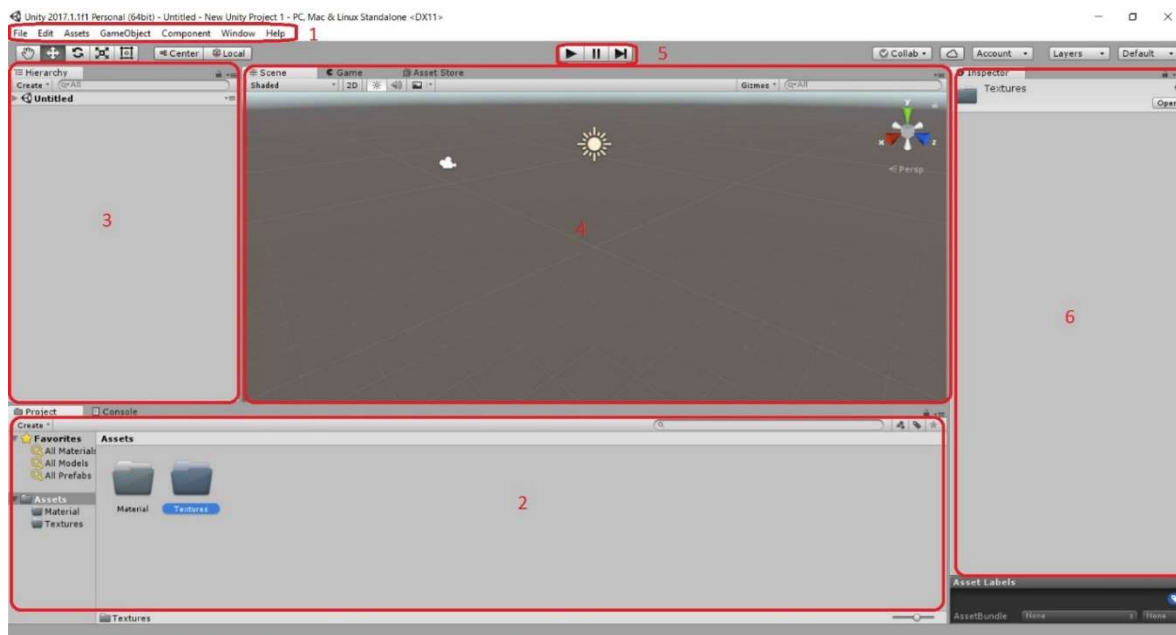


Рисунок 1.1- Робоче вікно Unity 3D

Таблиця 1.1 - Перелік робочих областей вікна Unity

Перелік робочих областей	Призначення робочих областей
1. Main menu (Головне меню)	Рядок тексту зверху, де розташовуються всі команди, доступні в програмі. Багато команд продубльовано кнопками і меню в робочих областях, тому головне меню не обов'язково використовувати.
2. Project View (Огляд проекту)	Список, в якому показані всі файли, які використовуються в грі: файл сцени, файл програмного коду, графічні і аудіофайли.
3. Hierarchy (Ієрархія)	Список об'єктів, які додано на сцену. Тут можна працювати з об'єктами, копіювати їх, перейменовувати, видаляти.
4. Scene (Сцена)	Область, де відображається ігровий світ або ігрова сцена. Тут ми можемо додавати нові об'єкти, перетягувати їх, змінювати вид.
5. Game (Гра)	Область попереднього перегляду, де видно, як сцена буде виглядати в грі. Тут можна налаштовувати різні налаштування екрану і режиму відео.

6. Inspector (Інспектор)	Список, що складається з декількох різних по виду розділів. Показує всі властивості обраного об'єкта: розміри, моделі, текстури, скрипти.
------------------------------------	---

Головне меню Unity 3D

Головне меню Unity 3D стандартно розташовується у верхньому лівому кутку вікна (рис. 1.2).

У головному меню містяться взагалі всі команди програми, а в основних областях вікна вони лише дублюються.

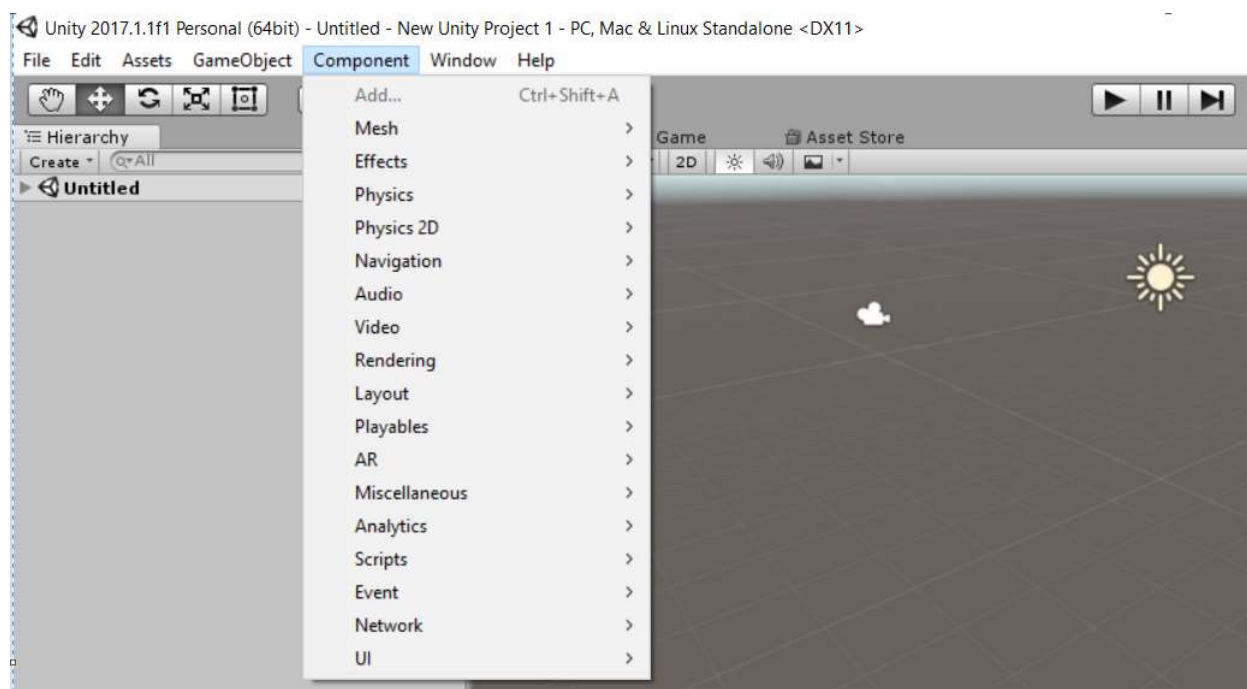


Рисунок 1.2- Головне меню Unity 3D

Огляд проекту (Project View)

Кожен проект містить папку «Assets». Вміст цієї папки представлено в області «Project View» (рис. 1.3). Це ресурси гри: файли-скрипти, 3D-моделі, текстури, аудіофайли, префаби (об'єкти, які можна клонувати).

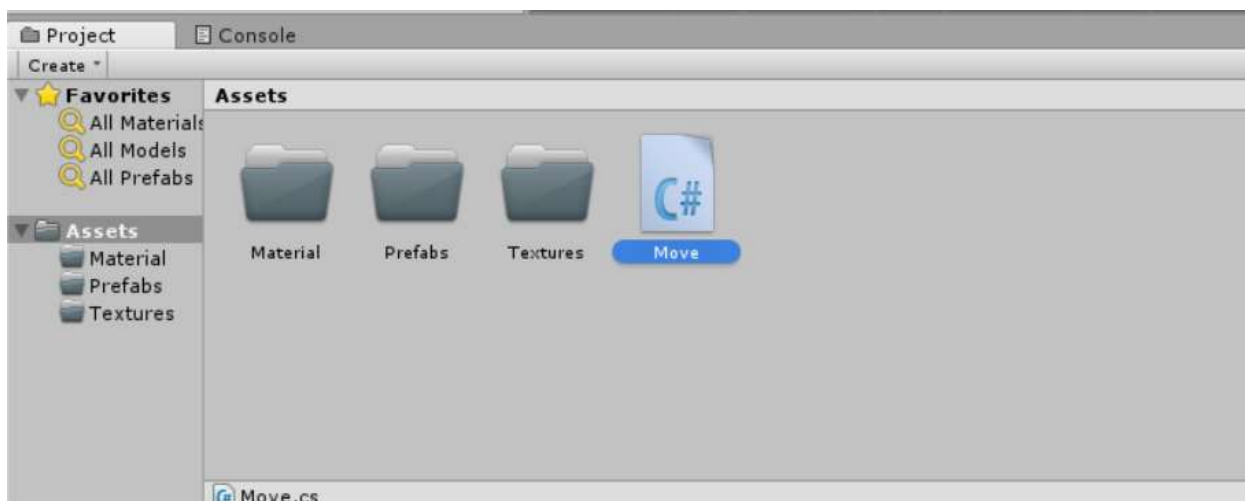


Рисунок 1.3- Область «Project View». Файли проекту

Переміщення файлів можна виконувати через стандартний **Провідник Windows**, вибравши «Reveal in Explorer». Але при переміщенні ресурсів в **Провіднику Windows** будуть втрачені всі посилання між об'єктами.

Краще завжди переміщати файли всередині проекту тільки в «**Project View**».

Додати новий ресурс в проект можна двома способами:

Перетягнути файл з **Провідника Windows** в область «**Project View**».

Виконати команду «**Assets**» → «**Import New Assets**».

Проекти гри складаються з одного або декількох файлів сцени. Кожна окрема сцена - це окремий рівень гри. Сцени так само зберігаються в папці «**Assets**» і відображаються в «**Project View**».

Деякі ігрові ресурси не є файлами, і створюються безпосередньо в Unity. Для створення ресурсу використовується меню «**Create**» (рис. 1.4).

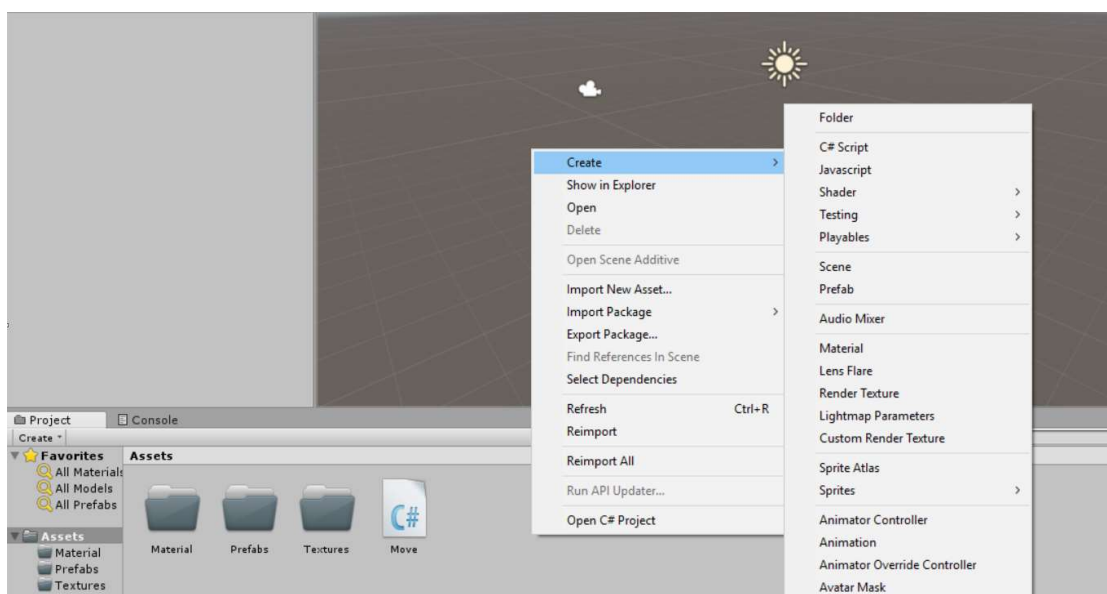


Рисунок 1.4 - Меню «Create»

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

Ієрархія (Hierarchy)

Ієрархія містить всі об'єкти (GameObject) відкритої сцени. Об'єкти, що додаються в сцену або видаляються з неї, відображаються або навпаки перестають відображатися в Hierarchy (рис. 1.5).

Спадкування (Parenting). В ієрархії Unity об'єктів можна задавати успадкування. Будь-який об'єкт може бути дочірнім по відношенню до іншого. Дочірній об'єкт буде рухатися і обертатися разом з батьківським об'єктом. Для створення дочірнього зв'язку досить перетягнути необхідний об'єкт на батьківський об'єкт в Hierarchy.

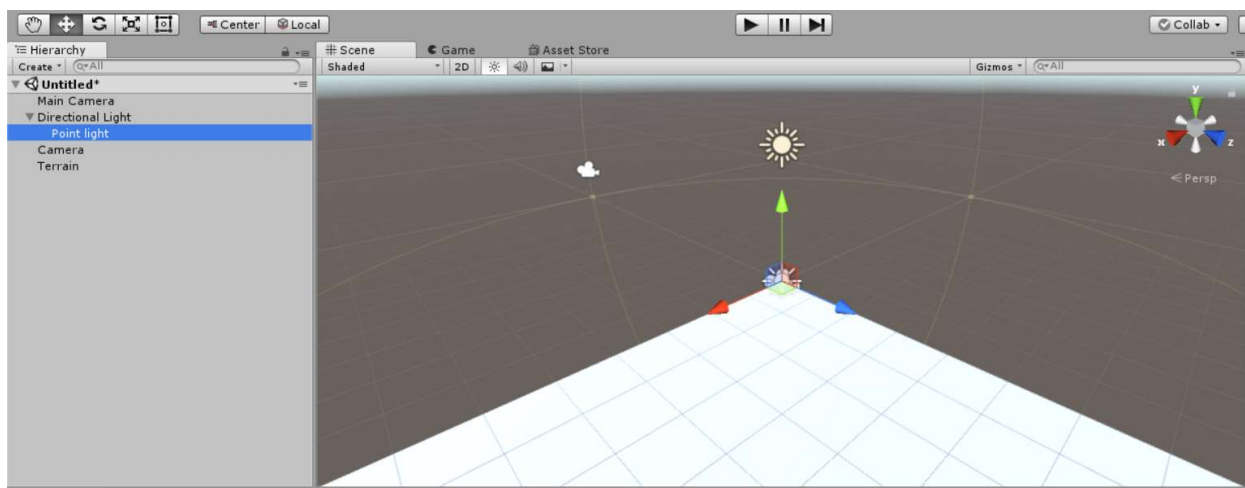


Рисунок 1.5 – Ієрархія об'єктів

Сцена (Scene View)

Ігрова сцена використовується для розстановки об'єктів (оточення, персонажі, камери, системи частинок та інше). Сцена може бути ігровим рівнем, головним меню, заставкою (рис. 1.6).

В 2D режимі управління є інтуїтивно зрозумілим. У 3D режимі є безліч прийомів для переміщення по сцені:

1. Затиснута права кнопка миші (ПКМ) активує режим вільного польоту.
2. Переміщатися можна клавішами WASD на манер гри в жанрі FPS.
3. Виберіть об'єкт у списку ієрархії і натисніть «F». Вид сцени буде центровано і масштабовано по вибраному об'єкту.

4. Затиснуті клавіша Alt та ліва кнопка миші (ЛКМ) буде крутити камеру навколо поточної точки опори.
5. Затиснуті клавіша Alt та середня кнопка миші (СКМ) будуть переміщати камеру.
6. Затиснуті клавіша Alt та ПКМ будуть масштабувати вигляд сцени.
7. Застосувати альтернативний режим переміщень можна за допомогою клавіші Q.

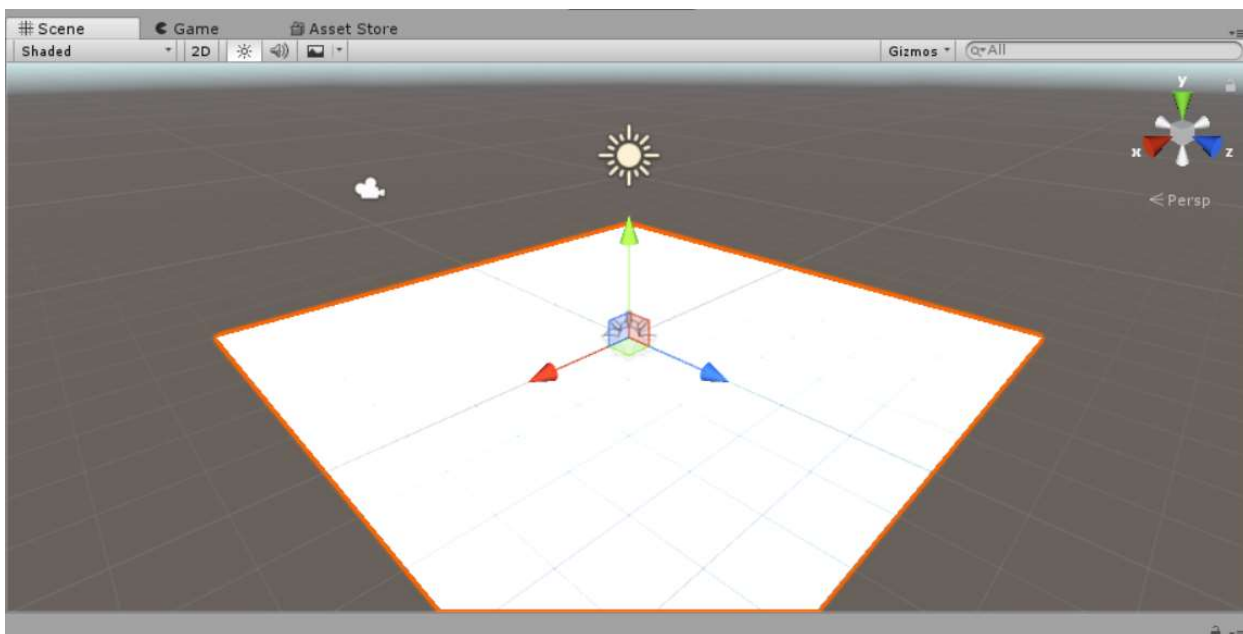


Рисунок 1.6 - Ігрова сцена

Ігровий вид (Game View)

Game View - попередній огляд гри (рендер з ігрової камери). У розташуванні вікон за замовчуванням «Game View» відсутній. Для його включення потрібно вибрати вкладку «Game» над ігровою сценою (рис. 1.7).

Play Mode

Три кнопки у верхній частині вікна Unity 3D відповідають за управління попереднім оглядом гри: «Play», «Pause» і «Step». Всі зміни, зроблені під час попереднього перегляду, скидаються при виході з нього. Винятком є зміни в префабах.

Game View Control Bar

Перше меню, що випадає 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).

Далі йде кнопка «Maximize on Play». Якщо вона натиснута, то «Game View» розтягується на всі вікна редактора при попередньому перегляді.

Кнопка «**Gizmos**» включає відображення контейнерів **Gizmo** в «**Game View**».

Остання кнопка - «**Stats**». Вона показує статистику рендеринга (**Rendering Statistics**), корисну при оптимізації.

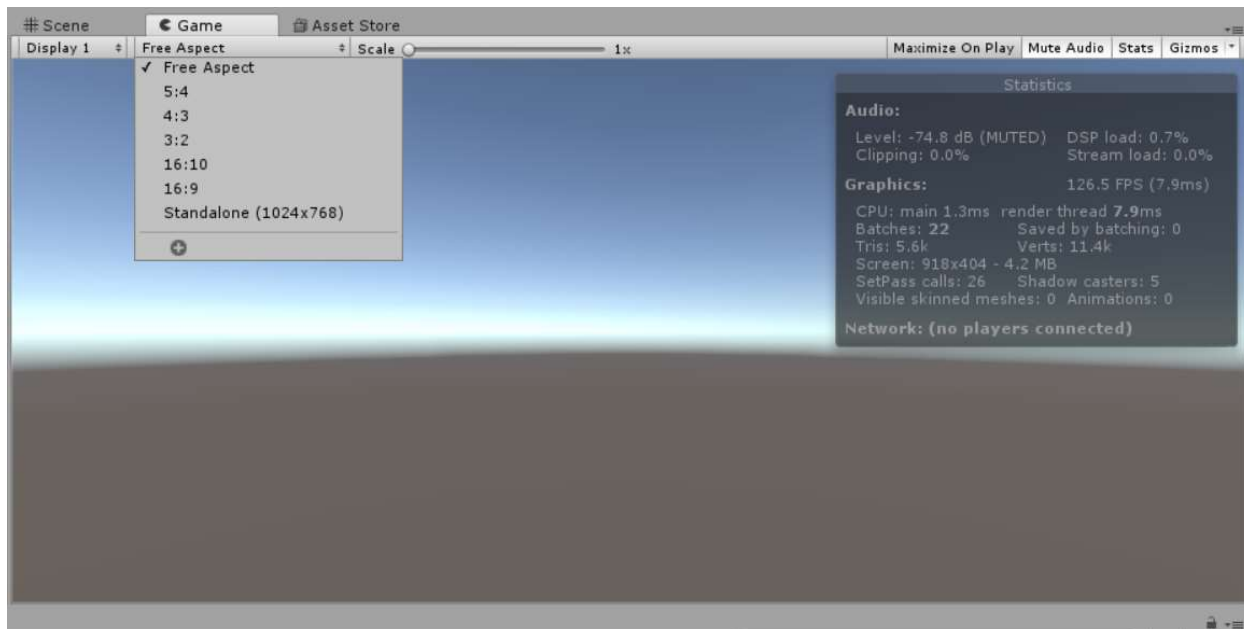


Рисунок 1.7 - Game View

Інспектор (Inspector)

GameObject - це будь-який об'єкт в грі. Сам по собі об'єкт не виконує дій. Йому потрібні спеціальні властивості, перш ніж він стане персонажем, оточенням або візуальним ефектом.

Об'єкти - це лише контейнери. Вони можуть містити в собі різні елементи, комбінації яких перетворюють об'єкт в персонаж, декорацію, спецефект. Ці елементи називаються компонентами (Components). Залежно від того, що потрібно створити, об'єкту присвоюються різні комбінації компонентів.

Перелік об'єктів міститься в області «**Hierarchy**». Якщо в списку ієрархії ми виберемо будь-який об'єкт, то в області «**Inspector**» відобразяться всі властивості цього конкретного об'єкта (рис. 1.8). Тут же можна редагувати всі ці властивості або додавати нові.

Об'єкт може містити в собі такі типи компонентів: розташування в просторі (**Transform**), меши (**meshes**), скрипти (**scripts**), звуки, джерела світла (**Lights**) та інші елементи.

Status Bar

Рядок стану (**Status Bar**) розташований у нижній частині вікна редактора. Він відображає помилки компіляції. Якщо виникають проблеми з

грою, варто заглянути в рядок стану. Подвійне натискання на ньому відкриє вікно консолі (**Console**), в якому відображаються всі помилки (рис. 1.9).

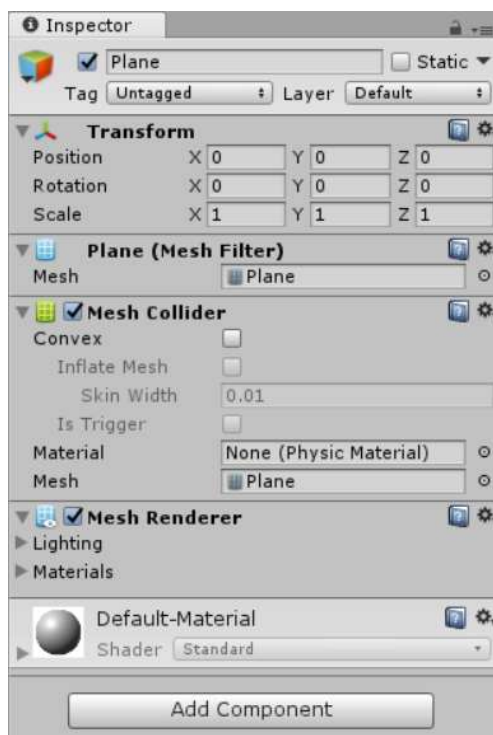


Рисунок 1.8 - Властивості об'єкта в області «Inspector»

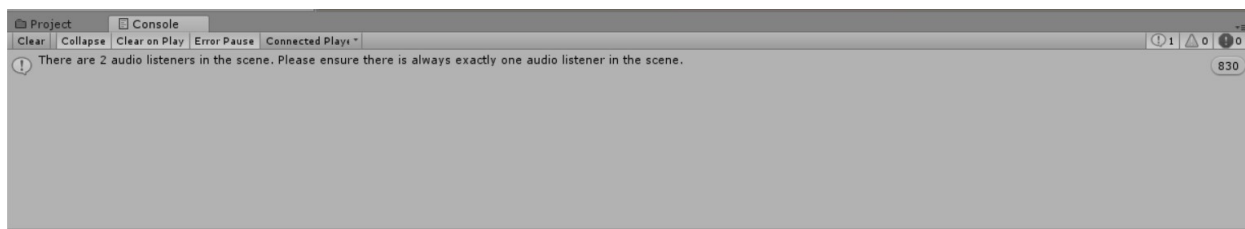


Рисунок 1.9 - Вікно консолі (**Console**)

Налаштування робочого вікна

Можна налаштувати 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.

Сектор може бути додано до однієї зі сторін існуючого вікна.

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

Плаваюче вікно редактора схоже на головне вікно, але не має панелі інструментів (Toolbar).

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

Створення проекту Unity. Імпорт необхідних ресурсів.

Запускаємо Unity.

Обираємо на вкладці **On Disk** пункт **New Unity Project** (рис. 1.10).

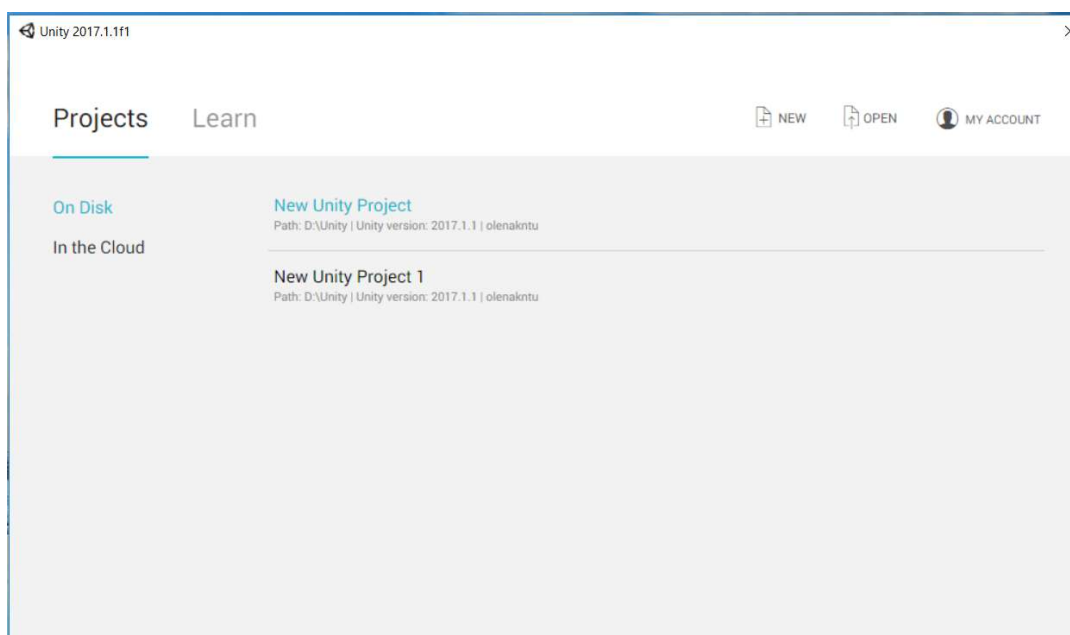


Рисунок 1.10 - Створення нового проекту Unity

Для імпорту необхідних ресурсів (матеріали, текстури) вибираємо меню «Assets → Import package → Custom package» і переходимо в папку з імпортованими ресурсами (рис. 1.11).

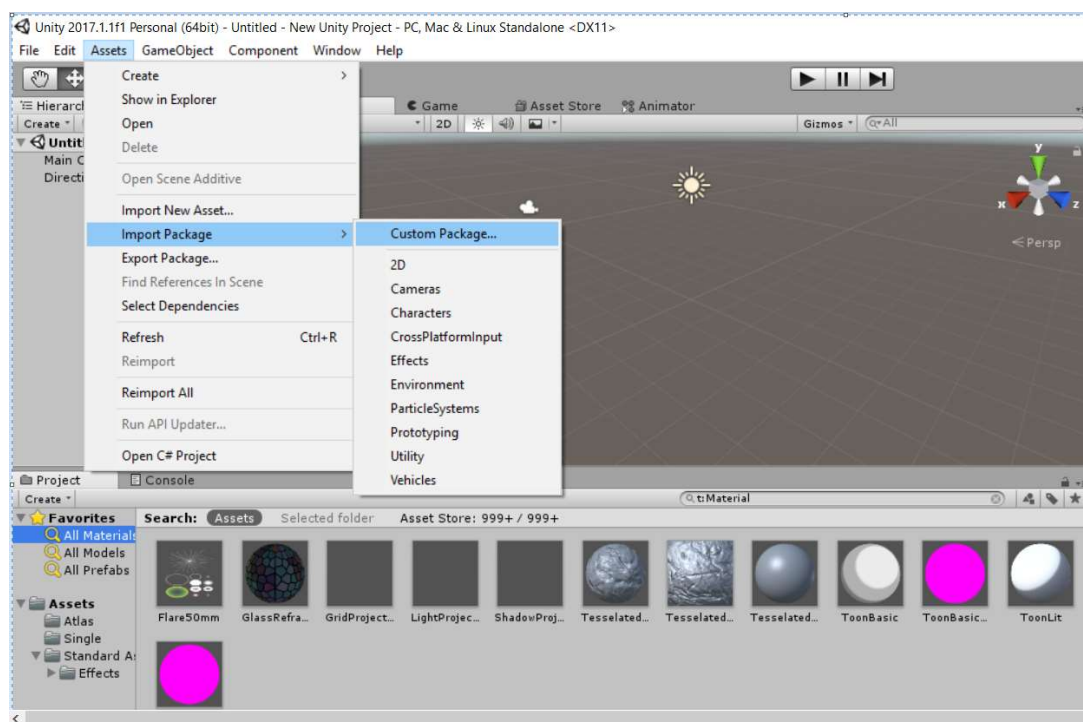


Рисунок 1.11 - Імпорт необхідних ресурсів

6. Генерація ландшафту

Для створення ландшафту в Unity використовується 3D Object «Terrain». Для створення ландшафту необхідно додати в вікно «Scene» 3D Object «Terrain» вибравши меню «Game Object → 3D Object → Terrain» (рис. 1.12).

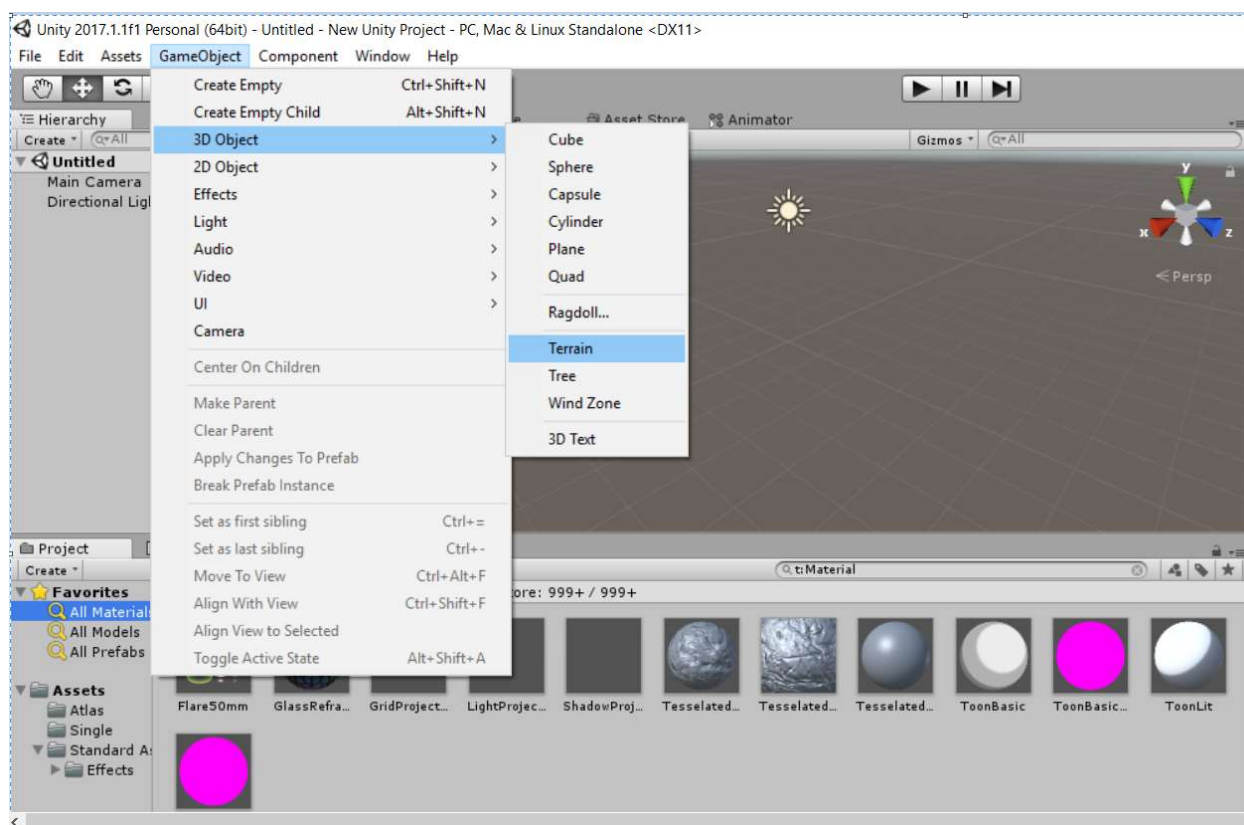


Рисунок 1.12 - Створення 3D Object «Terrain»

При цьому, ландшафт спочатку буде просто плоскою рівниною. Однак, якщо ви подивитесь на інспектор при виділеному об'єкті **Terrain**, ви побачите, що там представлений ряд інструментів, які ви можете використовувати для створення будь-яких потрібних вам ландшафтів (рис. 1.13).

Отже, ландшафт готовий. Задамо йому текстуру. 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**» (рис. 1.14).

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** (рис. 1.15).

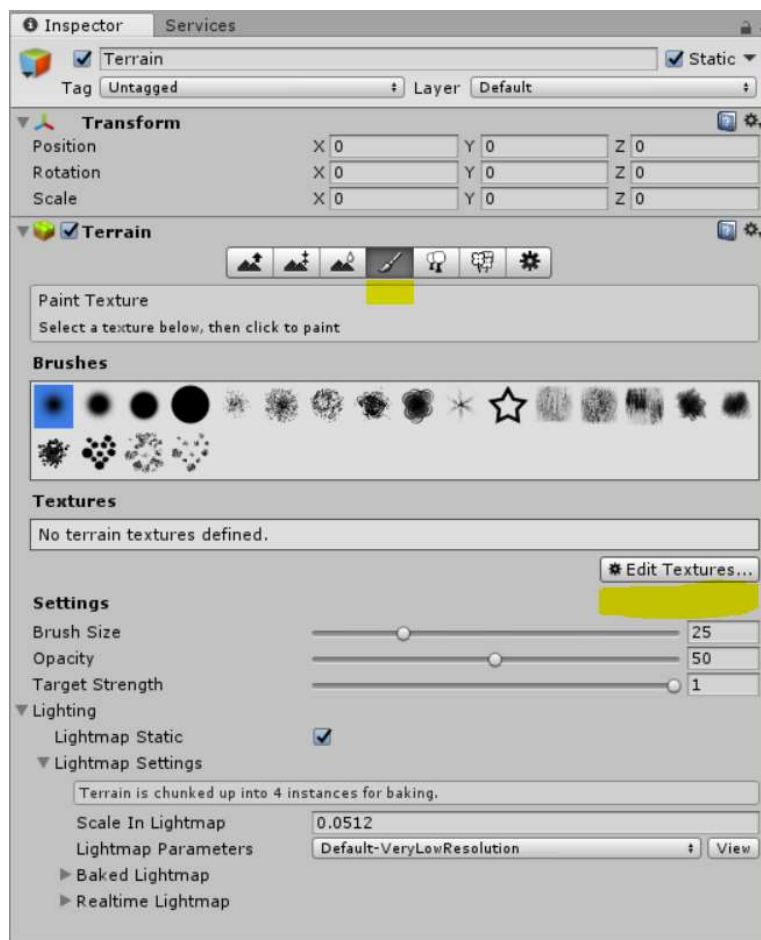


Рисунок 1.13 - Інструменти для створення ландшафтів



Рисунок 1.14 - Вікно додавання текстур до об'єкту **Terrain** «Add Terrain Texture»

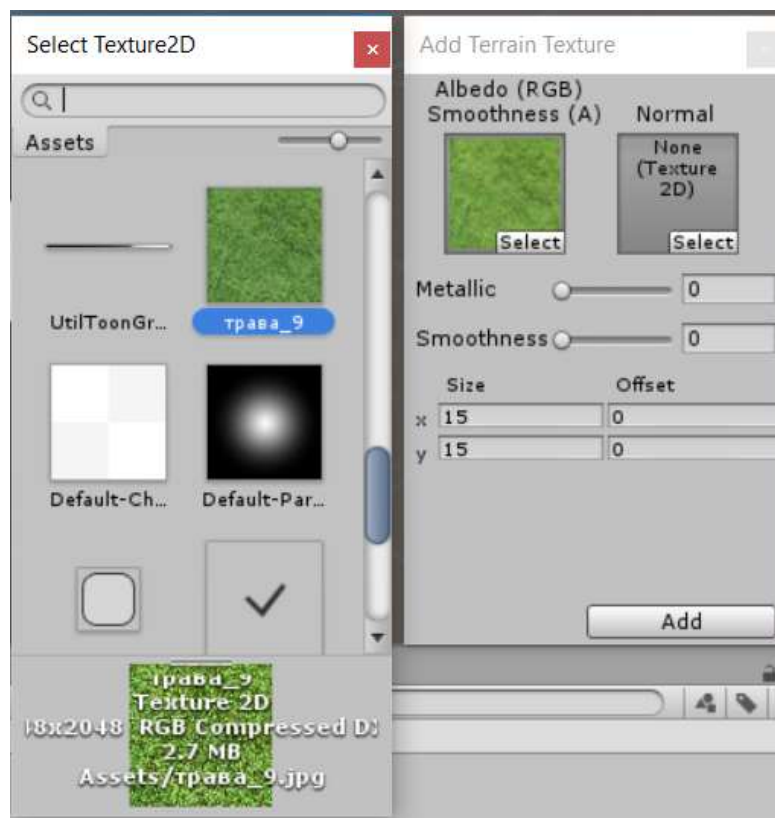


Рисунок 1.15 – Вікно вибору текстур «Select Texture 2D»

В інспекторі об'єктів натискаємо на кнопку «**Raise/Lower terrain**», вибираємо пензлик, встановлюємо розмір (**Brush Size**), інтенсивність (**Opacity**) (рис. 1.16).

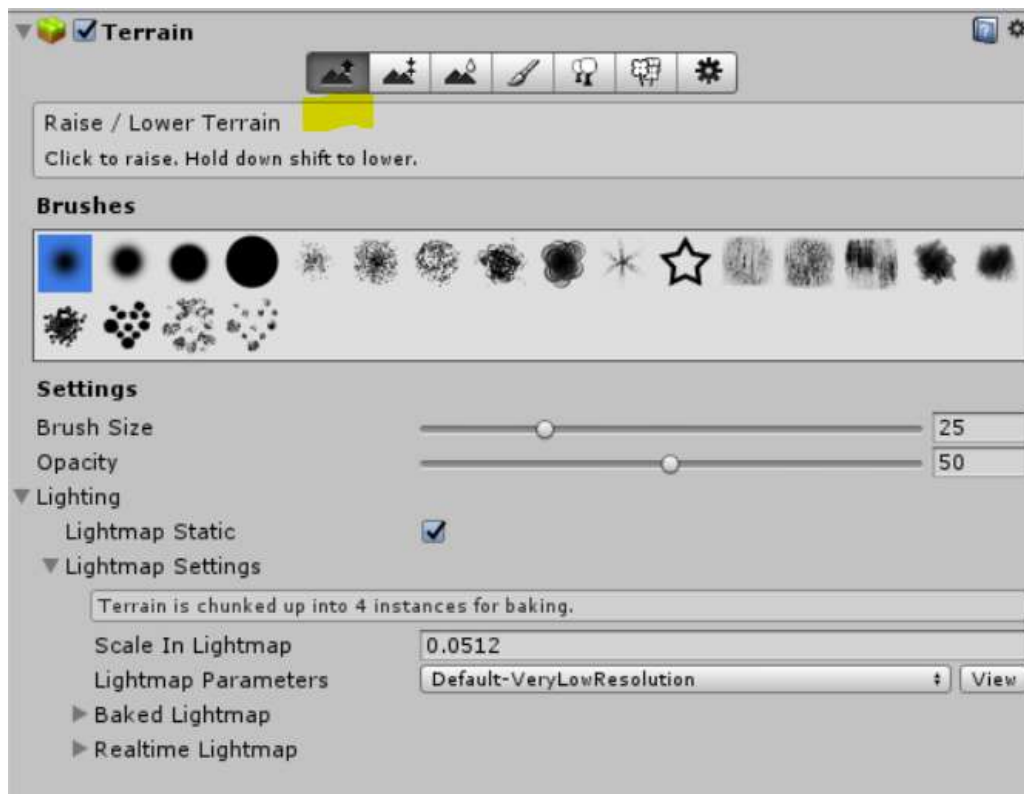


Рисунок 1.16 – Інспектор об'єктів. Кнопка «**Raise/Lower terrain**»

Натискаючи **ЛКМ** піднімаємо вершину, або натискаючи комбінацію **Shift** + **ЛКМ** опускаємо. Натискаючи кнопку «**Smooth Height**» прибираємо гострі кути (рис. 1.17).

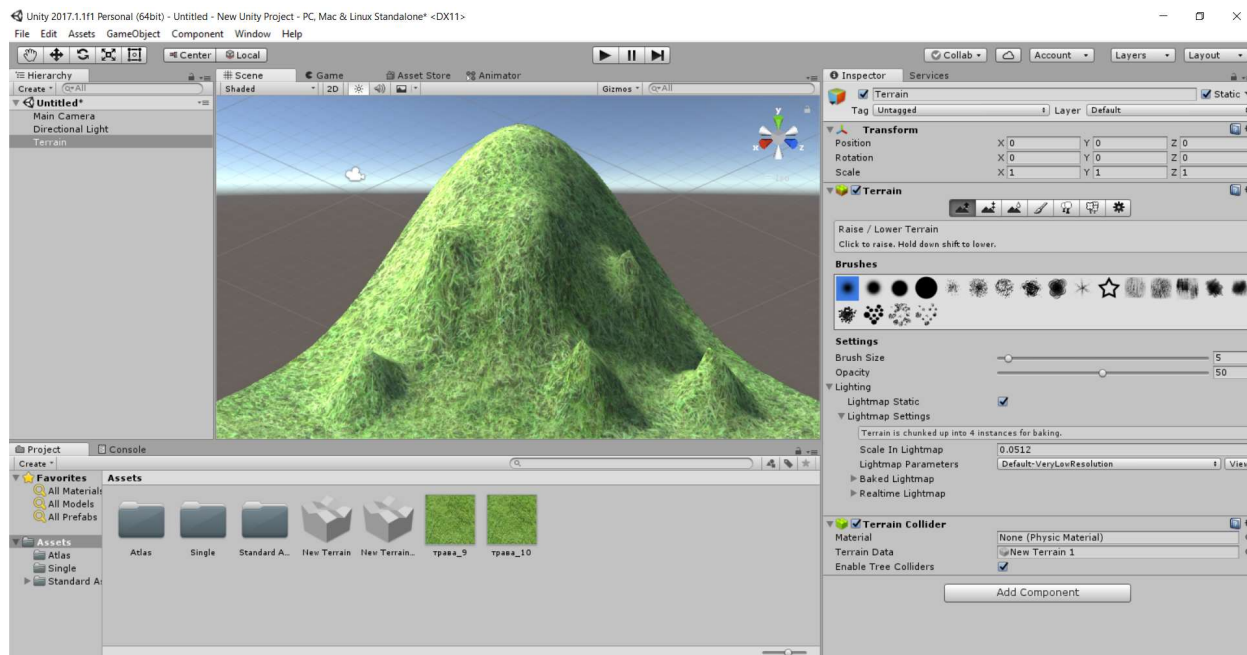


Рисунок 1.17 – Створення об'єкту **Terrain**

7. Робота з освітленням і камерою

Джерела світла використовуються для освітлення сцени та об'єктів.

В Unity розробнику доступні 5 типів джерел світла:

Directional Light. Найпростіше джерело світла, імітує сонячне світло.

Directional Light є нескінченною множиною паралельних променів.

Point Light. Точкове джерело світла, тобто промені розходяться в різні боки з однієї точки.

Spot lights. Джерело світла, що світить з точки в деякому напрямку і висвітлює об'єкти тільки всередині конуса.

Area Light. Джерело світла, що має площу. Уявіть собі прямокутну панель, з якої виходить світло, це і буде **Area Light**.

Light Probe Group. Особливе джерело світла, що впливає виключно на динамічні об'єкти.

Для створення джерела світла виберемо меню «**GameObject→Light**» (рис.1.18).

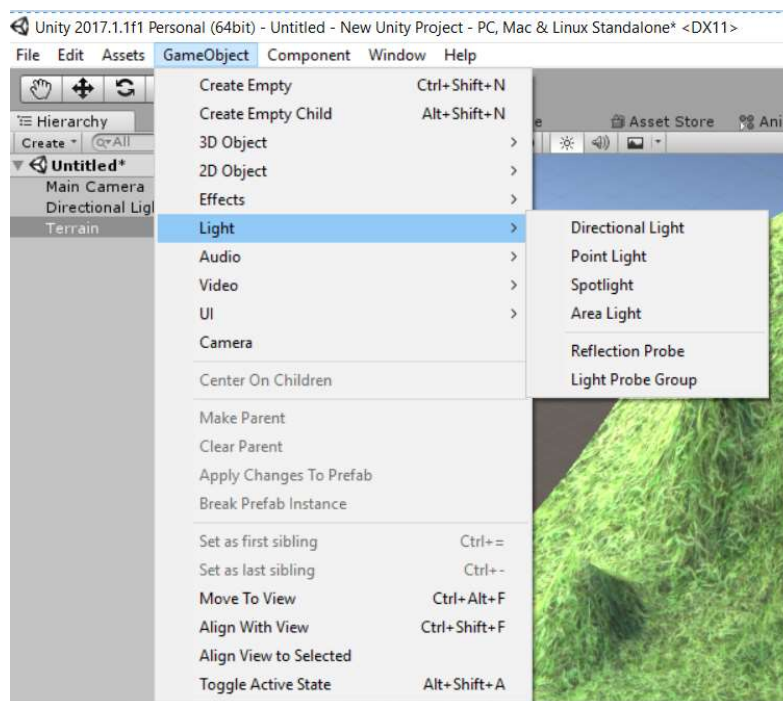


Рисунок 1.18 – Створення джерела світла

Спрямоване світло **Directional Light**.

Джерела спрямованого світла в основному використовуються для сонячного і місячного світла на сценах з відкритим простором. Джерела світла даного типу впливають на поверхні всіх об'єктів сцени. Також це найдешевше джерело світла для графічного процесора.

Для створення джерела світла **Directional Light** виберемо меню «**GameObject→Light→Directional Light**» (рис.1.19).

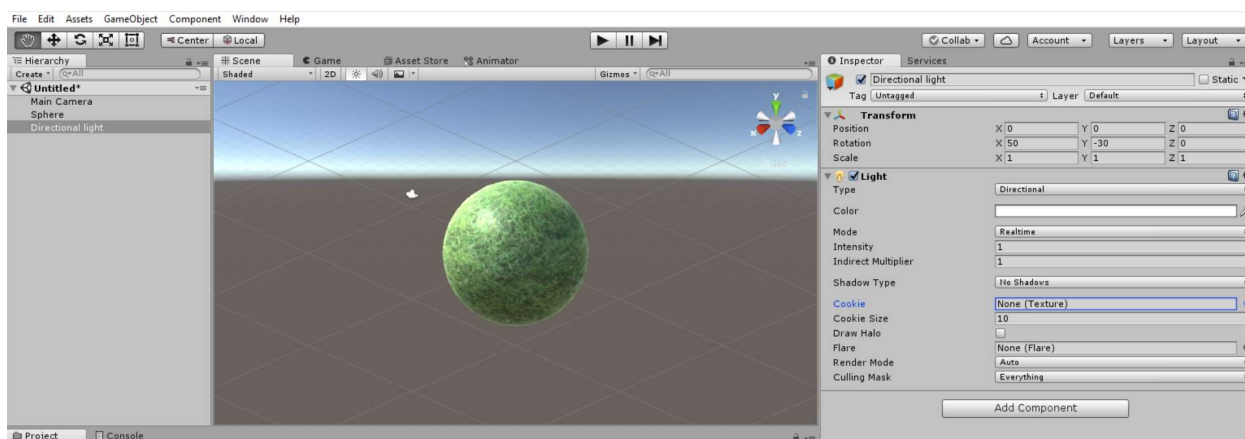


Рисунок 1.19 – Створення джерела світла **Directional Light**

Точкове освітлення (Point Lights)

Point light світить на всі боки з однієї точки. Це найпоширеніше джерело світла в іграх. Даний тип освітлення має середню ресурсовитратність для графічного процесора.

Для створення джерела світла **Point Lights** виберемо меню «**GameObject**→**Light**→**Point Lights**» (рис.1.20).

В інспекторі на вкладці «**Lights**» встановимо параметр «**Range=60**», параметр «**Color**» змінимо на «**Red**», встановимо параметр «**Intensity=50**».

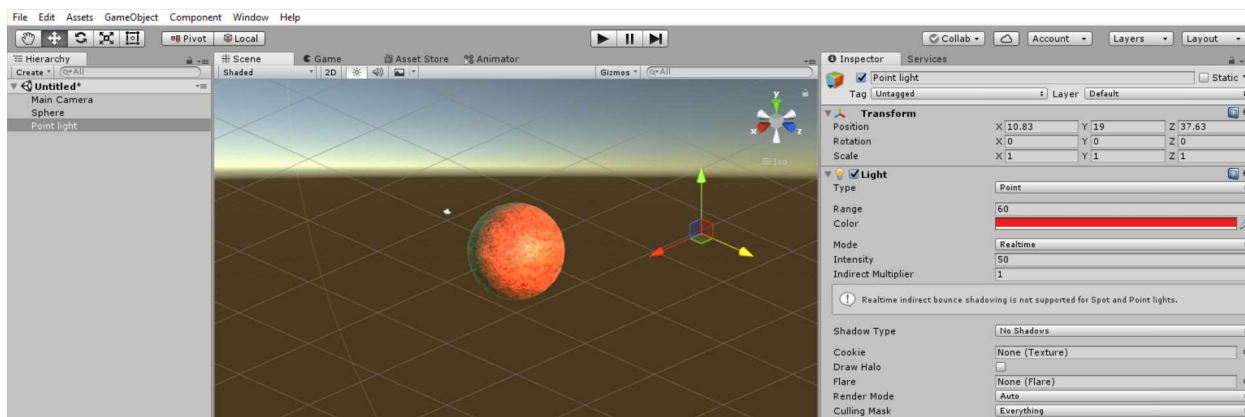


Рисунок 1.20 – Створення джерела світла **Point Lights**

Освітлення променем (Spot Lights)

Джерело світла **Spot Lights** світить в одному напрямку, всередині конуса. Джерела світла даного типу ідеально підходять для ручних ліхтариків, автомобільних фар або ліхтарних стовпів. Даний тип освітлення є ресурсозатратним.

Для створення джерела світла **Spot Lights** виберемо меню «**GameObject**→**Light**→**Spot Lights**» (рис.1.21).

В інспекторі на вкладці «**Lights**» встановимо параметр «**Range=60**», параметр «**Color**» змінимо на «**Red**», встановимо параметр «**Intensity=60**».

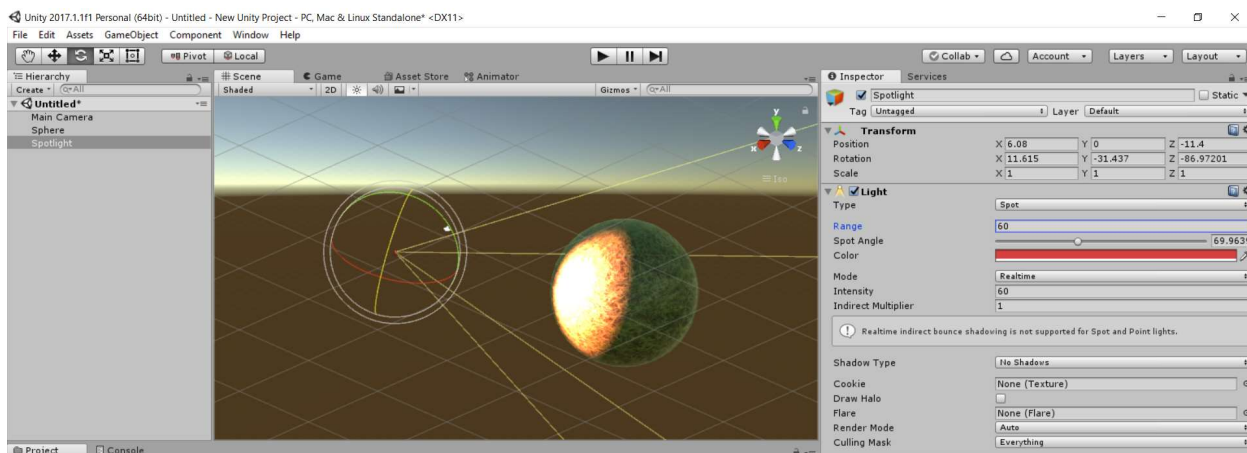


Рисунок 1.21 – Створення джерела світла **Spot Lights**

Плоске джерело світла (**Area Lights**)

Плоске джерело світла випромінює світло однієї зі сторін прямокутної області площині. Даний тип джерела світла впливає на об'єкти всередині області дії. Розмір прямокутника задається властивостями «**Width**» і «**Height**». Світло випромінюється з усією поверхні прямокутної області. Оскільки **Area Lights** вимагає великих обчислень процесора, плоскі джерела світла недоступні під час гри. Тому можливе тільки запікання в світлові карти.

Для створення джерела світла **Area Lights** виберемо меню «**GameObject**→**Light**→**Area Lights**» (рис.1.22).

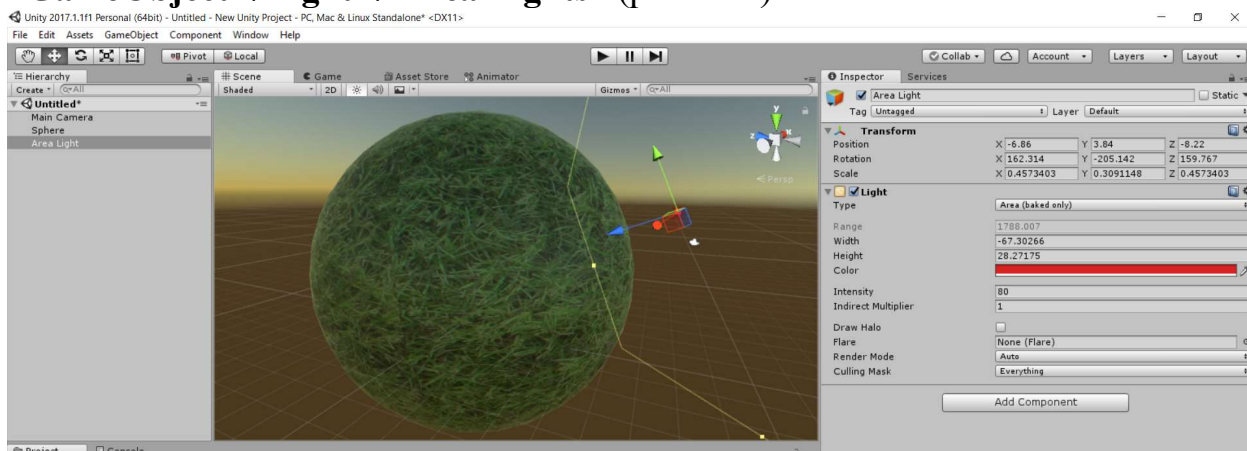


Рисунок 1.22 – Створення джерела світла **Area Lights**

Камери є пристроями, які захоплюють і відображають ігровий світ. Ви можете мати необмежену кількість камер в сцені та налаштувати рендеринг камерами в будь-якому порядку, на будь-якому місці екрану, або тільки в певних його частинах.

Для створення камери виберемо меню «**GameObject**→**Camera**» (рис.1.23).

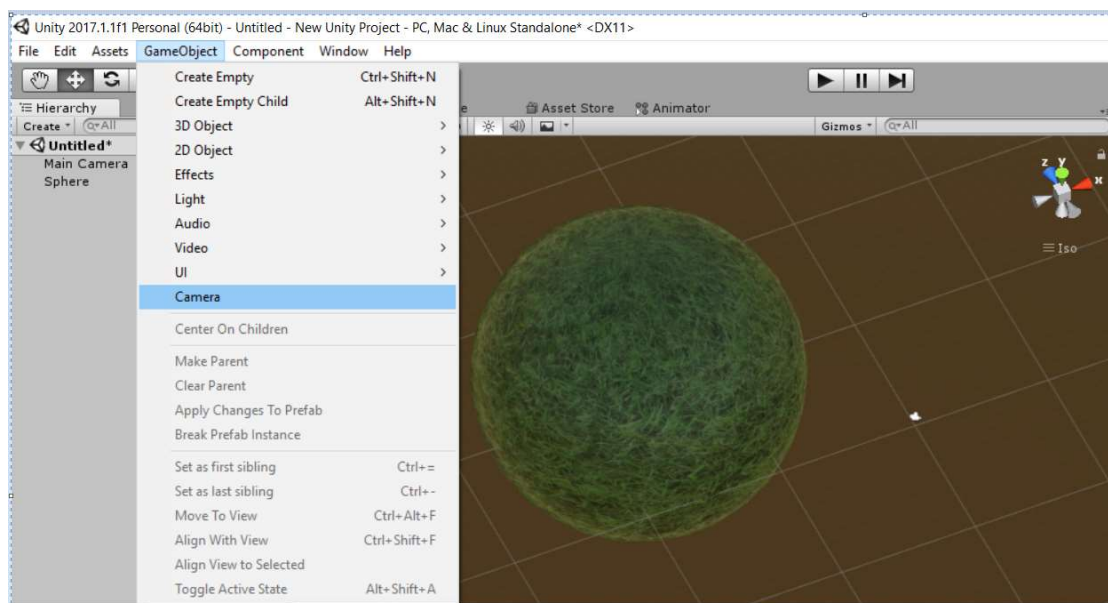


Рисунок 1.23 – Меню «**GameObject**→**Camera**»

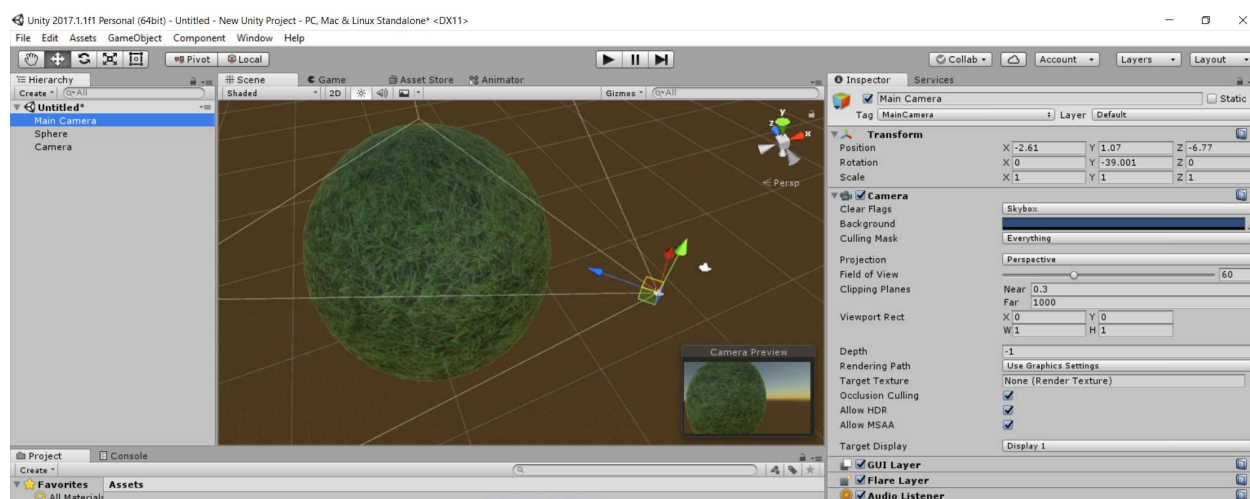


Рисунок 1.24 – Створення камери

Таблиця 2 – Властивості об'єкта «Camera»

Властивість	Функція
Clear Flags	Визначає, які частини екрану будуть очищені. Це зручно при використанні декількох камер для відтворення різних елементів гри.
Background	Колір, який застосовується для фону після відтворення всіх елементів, в разі відсутності скайбокса.
Culling Mask	Включення або виключення шарів об'єктів на рендер цією камерою. Призначення шарів об'єктів проводиться через Inspector.
Projection	Перемикає здатність камери симулювати перспективу. Perspective Камера буде рисувати об'єкти в перспективі. Orthographic Камера буде рисувати об'єкти рівномірно, без ефекту перспективи.
Size (коли обраний орографічний режим)	Розмір зони видимості камери для орографічного режиму.
Field of view (коли обраний режим перспективи)	Ширина кута огляду камери, вимірюється в градусах по локальній осі Y.
Clipping Planes	Дистанція, на якій камера починає і закінчує рендеринг. Near Найближча точка відносно камери, яка буде рисуватися. Far Дальня точка відносно камери, яка буде рисуватися.
Normalized View Port Rect	Чотири значення, що відображають те, в якій області екрану буде виведено зображення з камери, в екранних координатах (від 0 до 1). X Початкова позиція області по горизонталі виду камери, який буде рисуватися. Y Початкова позиція області по вертикалі, де вид камери буде рисуватися. W (Ширина) Ширина виду камери на екрані. H (Висота)

	Висота виду камери на екрані.
Depth	Позиція камери в черзі на прорисовування. Камери з великим значенням будуть нарисовані поверх камер з меншим значенням.
Rendering Path	Опції для визначення методів рендеринга для камери. Use Player Settings Камера використовує метод візуалізації, встановлений в Player Settings. Vertex Lit Всі об'єкти, що рисуються цією камерою будуть рендеритися як Vertex-Lit-об'єкти. Forward Всі об'єкти будуть рендеритися з одним проходом на матеріал
HDR	Включення технології High Dynamic Range.

Завдання до лабораторної роботи №1

1. Створити новий проект Unity.
2. Здійснити налаштування інтерфейсу.
3. Здійснити імпорт ресурсів (матеріалів, текстур).
4. За допомогою 3D-об'єкту «Terrain» згенерувати ландшафт та додати текстуру ландшафту.
5. За допомогою меню «Game Object→3D Object→Sphere» створити 3D-об'єкт «Sphere» та прив'язати до нього 4 типи джерел світла (Directional Light, Point Light, Spot lights, Area Light).
6. За допомогою меню «Game Object→Camera» створити об'єкт «Camera» та прив'язати його до об'єкту «Sphere».
7. Здійснити рендеринг.

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 поняттю «рендеринг».

ЛАБОРАТОРНА РОБОТА № 2

ТЕМА: «КОМПОНЕНТИ UNITY 3D ДЛЯ РОБОТИ З ФІЗИКОЮ 3D-ОБ'ЄКТІВ»

Анотація

Лабораторна робота орієнтована на оволодіння студентами навичок роботи з основними компонентами, 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 дозволити 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.

Компоненти для роботи з фізикою 3D - об'єктів

Для того, щоб фізична поведінка об'єктів в ігровому додатку була реалістичною, об'єкт у грі потрібно правильно прискорити і задіяти зіткнення, гравітацію та інші сили. Вбудовані в Unity фізичні движки забезпечують вас компонентами для обробки симуляції фізики. За допомогою налаштування всього декількох параметрів, можна створити об'єкти, які поведуть себе пасивно реалістично (тобто вони будуть переміщатися в результаті зіткнень і падінь, але не почнуть рухатися самі по собі). Керуючи фізикою за допомогою скриптів, ви можете придати об'єкту динаміку.

Unity має два окремих фізичних движка, один для 3D-фізики і один для 2D-фізики. Основні поняття ідентичні в обох движках (за винятком додаткового виміру в 3D), але вони реалізовані з різними компонентами. Так, наприклад, існує компонент Rigidbody для 3D фізики і аналогічний Rigidbody 2D для 2D фізики.

Розглянемо далі основні компоненти для роботи з фізикою 3D - об'єктів.

Rigidbody (Тверде тіло)

Rigidbody - це основний компонент, що включає фізичну поведінку для об'єкта. Якщо додати компонент Rigidbody до об'єкту, то він негайно почне реагувати на гравітацію.

Створіть новий проект або нову сцену в готовому проекті.

Додайте 3D-об'єкт **Sphere** за допомогою меню «**GameObject**→**3D Object** →**Sphere**», 3D-об'єкт **Plane** за допомогою меню «**GameObject**→**3D Object**→**Plane**» і точкове джерело світла **Point Light** за допомогою меню «**GameObject**→**Light**→ **Point Light**» (рис.2.1).

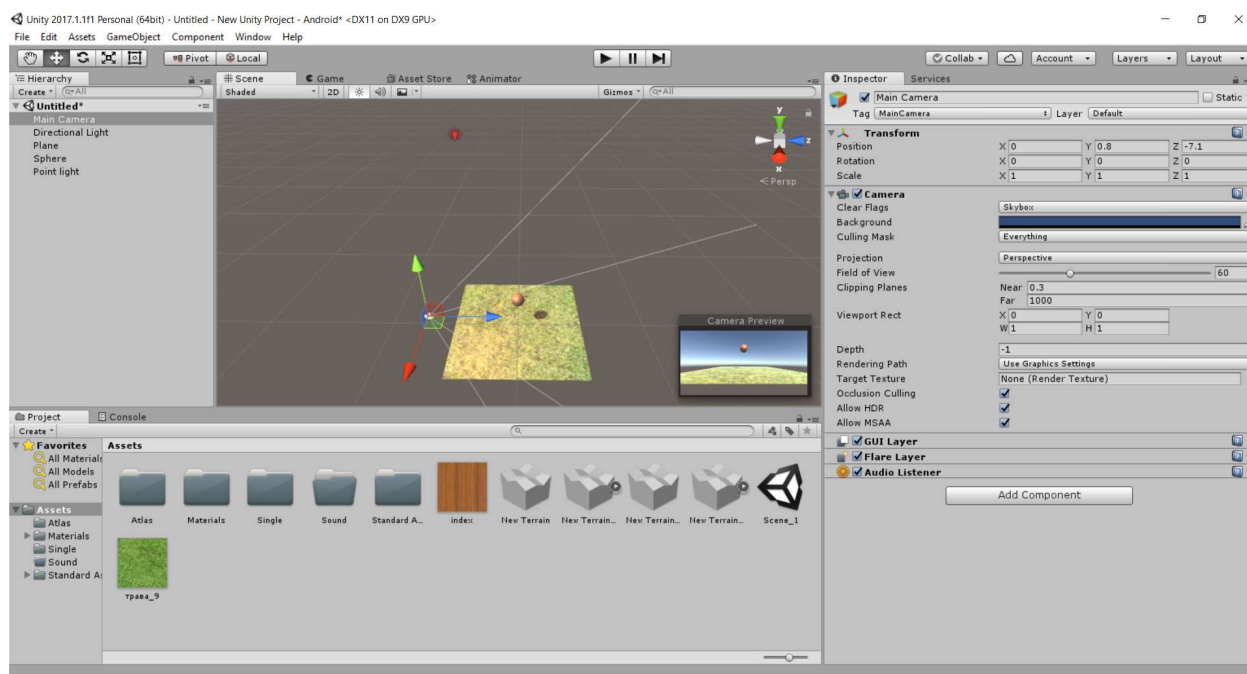


Рисунок 2.1 – Створення 3D-об'єктів

Виберіть 3D-об'єкт **Sphere** на панелі «**Hierarchy**». Після чого на панелі «**Inspector**» буде відкрито доступ до властивостей 3D-об'єкту **Sphere**. За допомогою кнопки «**Add Component**→**Physics**→ **Rigidbody**» додайте компонент **Rigidbody** до об'єкту **Sphere**. Доступ до властивостей компоненту **Rigidbody** також буде відкрито на панелі «**Inspector**» (рис.2.2).

Якщо натиснути на кнопку «**Play**», ви побачите як 3D-об'єкт **Sphere** падає на 3D-об'єкт **Plane** під дією сили тяжіння (рис.2.3).

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.

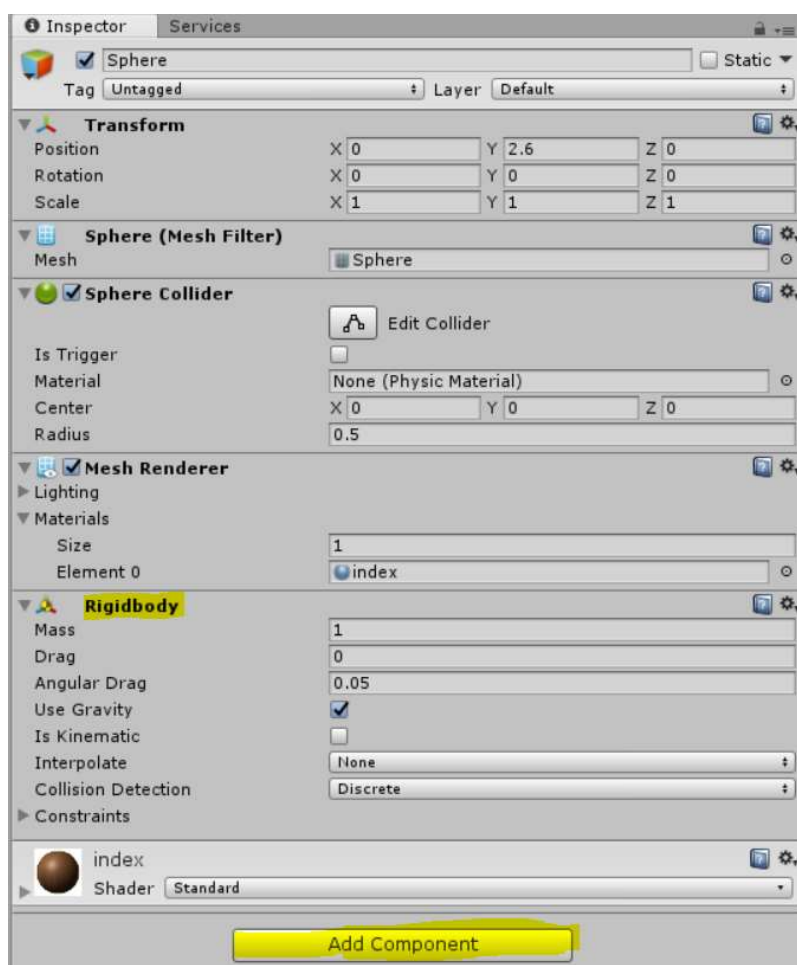


Рисунок 2.2 – Властивості компоненту **Rigidbody**

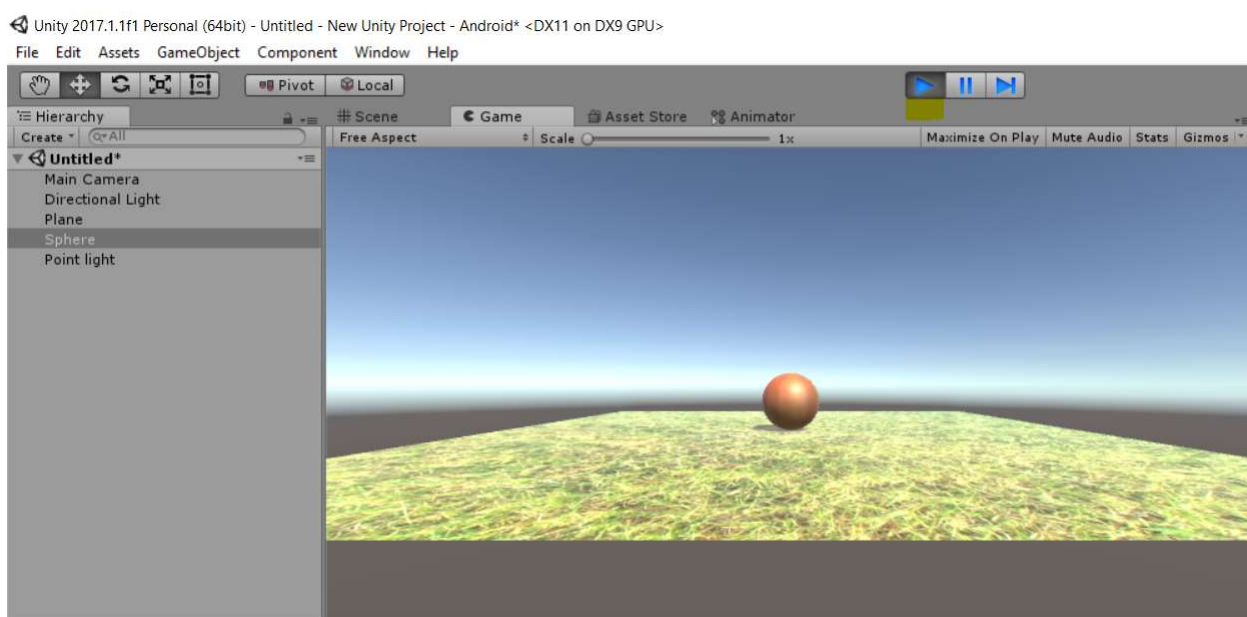


Рисунок 2.3 – Кнопка «Play»

Властивості компоненту **Rigidbody** подано у табл. 2.1.

Таблиця 2.1 - Властивості компоненту **Rigidbody**

Властивість	Функція
Mass	Маса об'єкта (за замовчуванням в кілограмах).
Drag	Повітряний опір, який діє на об'єкт поки він переміщується під впливом цих сил. 0 означає відсутність опору, а нескінченність (infinity) припиняє переміщення об'єкта.
Angular Drag	Повітряний опір, який діє на об'єкт поки він обертається під впливом сили обертання. 0 означає відсутність опору. Слід зазначити, що не можна зупинити обертання об'єкта шляхом установки його кутового опору (Angular Drag) в нескінченне (infinity) становище.
Use Gravity	При включенні на об'єкт діє гравітація.
Is Kinematic	При включенні вилучає об'єкт з-під контролю фізичного движка, і дозволяє переміщати його кінематично за допомогою скрипта.
Interpolate	Застосовується, якщо помічено тряску в переміщенні твердого тіла. - None Не застосовано жодної інтерполяції. - Interpolate Згладжування трансформації, що засноване на трансформації з попереднього кадру. - Extrapolate Згладжування трансформації, що засноване на приблизної трансформації наступного кадру.
Collision Detection	Використовується для запобігання проникнення об'єктів, які швидко рухаються крізь інші об'єкти без визначення зіткнень. - Discrete Для всіх колайдерів сцени необхідно використовувати Discrete-виявлення зіткнень. Також колайдери використовуватимуть Discrete-виявлення зіткнень при перевірці на зіткнення проти них. Значення Discrete встановлено за умовчанням. - Continuous

	<p>Використовується для виявлення зіткнень проти динамічних колайдерів (з твердими тілами) і безперервне виявлення зіткнень проти статичних меш колайдерів (без твердих тіл).</p> <p>Тверді тіла, встановлені як Continuous Dynamic будуть використовувати Continuous-виявлення зіткнень при тестуванні на зіткнення проти них. Інші тверді тіла також будуть використовувати Continuous-виявлення зіткнень.</p> <p>Використовується для об'єктів, з якими потрібно зіткнутися за допомогою динамічного виявлення зіткнень. Все це дуже впливає на фізичну продуктивність, тому залиште дане значення встановленим в Discrete, якщо не відчуваєте проблем із зіткненням об'єктів, які швидко рухаються.</p> <p>- Continuous Dynamic</p> <p>Використовується безперервне і безперервне динамічне виявлення зіткнень проти об'єктів налаштованих на безперервне і безперервне динамічне зіткнення. Також буде використовуватися безперервне виявлення зіткнень проти статичних меш колайдерів (без твердих тіл). Для всіх інших колайдерів буде використовуватися Discrete-виявлення зіткнень.</p> <p>Використовується для об'єктів, які швидко рухаються.</p>
Constraints	<p>Обмеження руху твердого тіла</p> <p>- Freeze Position</p> <p>Вибірково зупиняє переміщення твердого тіла по осях X, Y і Z.</p> <p>- Freeze Rotation</p> <p>Вибірково зупиняє обертання твердого тіла по осях X, Y і Z.</p>

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.

Давайте трохи змінимо фізичні властивості 3D-об'єкту **Sphere**. Знову виберіть 3D-об'єкт **Sphere**. На панелі «**Inspector**» за допомогою кнопки «**Add**

Component→**Physics**→ **Box Collider**» додайте компонент **Box Collider** до об'єкту **Sphere**. Доступ до властивостей компоненту **Box Collider** також буде відкрито на панелі «**Inspector**» (рис.2.4).

Box Collider базовий кубічний примітив зіткнень.

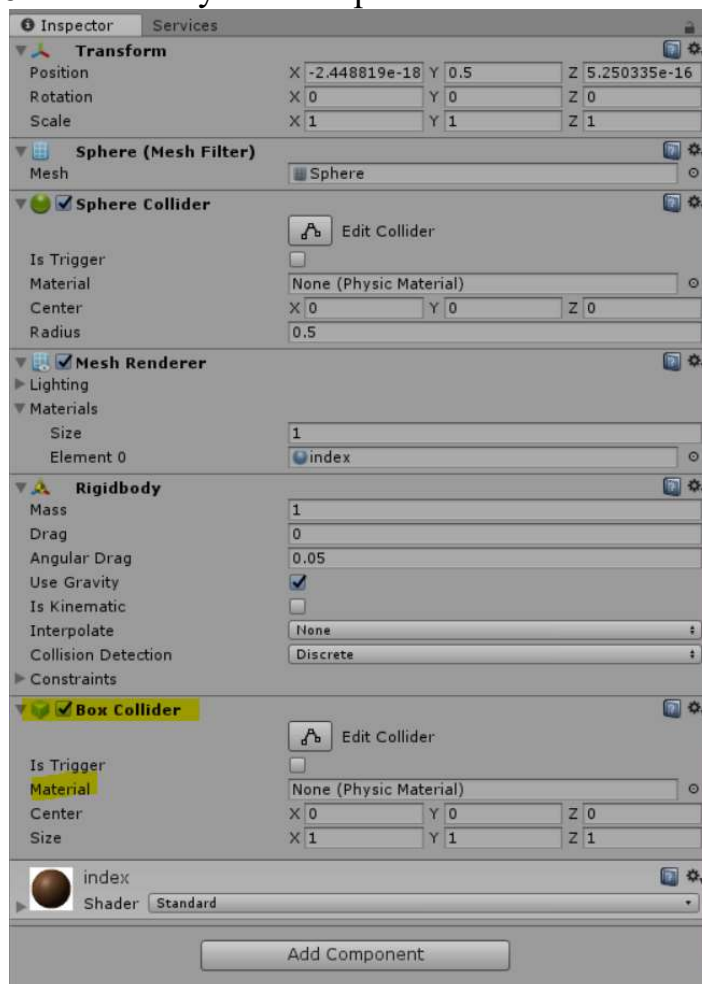


Рисунок 2.4 – Компоненту **Box Collider**

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

Таблиця 2.2 - 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.

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

Для того, щоб створити новий фізичний **Material** скористайтеся меню «**Assets**→**Create**→**Physic Material**» (рис. 2.5).

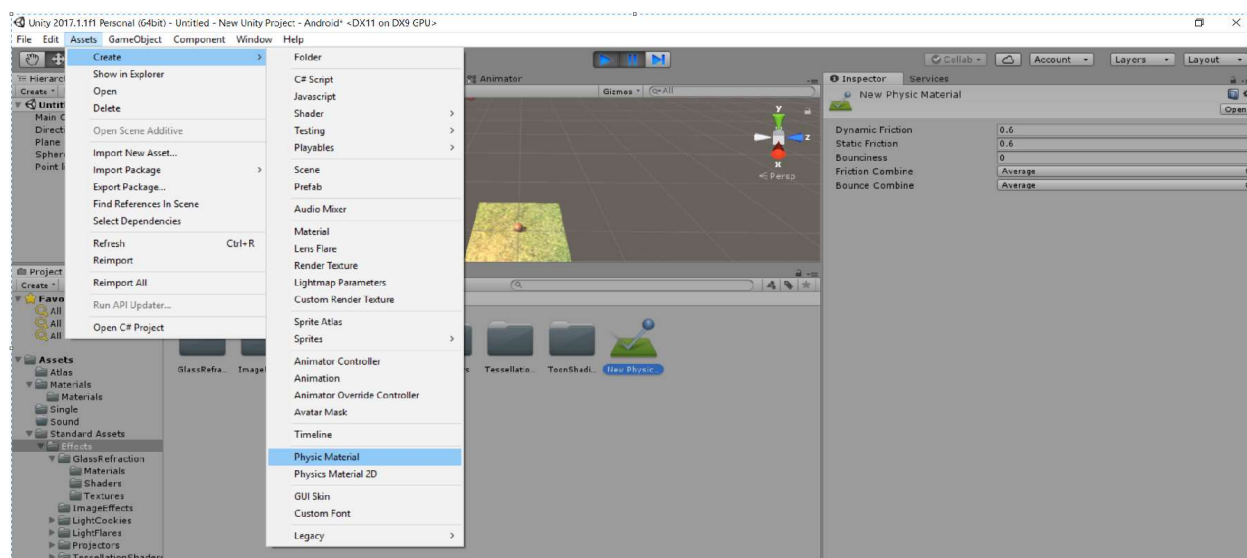


Рисунок 2.5 – Створення нового **Physics Material**

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 слабку 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 означатиме 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.

	<ul style="list-style-type: none"> - 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.

Здайте значення **Physics Material**, як показано на рис. 2.6.

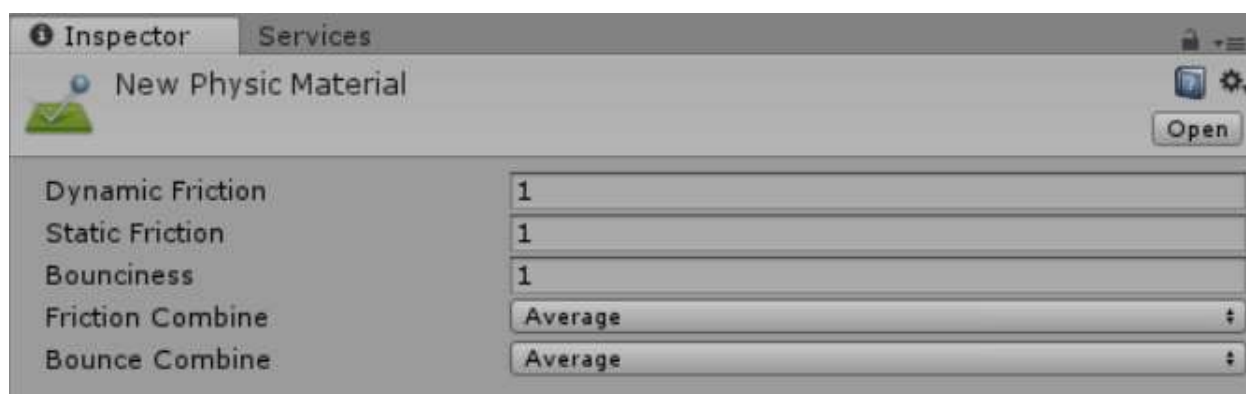


Рисунок 2.6 – Властивості **Physics Material**

Додайте щойно створений **Physics Material** до компоненту **Box Collider**. Якщо натиснути на кнопку «**Play**», ви побачите як після падіння 3D-об'єкт **Sphere** почне відскакувати від поверхні (3D-об'єкт **Plane**).

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. 2.4.

Таблиця 2.4 - 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 **Rigidbody** на об'єкти, які потрібно з'єднати за допомогою **Fixed Joints**.

Додайте до 3D-об'єкту **Sphere** компонент **Fixed Joints**. На панелі «**Hierarchy**» виберіть 3D-об'єкт **Sphere**. На панелі «**Inspector**» за допомогою кнопки «**Add Component**→**Physics**→**Fixed Joint**» додайте компонент **Fixed Joint** до об'єкту **Sphere**. Доступ до властивостей компоненту **Fixed Joint** також буде відкрито на панелі «**Inspector**» (рис.2.7).

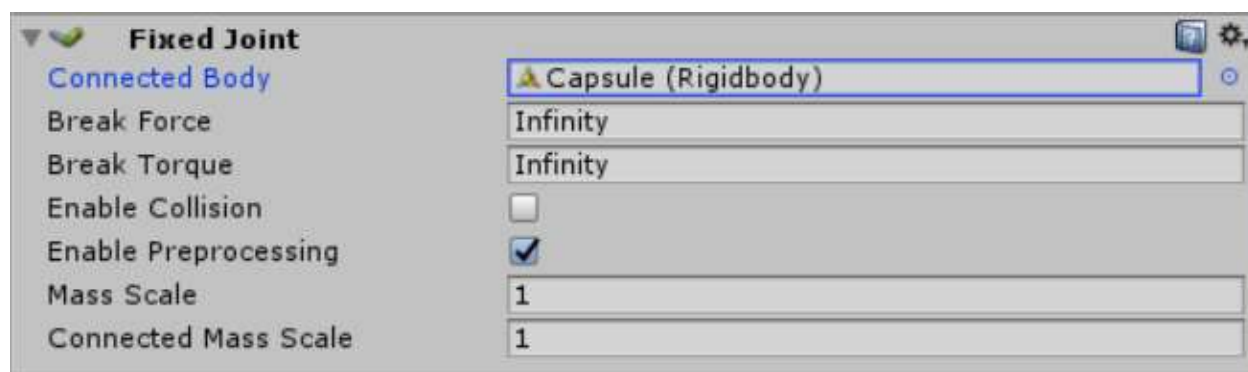


Рисунок 2.7 – Компонент **Fixed Joint**

З'єднайте 3D-об'єкт **Sphere** з будь-яким іншим компонентом на сцені за допомогою властивості **Connected Body** (в нашому випадку це 3D-об'єкт **Capsule**).

Компонент **Fixed Joints** обмежить рух 3D-об'єкту **Sphere**, з'єднавши його з іншим 3D-об'єктом **Capsule**.

4.4 Spring Joint (Пружне з'єднання)

Spring Joint з'єднує пружиною два Rigidbody-об'єкта.

Властивості **Spring Joint** подано у табл. 2.5.

Таблиця 2.5 - 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'єданого 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.

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.2.6).

Таблиця 2.6 - 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.

Додайте 3D-об'єкт **Capsule** за допомогою меню «**GameObject**→**3D Object** → **Capsule**», 3D-об'єкт **Plane** за допомогою меню «**GameObject**→**3D Object**→**Plane**».

На панелі «**Inspector**» ви побачите, що об'єкту **Capsule** призначено капсульний колайдер (рис.2.8).

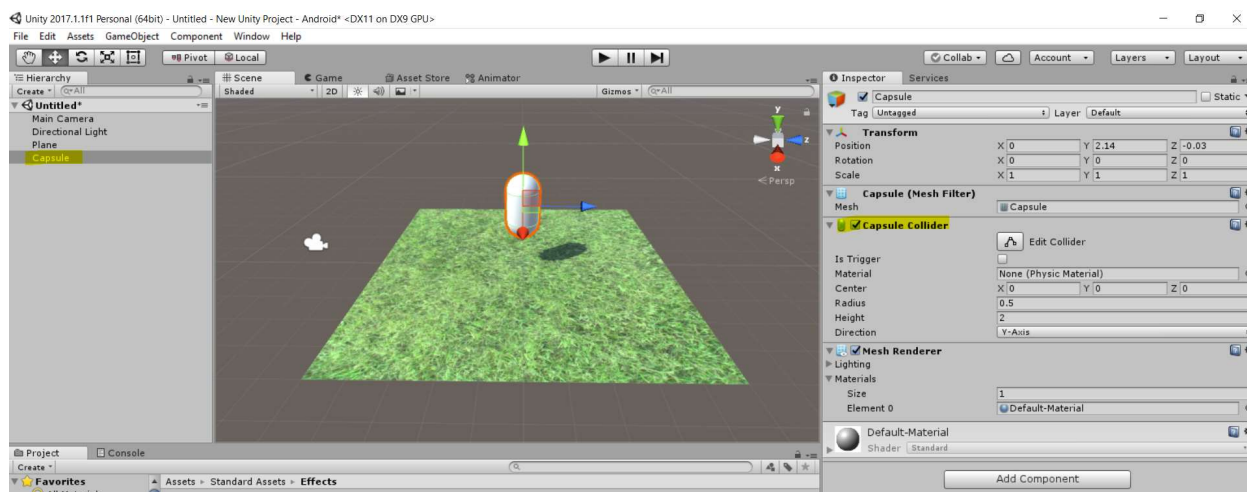


Рисунок 2.8 – 3D-об'єкт Capsule

Капсульний колайдер необхідно видалити. Для цього натисніть на значок із зображенням шестерні праворуч від імені компонента, як показано на рис. 2.9. Відкриється меню, в якому ви знайдете команду **Remove Component**.

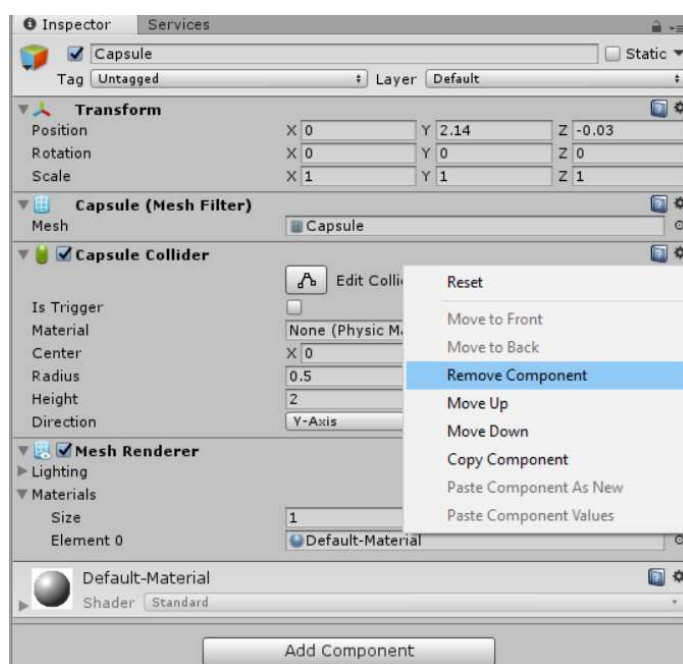


Рисунок 2.9 – Команда **Remove Component**

Замість капсульного колайдера призначимо об'єкту компонент **Character Controller**. На панелі «Inspector» ви знайдете кнопку «Add Component», за допомогою якої можна відкрити меню з переліком типів доступних компонентів. У розділі **Physics** знаходиться потрібний нам компонент **Character Controller**. Призначимо його 3D-об'єкту **Capsule** (рис.2.10).

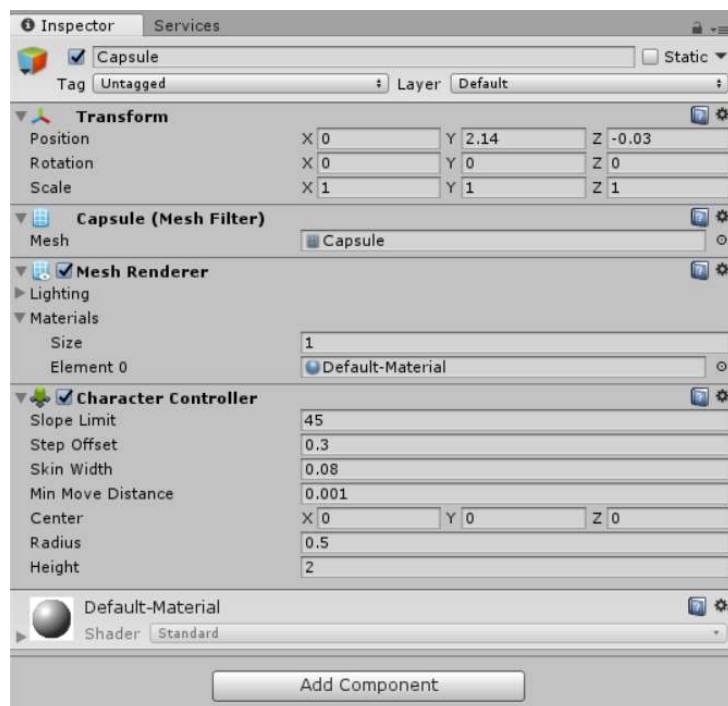


Рисунок 2.10 – Компонент **Character Controller**

Компонент **Character Controller** забезпечить природне переміщення 3D-об'єкту **Capsule**.

Для того, щоб переміщати 3D-об'єкт, який володіє компонентом **Character Controller**, необхідно створити C# -скрипт в даному випадку з ім'ям **ExampleClass**.

На панелі **Project** в папці **Assets** за допомогою контекстного меню «**Create**→**C# Script**» створіть файл C# -скрипта з ім'ям **ExampleClass**.

Приклад C# -скрипта для переміщення 3D-об'єкту подано у лістингу 2.1.

За допомогою кнопки «**Open**» на панелі «**Inspector**» завантажте Visual Studio та відредагуйте C# -скрипт **ExampleClass** згідно лістингу 2.1.

Додати створений скрипт до 3D-об'єкту з компонентом **Character Controller** можна шляхом перетягування скрипта на об'єкт в сцені.

Лістинг 2.1 Переміщення компоненту **Character Controller**

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public float speed = 6.0F;
    public float jumpSpeed = 8.0F;
    public float gravity = 20.0F;
    private Vector3 moveDirection = Vector3.zero;
    void Update() {
```

```
CharacterController controller =  
GetComponent<CharacterController>();  
if (controller.isGrounded) {  
    moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0,  
Input.GetAxis("Vertical"));  
    moveDirection = transform.TransformDirection(moveDirection);  
    moveDirection *= speed;  
    if (Input.GetButton("Jump"))  
        moveDirection.y = jumpSpeed;  
}  
moveDirection.y -= gravity * Time.deltaTime;  
controller.Move(moveDirection * Time.deltaTime);  
}
```

Натисніть на кнопку «**Play**», щоб побачити результат роботи C# - скрипта. Переміщати 3D-об'єкт **Capsule** можна за допомогою стрілок.

Завдання до лабораторної роботи №2

1. Створіть новий проект Unity.
2. Додайте 3D-об'єкт **Sphere** за допомогою меню «**GameObject→3D Object →Sphere**», 3D-об'єкт **Plane** за допомогою меню «**GameObject→3D Object→Plane**» і точкове джерело світла **Point Light** за допомогою меню «**GameObject→Light→ Point Light**».
3. Додайте компонент **Rigidbody** до об'єкту **Sphere**.
4. Натисніть на кнопку «**Play**», щоб побачити як 3D-об'єкт **Sphere** падає на 3D-об'єкт **Plane** під дією сили тяжіння.
5. За допомогою кнопки «**Add Component→Physics→ Box Collider**» додайте компонент **Box Collider** до об'єкту **Sphere**.
6. Створіть новий фізичний **Material** за допомогою меню «**Assets→Create→Physic Material**». Задайте всі необхідні властивості фізичного матеріалу.
7. Додайте створений **Physics Material** до компоненту **Box Collider**.
8. Натисніть на кнопку «**Play**», щоб побачити як після падіння 3D-об'єкт **Sphere** почне відскакувати від поверхні (3D-об'єкт **Plane**).
9. Створіть новий проект Unity.
10. Додайте 3D-об'єкт **Capsule** за допомогою меню «**GameObject→3D Object → Capsule**», 3D-об'єкт **Plane** за допомогою меню «**GameObject→3D Object→Plane**».

11. Додайте до 3D-об'єкт **Capsule** компонент **Character Controller**.
12. Створіть C# -скрипт для переміщення 3D-об'єкту **Capsule**.
13. Натисніть на кнопку «**Play**», щоб побачити результат роботи C# -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?

ЛАБОРАТОРНА РОБОТА № 3

ТЕМА: «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.

Налаштування 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.3.1).

Переконайтеся, що в **BuildSettings** цільової платформи (меню: **File**→**Build Settings**) включені опції **Development Build** і **Script Debugging** (рис.3.2).

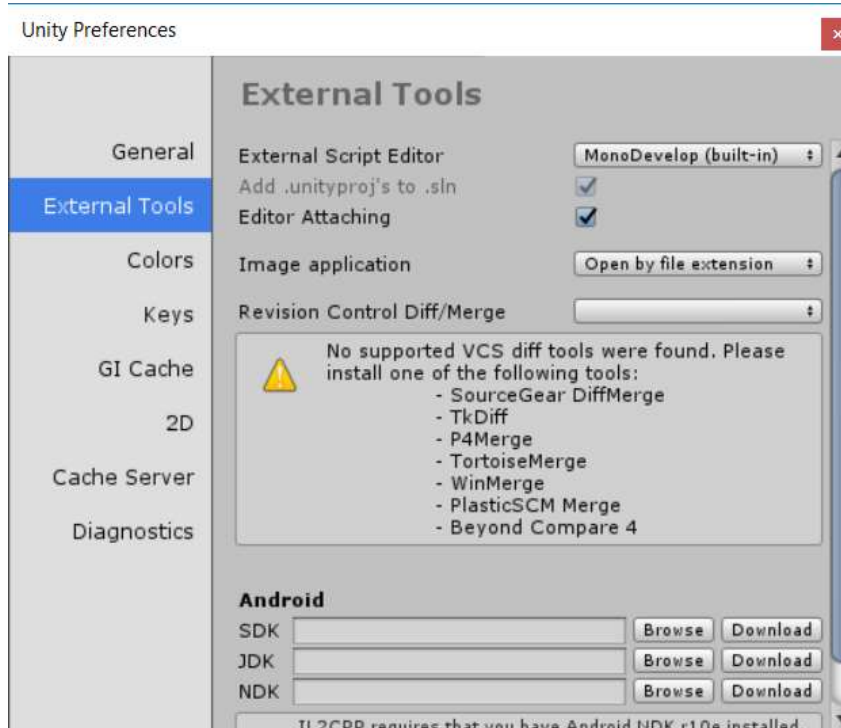


Рисунок 3.1 - Вкладка **External Tools**

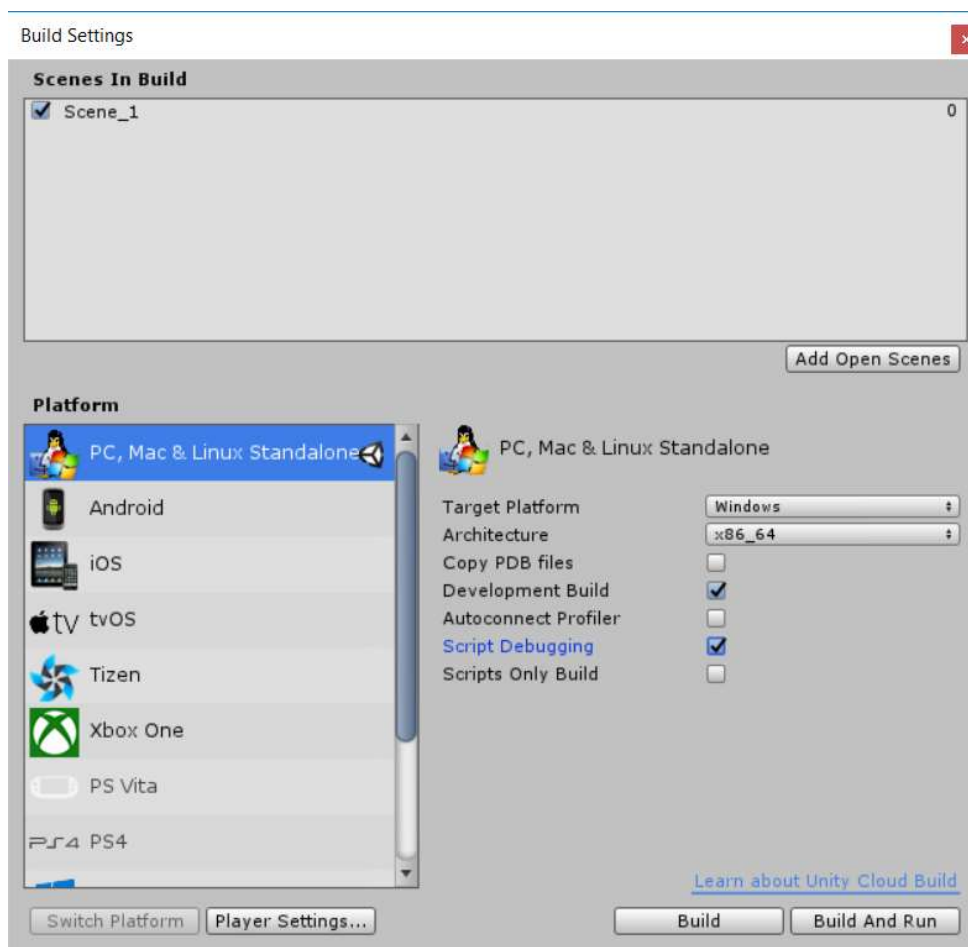


Рисунок 3.2 - Вікно **BuildSettings**

Огляд можливостей налаштування.

Для налаштування **MonoDevelop** відкриємо вікно **Options** за допомогою меню **Tools**→ **Options**.

Розділ Environment

Author Information. Заповнюється за бажанням, можна вказати своє ім'я, електронну адресу, назву організації, інформацію про товарні знаки і авторські права (рис. 3.3).

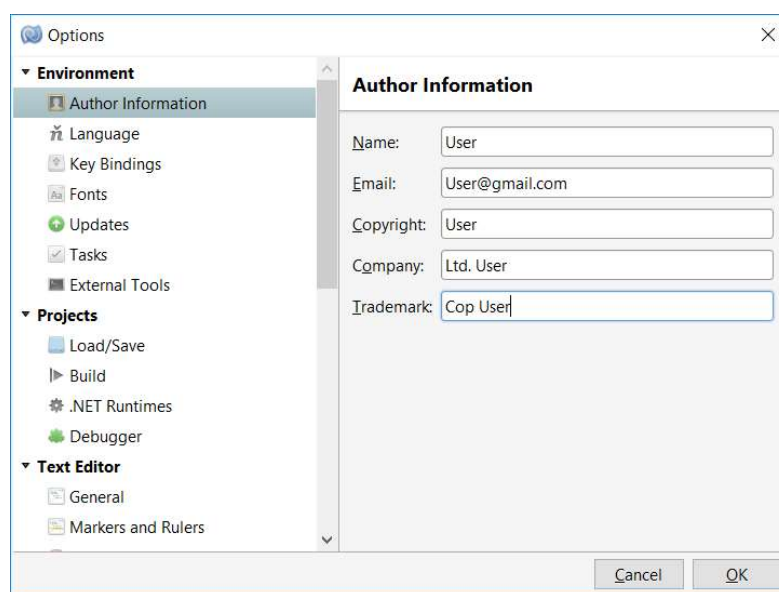


Рисунок 3.3 - Вікно **Options**. Розділ **Environment**

Language. Налаштування мови. Рекомендуємо встановити англійську мову (рис. 3.4).

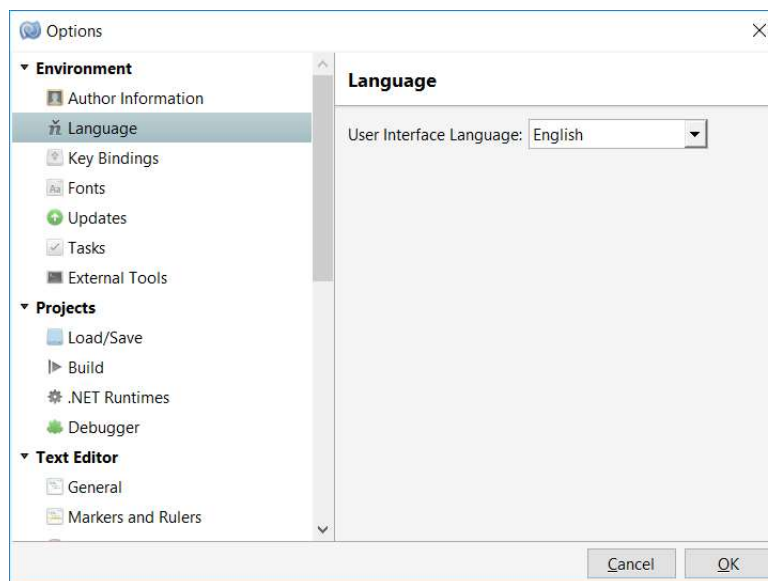


Рисунок 3.4 - Вікно Options. Розділ Language

Key bindings. Налаштування клавіатурних комбінацій, причому є і така річ як схема клавіатурних комбінацій (рис. 3.5).

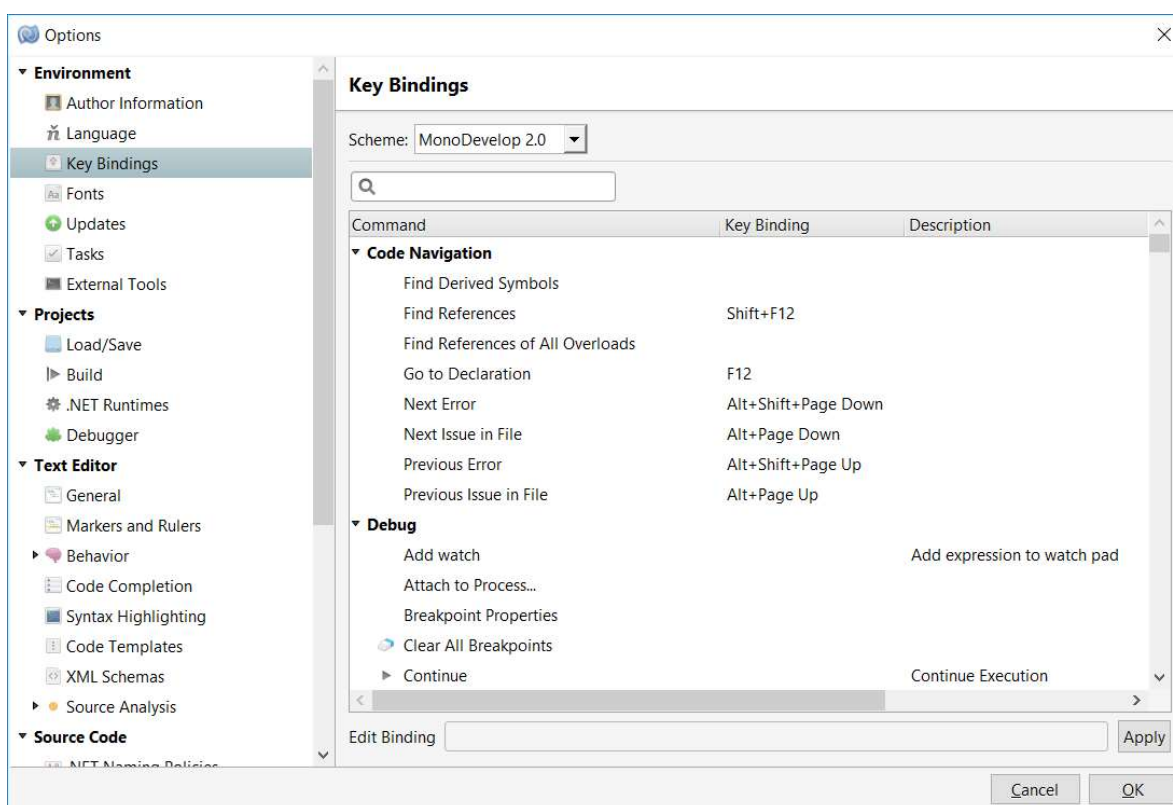


Рисунок 3.5 - Вікно Options. Розділ Key bindings

Схеми клавіатурних комбінацій дозволяють налаштувати прив'язки клавіш такі як наприклад, в **Visual Studio**. Найцікавіша річ - це пункт, який

з'являється в разі наявності однієї і тієї ж дії на різні комбінації, і він називається **View Conflicts**. Ця опція покаже комбінації з конфліктами і допоможе їх усунути, а корисною вона стає через те, що дуже часто звичайні комбінації в **Monodevelop** починають з незрозумілої причини відмовляти - і, якщо ви спостерігаєте таке в себе, то варто скористатися **View Conflicts** і усунути конфлікти прив'язок.

Fonts. Налаштування шрифтів, як для інтерфейсу, так і для редактора (рис. 3.6).

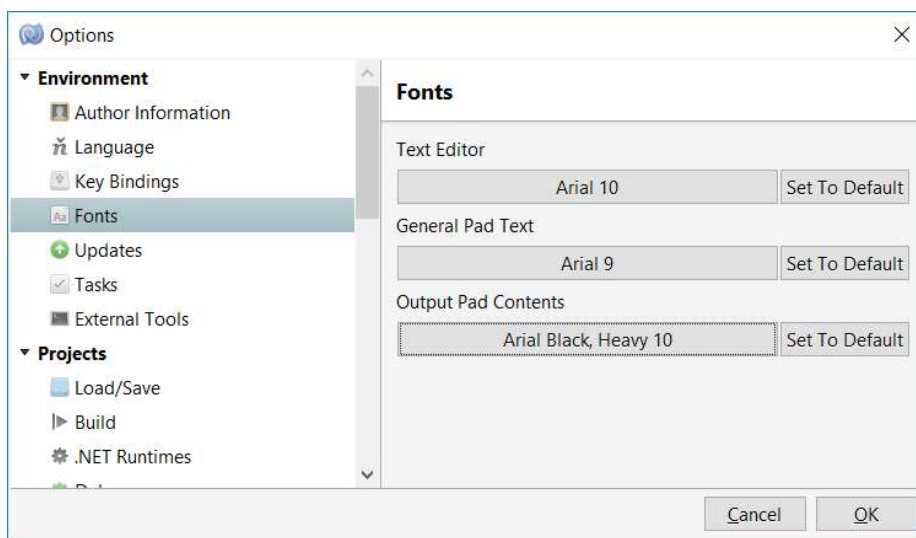


Рисунок 3.6 - Вікно **Options**. Розділ **Fonts**

Розділ **Projects**

Build. Загальні настройки процесу збірки (рис. 3.7).

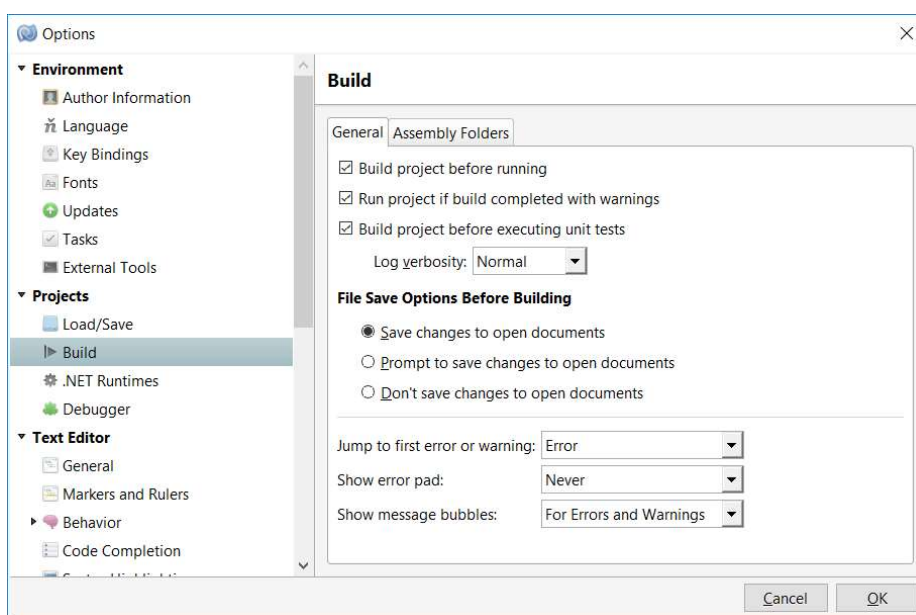


Рисунок 3.7 - Загальні настройки процесу збірки

Після налаштування параметрів в розділі **Build** отримуємо такі можливості: автоматична збірка перед виконанням/запуском юніт-тестів, автоматичне збереження змін під час запуску/збірки, деталізований висновок інформації про помилки, автоматичне підсвічування червоним кольором рядки з помилкою плюс у спливаючому вікні буде виведено інформацію про те, що сталося (іншими словами, підсвічування помилок з інформацією про них з перших).

Розділ Text Editor

General. Встановлюємо такі опції (рис. 3.8):

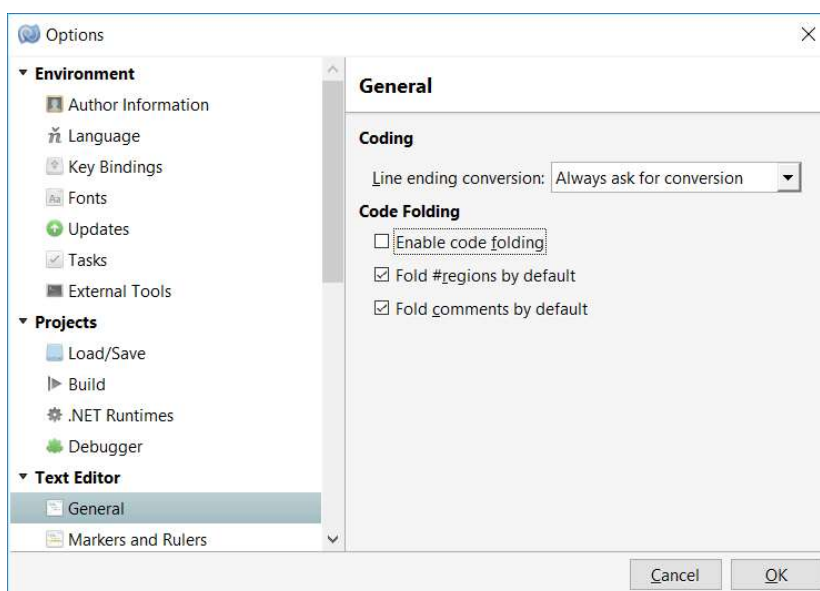


Рисунок 3.8 - Вікно **Options**. Розділ **General**

У підсумку, отримуємо згортку коментарів за замовчуванням, згортку регіонів коду і запити на конвертацію символу рядка (якщо виникне така потреба).

Markers and Rulers. Налаштування всякого роду маркерів, позначок, які допомагають орієнтуватися в коді (рис. 3.9).

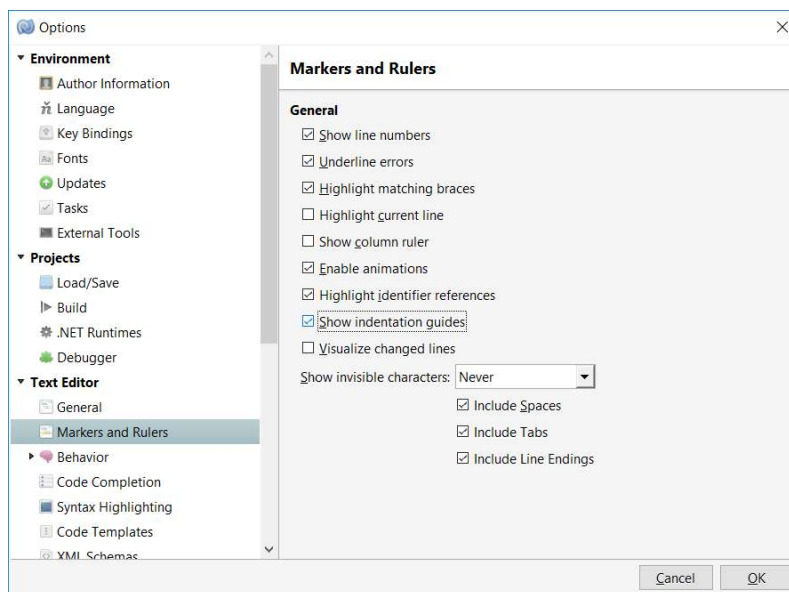


Рисунок 3.9 - Вікно **Options**. Розділ **Markers and Rulers**

Після налаштування параметрів в розділі **Markers and Rulers** отримуємо відображення номерів рядків, підкреслення помилок (відзначаються прямо на ходу: щось помилково набрали на думку аналізатора - і це відразу ж буде підкреслено), підсвічування збігу дужок, посилань на ідентифікатори, а крім того маємо відображення ліній інденту (тобто графічне відображення рівня відступу). Нижні опції відключають візуалізацію змінених рядків і відображення невидимих символів (пропуски, переклади рядків, кінці файлів і т.п).

Behavior. Налаштування для форматування коду (рис. 3.10).

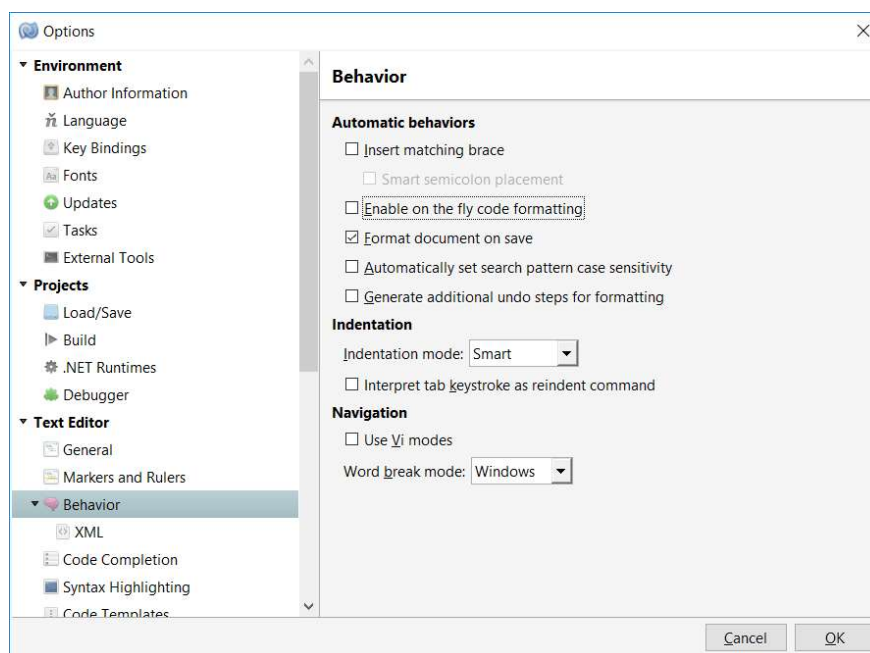


Рисунок 3.10 - Вікно **Options**. Розділ **Behavior**

Code Completion. Налаштування автодоповнень (рис. 3.11).

Після налаштування параметрів в розділі **Code Completion** в автодоповненні будуть показуватися і обсяги імпорту, члени класів будуть показуватися в розширеному вигляді і доповнення по ним також буде розширено.

Крім того, в списку під автодоповненням (список виникає після натискання клавіші **Tab**) буде як мінімум 10 рядків (тобто збільшено обсяг списку).

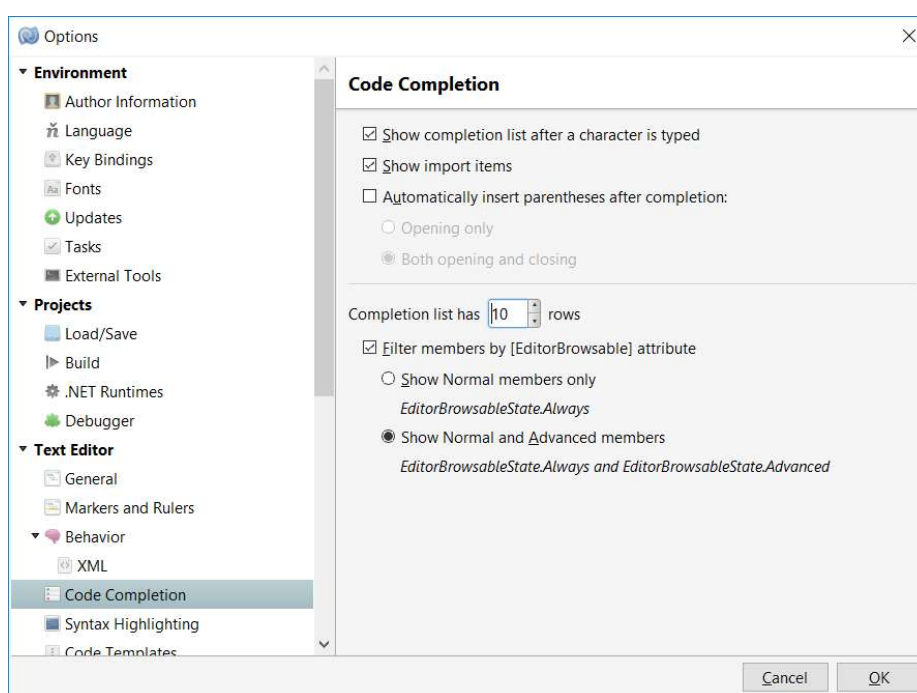


Рисунок 3.11 - Вікно **Options**. Розділ **Code Completion**

Syntax Highlighting. Зміна колірної схеми підсвічування коду (рис. 3.12).

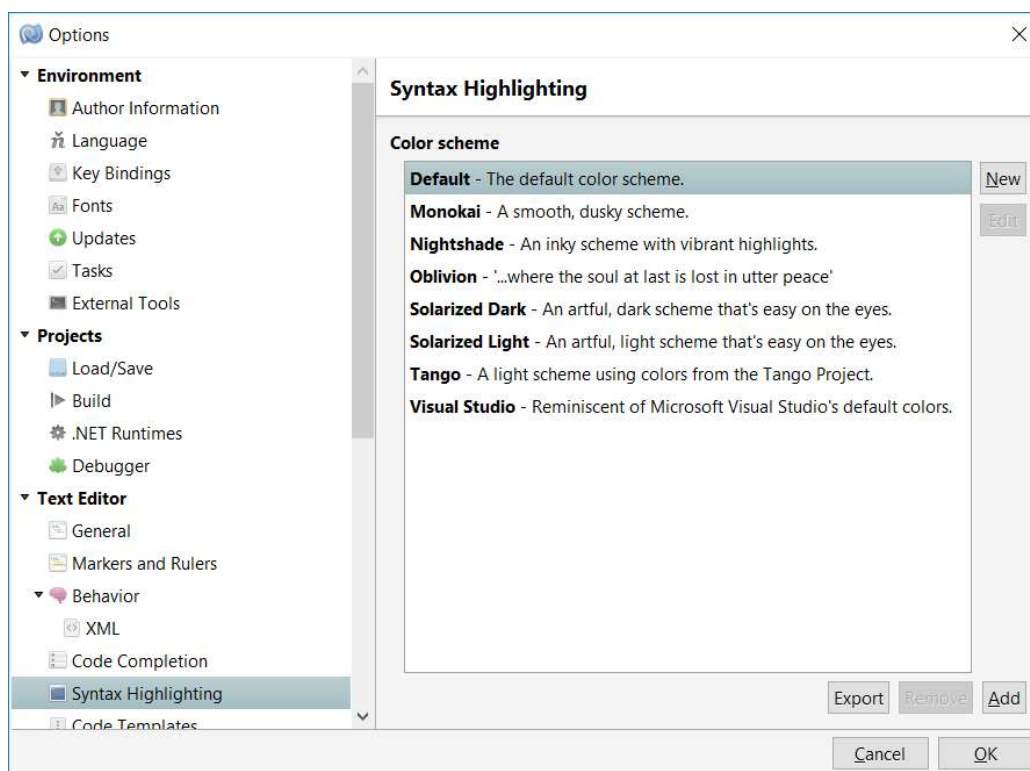


Рисунок 3.12 - Вікно Options. Розділ Syntax Highlighting

Source Analysis. Включає аналіз вихідного коду (рис. 3.13).

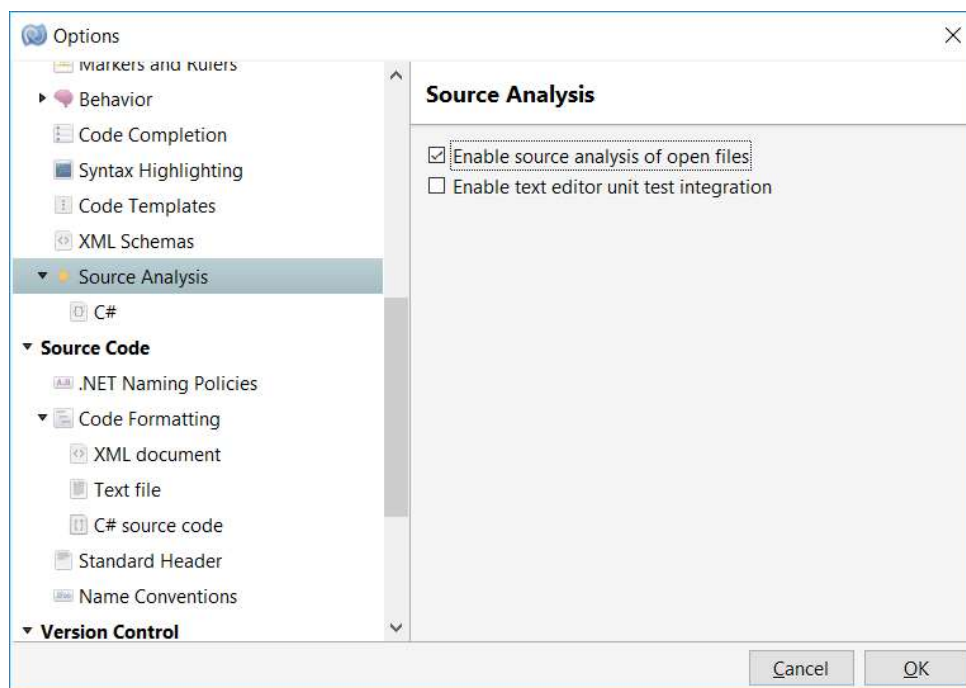


Рисунок 3.13 - Вікно Options. Розділ Source Analysis

Створення проекту в MonoDevelop

Для створення нового проекту в меню «**File**» вибираємо пункт «**New→Workspace**». Відкриється майстер проекту (рис. 3.14).

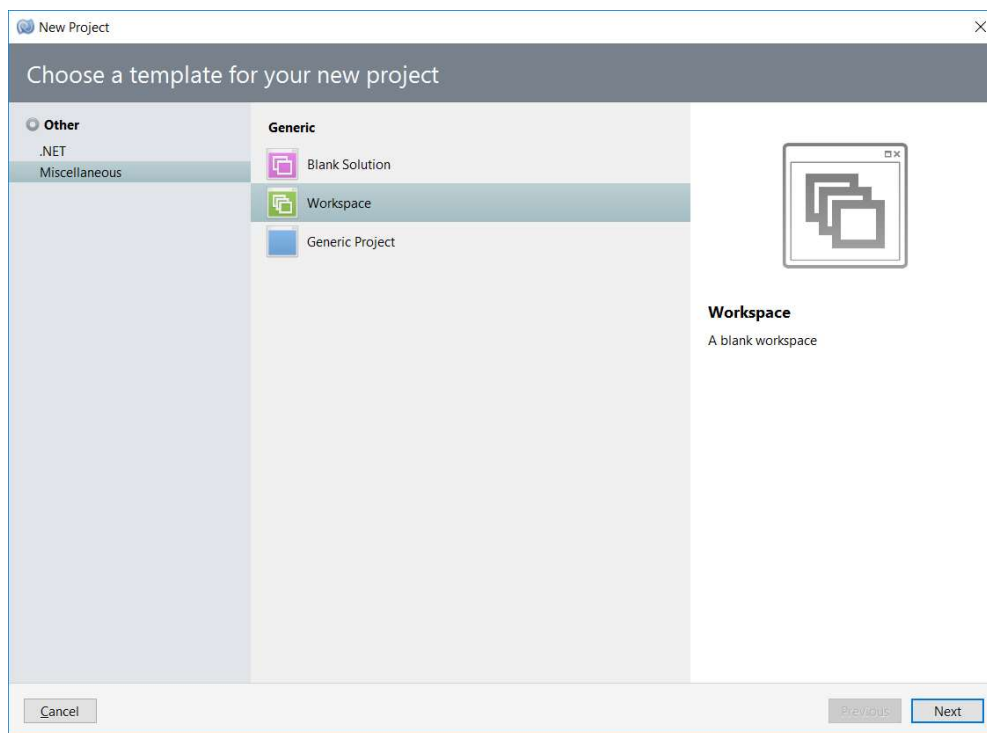


Рисунок 3.14 - Майстер проекту

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. 3.15).

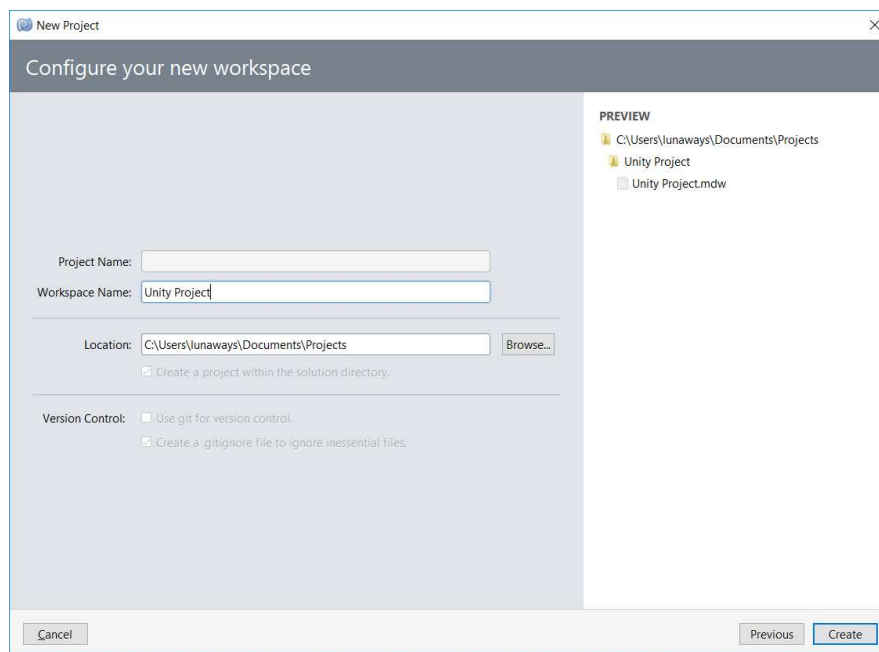


Рисунок 3.15 - Створення проекту

Натискаємо кнопку **Create**, після чого отримуємо порожню папку проекту (рис. 3.16).

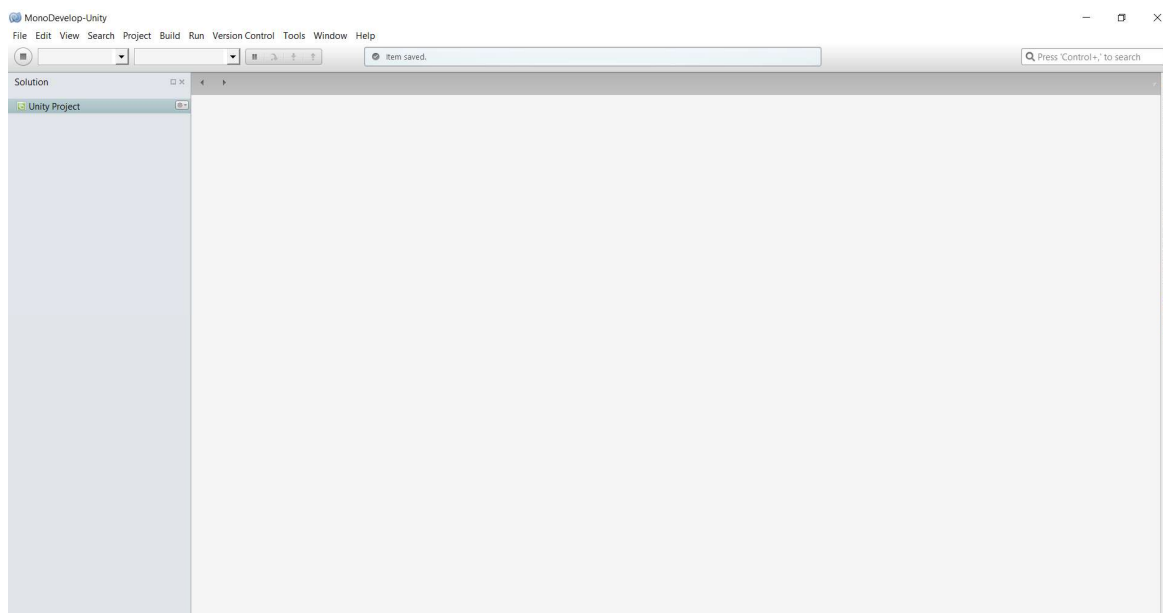


Рисунок 3.16 - Папка проекту

Створимо новий C# - файл за допомогою меню **File**→**New**→**File** (рис. 3.17).

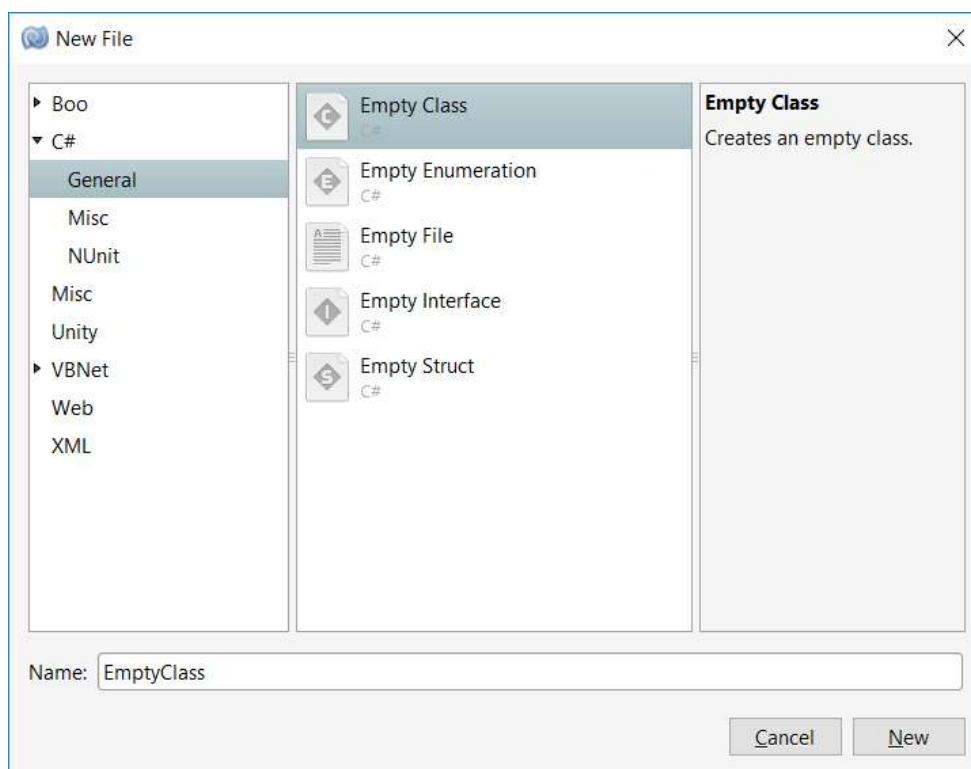


Рисунок 3.17 - Створення нового C# - файлу

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

Налагодження вихідного коду

Відкриті файли відображаються в **MonoDevelop** як вкладки і можуть бути відредаговані в стандартному текстовому редакторі. Зліва від панелі редактора знаходиться сіра шкала точок останова (**breakpoint bar**). Клік ПКМ на цій шкалі додає маркер точок переривання (**breakpoint**) напроти поточного рядка коду.

Додавання точки переривання на шкалу повідомляє **Unity**, що після досягнення цієї лінії в режимі відтворення слід призупинити виконання сценарію. Коли скрипт так «заморожується», ви можете використовувати **Debugger** для визначення того, що саме робить скрипт (рис. 3.18).

При припиненні виконання на точці переривання, у вкладках в нижній частині вікна **MonoDevelop** відображається інформація про поточний стан виконання. Найбільш важливою з вкладок є **Locals**.

У цій вкладці показані значення локальних змінних в поточній функції (псевдо-локальна змінна з ім'ям **this** автоматично доступна в кожній функції, хоч вона і не оголошена явно; це посилання на поточний екземпляр скрипта, так за допомогою «**this**» можна отримати доступ до всіх змінних, що оголошені в скрипті).

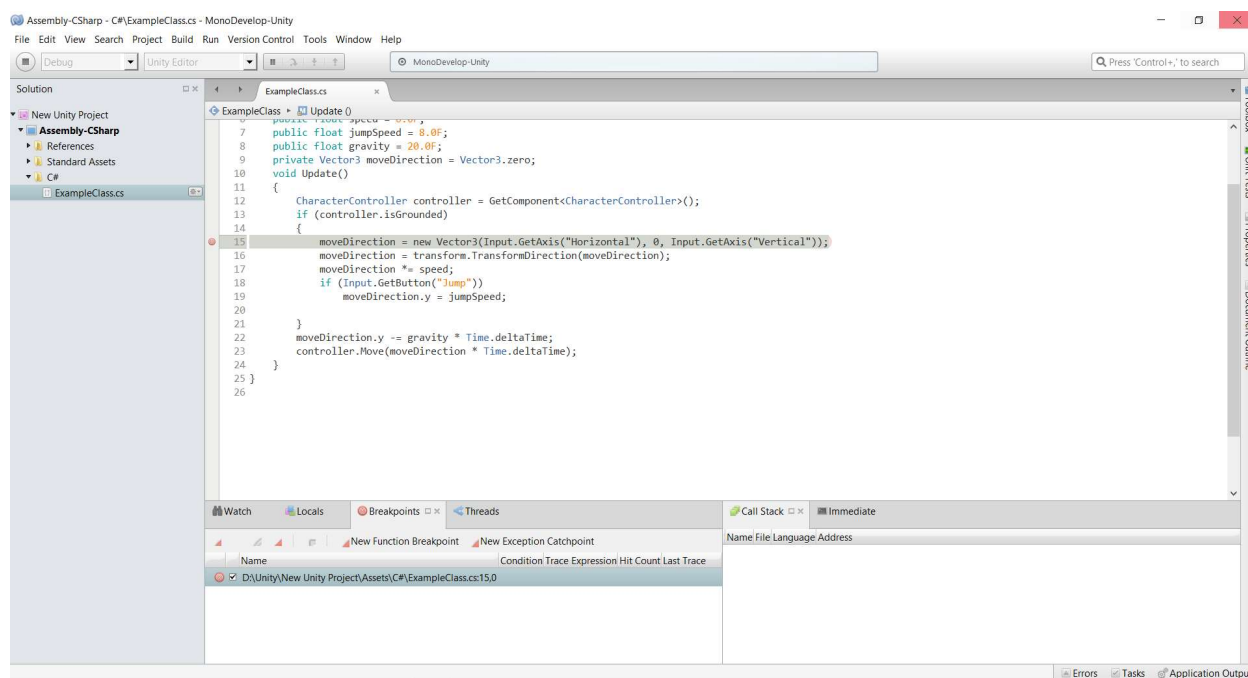


Рисунок 3.18 - Додавання точки переривання

Ви можете використовувати точки переривання разом з вкладкою **Locals** для отримання ефекту, аналогічного додаванню виразів **print** в ваш код - ви можете переглядати значення змінних в будь-який зручний вам момент часу.

У вкладці **Locals** ви також можете редагувати значення змінних. Це може виявитися корисним в разі, якщо ви знайшли невірну встановлену змінну і хочете перевірити, чи пропаде ця проблема при установці правильного значення.

Інша корисна функція MonoDevelop - послідовне виконання. Коли виконання призупиниться на точці з переривання, стане доступна шкала інструментів налаштування у верхній частині вікна **MonoDevelop**.



Рисунок 3.19 - Інструменти покрокового налагодження MonoDevelop

Перші чотири кнопки це **Continue** (продовжити), **Step Over** (крок через), **Step In** (крок всередину) і **Step Out** (крок назовні), які також можуть спрацьовувати по командам з меню **Run** (Праву кнопку, **Detach** може бути використано для закінчення сесії). **Continue** продовжує виконання до чергової точки переривання. **Step Over** і **Step In** виконують один рядок коду в один момент часу. Відмінності між ними в тому, що **Step Over** повністю виконує будь-які функції за одну лінію, а **Step In** дозволяє «зробити крок» всередину функції.

Застосування MonoDevelop при створенні ігрових додатків

Продемонструємо застосування MonoDevelop при виявленні зіткнень (collisions) між об'єктами.

Для цього створимо нову сцену. Додамо об'єкт «Plane», об'єкти «Cube» «Sphere» (рис. 3.20).

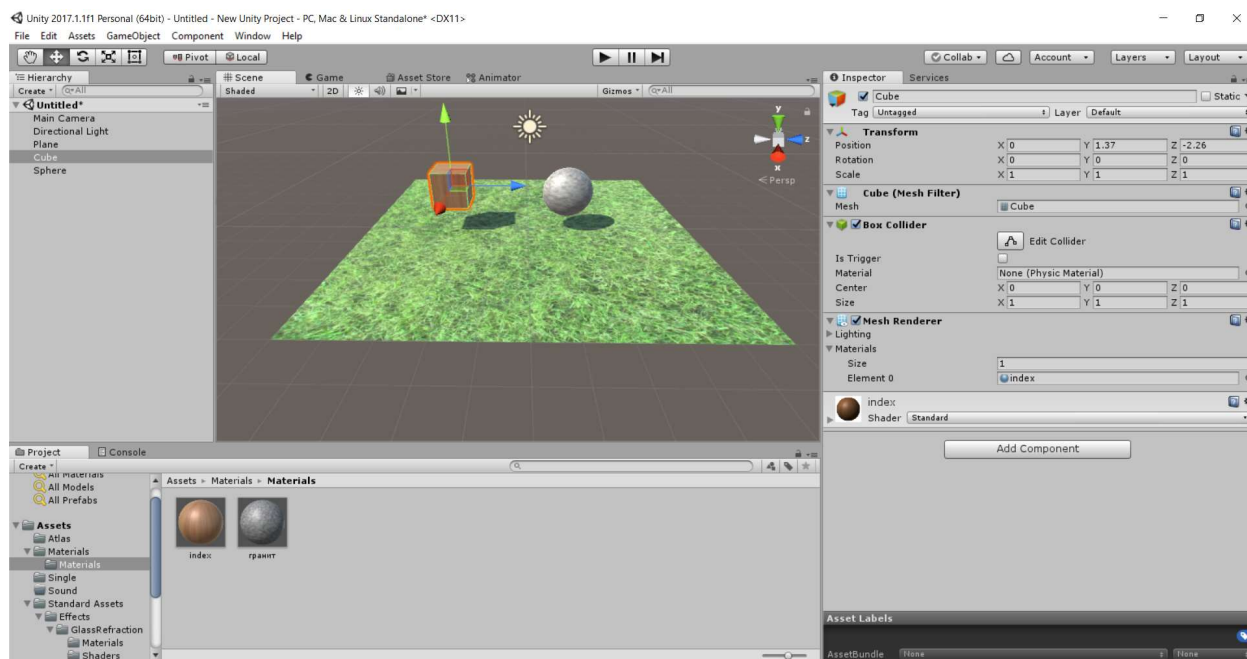


Рисунок 3.20 – Нова сцена з об'єктами «Plane», «Cube» та «Sphere»

У **Project View** за допомогою контекстного меню «Create→ C# Script» створіть новий файл C# - скрипта. Назвіть його «Collisions». Двічі натисніть ПКМ на файлі скрипта і ви відкриєте його в редакторі скриптів **MonoDevelop** (рис. 3.21).

Зверніть увагу, що назва класу завжди має збігатися з назвою скрипта (реєстр букв теж має значення):

```
public class Collisions : MonoBehaviour {  
    // Use this for initialization  
    void Start() {  
    }  
    // Update is called once per frame  
    void Update() {  
    }  
}
```

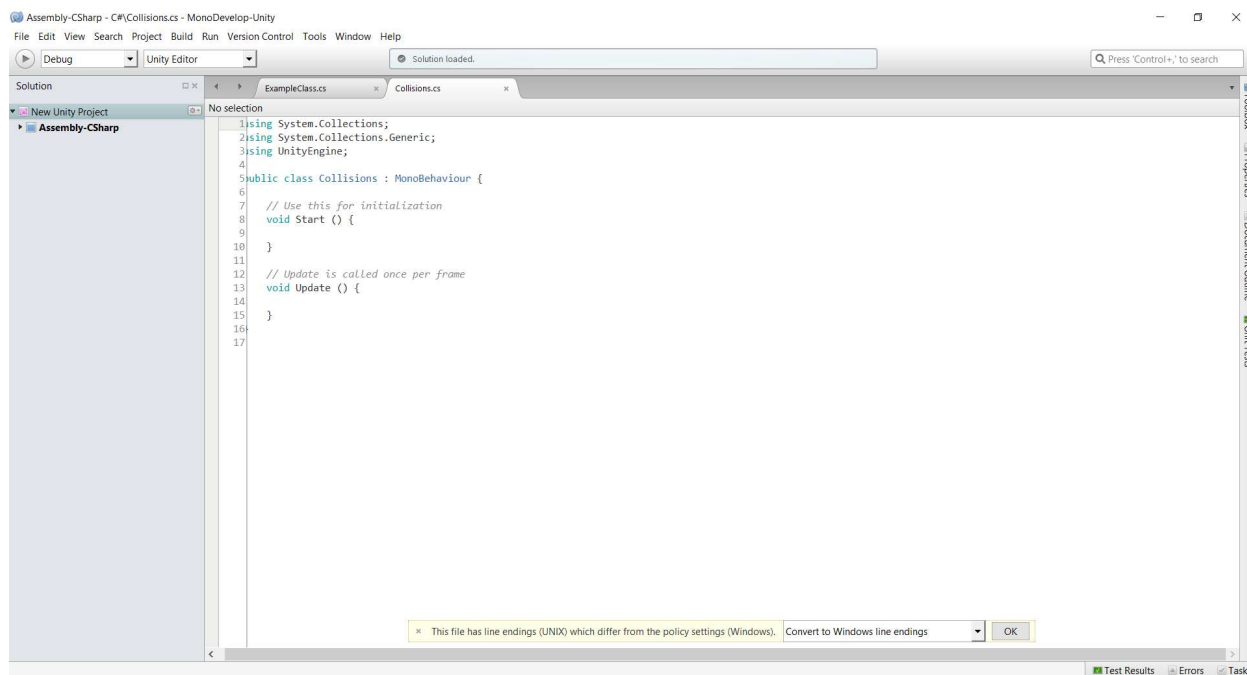


Рисунок 3.21 – C# - скрипт «Collisions» в редакторі скриптів **MonoDevelop**

За замовчуванням Unity створює каркас для C# -скрипта. Видаліть з тіла класу «Collisions» весь вміст і замініть наступним:

```
// Метод, який викликається при зіткненні об'єкта  
void OnCollisionEnter() {  
    Debug.Log("Hit Something");// Передаем сообщение в консоль Unity  
}
```

Функція **OnCollisionEnter** визначає зіткнення об'єкта з іншими об'єктами. А статичний метод **Log** класу **Debug** пише повідомлення в консоль **Unity**. Після збереження скрипта, додайте його як компонент для падаючого куба.

Тепер якщо ви натиснете **Play**, в момент коли куб торкнеться підлоги консоль видасть відповідне повідомлення. Відкрити вікно консолі можна комбінацією **ctrl + shift + c** в Windows. Останнє консольне повідомлення відображається в статус барі проекту (внизу вікна).

Причому повідомлення буде видаватися при кожному зіткненні.

Для того, щоб зрозуміти з яким саме об'єктом зіткнувся куб, ми будемо брати значення параметра класу **Collision**, яке буде приймати метод **OnCollisionEnter**:

```
// Тепер метод приймає об'єкт класу Collision,  
// з яким відбувається зіткнення  
void OnCollisionEnter(Collision myCollision) {  
    // визначення зіткнення з двома об'єктами  
    if (myCollision.gameObject.name == "Floor") {  
        // Звертаємося до імені об'єкта з яким зіткнулися  
        Debug.Log("Hit the floor");  
    }  
    else if (myCollision.gameObject.name == "Wall") {  
        Debug.Log("Hit the wall");  
    }  
}  
}
```

Натисніть кнопку **Play** і після того як ваш куб торкнеться кілька разів сфери зупиніть гру і відкрийте консоль. Тепер при зіткненні буде видаватися відповідні повідомлення.

Завдання до лабораторної роботи №3

1. Завантажити проект Unity.
2. Створити нову сцену. Додати об'єкт «**Plane**», об'єкти «**Cube**» «**Sphere**».
3. У **Project View** створіть новий файл C# - скрипта. Назвіть його «Collisions».
4. Відкрийте C# - скрипт в редакторі скриптів **MonoDevelop**.
5. Відредагуйте C# - скрипт, додавши новий метод **void OnCollisionEnter**.
6. Натисніть кнопку **Play** та отримайте відповідне повідомлення в консолі.

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?

ЛАБОРАТОРНА РОБОТА № 4

ТЕМА: «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 імпорту аудіокліпів в ігрові додатки.

Очікувані результати

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

Імпорт звукових ефектів

Для відтворення звуків потрібно імпортувати аудіофайли в Unity - проект.

Формати аудіофайлів, які підтримує Unity подано у табл. 4.1.

Таблиця 4.1 - Формати аудіофайлів, які підтримує Unity

Тип файла	Переваги та недоліки
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

	<p>модуляцію. Також існує і стисла версія формату AIFF, яку називають AIFC (іноді AIFF-C), в якій для стиснення можуть бути використані різні кодеки.</p> <p>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.</p>
MP3	<p>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.</p> <p>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).</p> <p>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.</p> <p>Режими кодування і опції:</p> <p>Існує три версії MP3 формату для різних потреб: MPEG-1, MPEG-2 і MPEG-2.5. Відрізняються вони можливими діапазонами бітрейту і частоти дискретизації:</p>

	<p>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.</p>
OGG	<p>Ogg - відкритий стандарт формату мультимедіаконтейнера, який є основним файловим і потоковим форматом для мультимедіакодеків 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.</p> <p>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.</p> <p>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:</p> <p>з втратами:</p> <p>Opus (раніше Harmony) - з низькою затримкою кодування (від 2,5 мс до 60 мс, налаштовується) і більш високою компресією аудіо, так само бітрейт від 6 до 510 кбіт/с; Sreex - 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).</p> <p>без втрат:</p> <p>FLAC - для обробки звукових архівів та інших аудіо, що вимагають високої якості відтворення.</p>
MOD	<p>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.</p> <p>Кожен файл формату MOD містить в собі оцифровані записи реального звучання інструментів, так звані семпли.</p>

	<p>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.</p>
XM	<p>Формат XM – це розширений модуль MOD, тип аудіофайлу FastTracker 2, який було введено розробником - демогрупою Triton.</p> <p>XM-файл є мультисемплінгом 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).</p> <p>XM є основним форматом для більшості трекерної музики.</p>

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.

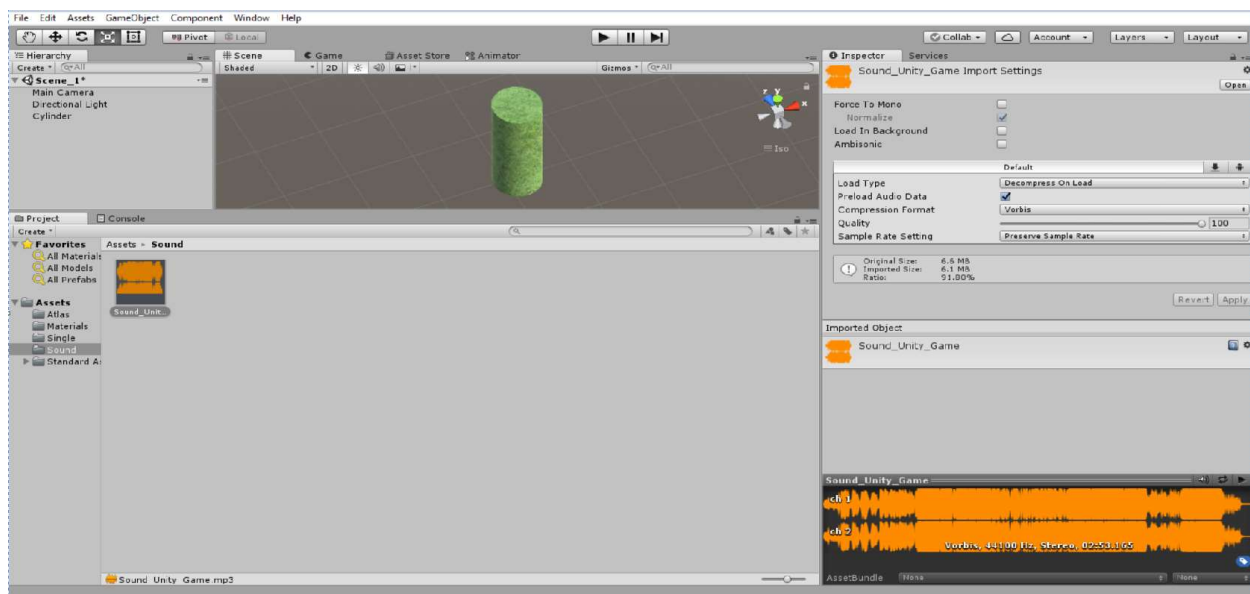


Рисунок 4.1 – Імпорт аудіофайлу

На панелі «Inspector» можна задати параметри імпорту аудіофайлів (рис. 4.2). Прапорець «Force To Mono» встановлювати не потрібно. Він дозволяє вибрати між моно- і стереозвуком. Найчастіше звук записується в стерео, тобто в файлі присутні одночасно дві хвилі, одна для лівого, інша для правого навушника або колонки. Щоб зменшити розмір файлу, можна прибрати половину звукової інформації.

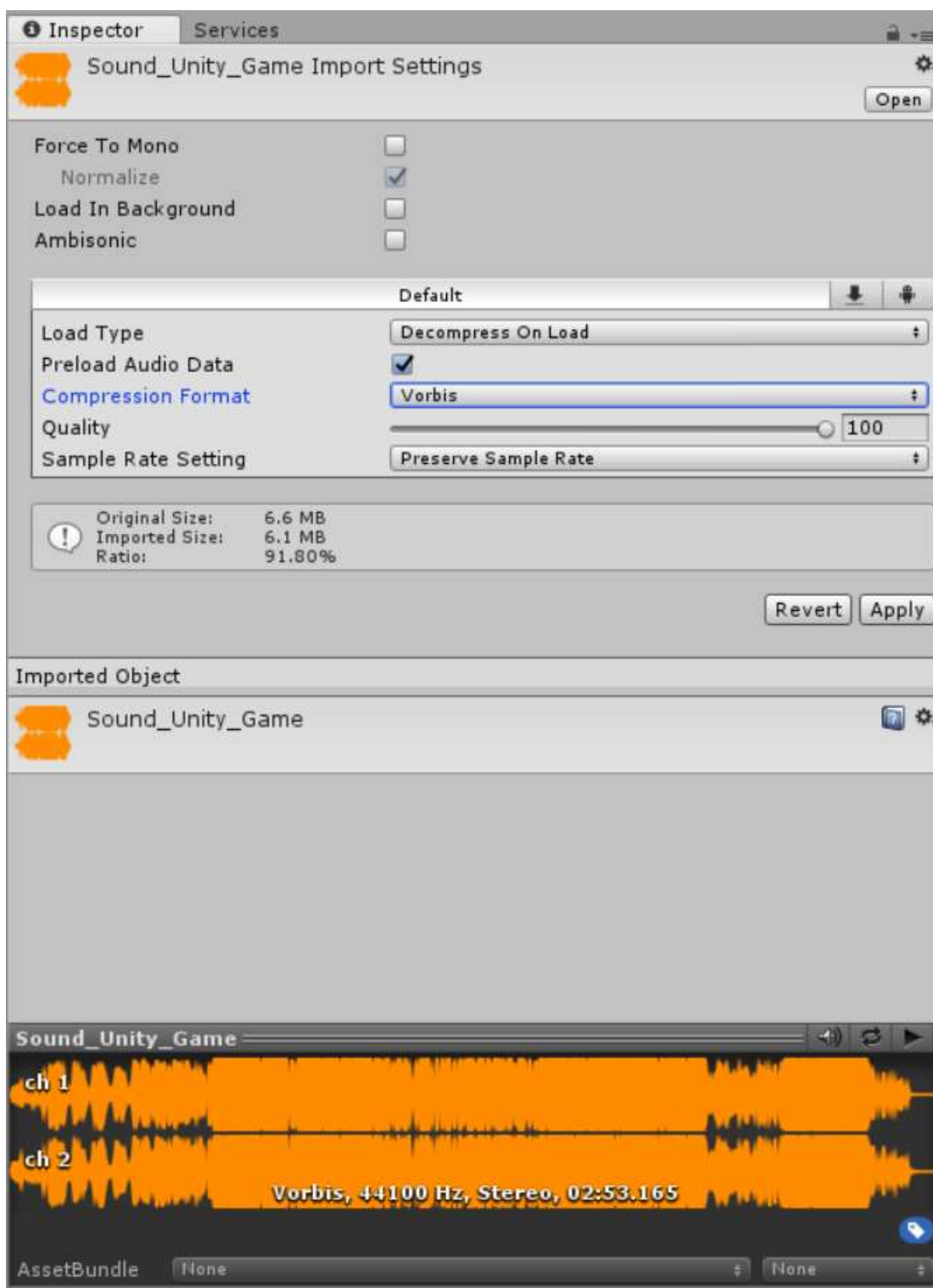


Рисунок 4.2 – Параметри імпорту аудіофайлу

Після цього на обидві колонки буде надсилатися одна хвиля замість двох.

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

Завантаження звуку у фоновому режимі дозволяє програмі продовжити свою роботу. Цей варіант в загальному випадку підходить для довгих музичних кліпів, але виключає можливість негайного відтворення звуку. Для коротких звукових кліпів прапорець «**LoadInBackground**» зазвичай не встановлюється, що гарантує їх повне завантаження перед відтворенням.

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

Технологія **Ambisonics** призначена перш за все для запису зі збереженням реальної звукової картини. При використанні багатоканального запису можлива імітація кругової (три канали) і сферичної (чотири канали) панорами. Існує матрична система UHJ, що дозволяє кодування зробленої за технологією **Ambisonics** багатоканального запису в два канали. Результат кодування сумісний зі стерео і моно. Поряд з двоканальним, можливо і багатоканальне відтворення записів, зроблених за технологією **Ambisonics**.

При використанні чотирьох каналів відтворення (тобто чотирьох незалежних акустичних систем) можна отримати кругову панораму. Для сферичної панорами необхідно шість каналів, можливе використання і більшої кількості акустичних систем.

Найбільш важливими параметрами є «**Load Type**» (тип, що завантажується) та «**Compression Format**» (формат стиснення). Останній відповідає за формат збережених звукової інформації.

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.

5. 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 мультिकанальні 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, що правда для них недоступний перегляд форми сигналу в інспекторі імпорту.

Компонент **Audio Source** відтворює **Audio Clip** в сцені. Якщо **Audio Clip** є 3D кліпом, джерело програватиметься 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.

Властивості компоненту **Audio Source** подано у табл. 4.2.

Таблиця 4.2 - Властивості компоненту **Audio Source**

Властивість	Функція
Audio Clip	Посилання на аудіо кліп для програвання цим джерелом.
Output	Звук можна виводити через Audio Listener або звуковий мікшер.
Mute	Якщо обрана ця властивість, звук все ще буде програватися, але не буде звучати.
Bypass Effects	Застосовується для швидкого пропуску всіх звукових ефектів, застосованих до джерела звуку. Простий шлях включення/відключення всіх звукових ефектів.
Bypass Listener Effects	Застосовується для швидкого включення/відключення всіх звукових ефектів, застосованих до Audio Listener.
Bypass Reverb Zones	Застосовується для швидкого включення/відключення всіх зон реверберації.
Play On Awake	Якщо обрана ця властивість, звук почне відтворюватися одразу після завантаження сцени. Якщо властивість відключено, вам потрібно буде запустити звук програмно, за допомогою методу Play ().
Loop	Застосовується для нескінченного повтору Audio Clip після його закінчення.
Priority	Визначає пріоритет даного джерела звуку серед всіх інших джерел у сцені (0 = найбільш важливий, 256 = найменш важливий, 128 за замовчуванням). Використовуйте 0 для музики, щоб уникнути її випадкового перемикання.
Volume	Визначає наскільки гучний звук на відстані однієї світової одиниці виміру (одного метра) від Audio Listener.
Pitch	Ступінь зміни висоти тону при уповільненні/прискоренні Audio Clip. Величина 1 означає нормальну швидкість відтворення.
Stereo Pan	Встановлює положення в стерео області 2D звуків.
Spatial Blend	Встановлює ступінь впливу 3D движка на джерело звуку.

Doppler Level	Визначає кількість ефекту Доплера, що застосовується до даного джерела (при значенні 0 ефект застосовуватися не буде).
Spread	Встановлює кут поширення для 3d стерео або мультіканального звуку в просторі динаміків.
Min Distance	Гучність звуку буде максимальною, наскільки це можливо, протягом MinDistance. Поза MinDistance вона буде поступово знижуватися. Збільште MinDistance звуку, щоб зробити його «голосніше» в тривимірному світі, знизьте, щоб зробити звук «тихіше» в тривимірному світі.
Max Distance	Відстань, на якому звук перестає затухати. За межами цієї точки його гучність залишиться на рівні, на якому вона була б на відстані MaxDistance одиниць від слухача і більше не буде затухати.
Rolloff Mode	Параметр визначає швидкість загасання звуку. Чим вище значення, там ближче повинен бути слухач до звуку перш, ніж його можна буде почути (визначається за графіком). - Logarithmic Rolloff Гучність звуку є досить високою, коли ви близько до джерела, але у разі віддалення від об'єкта вона досить швидко падає. - Linear Rolloff Чим далі ви від джерела звуку, тим гірше ви його чуєте. - Custom Rolloff Звук джерела аудіо поводиться відповідно до графіка загасання.

Компонент **Audio Listener** поводить як мікрофон. Він отримує вхідні дані з будь-якого джерела звуку (Audio Source) в сцені і програв звуку 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.

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** (рис. 4.3-4.4).

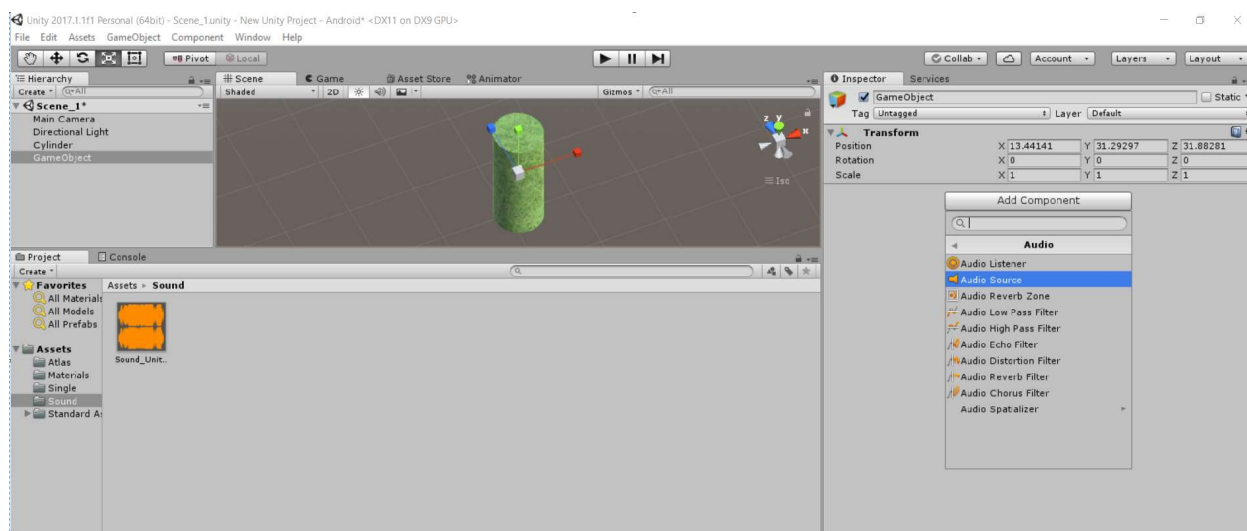


Рисунок 4.3 – Створення компоненту **Audio Source**

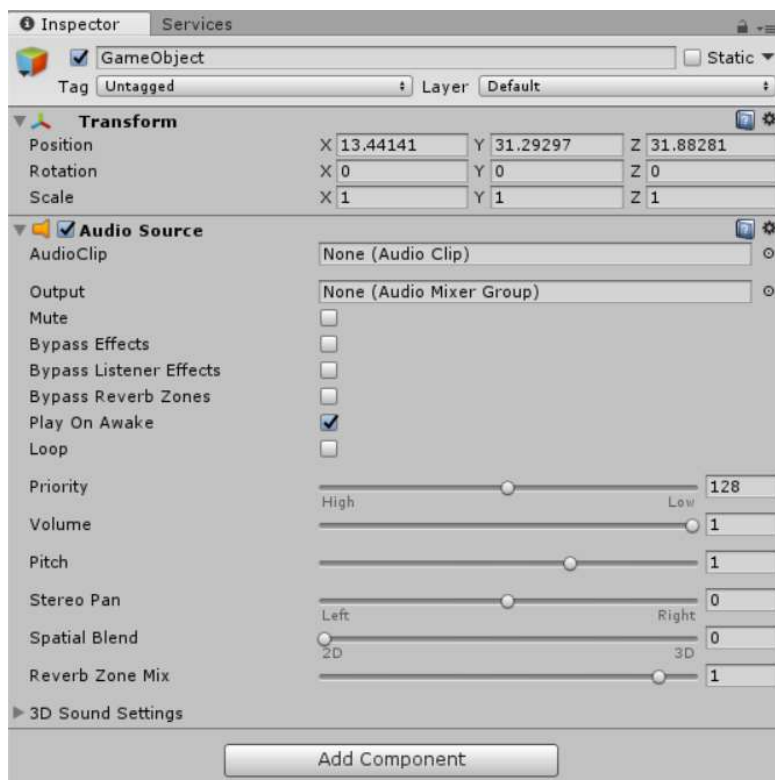


Рисунок 4.4 – Властивості компонента **Audio Source**

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».

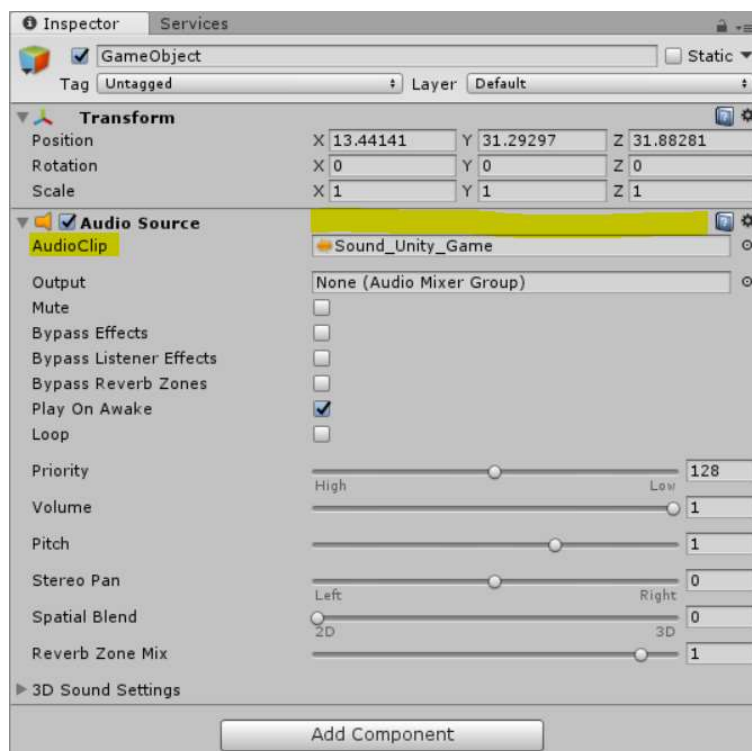


Рисунок 4.5 – Властивість **AudioClip** компонента **Audio Source**

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

Слід зауважити, що якщо ви хочете створити **Audio Source** тільки для одного компонента **Audio Clip**, який знаходиться в папці **Assets** вашого проекту, то ви можете просто перетягнути кліп у вікно сцени - **GameObject** з компонентом **Audio Source** для вашого кліпу буде створений автоматично. Перетягування кліпу на існуючий **GameObject** додасть до нього кліп разом з новим **Audio Source**, якщо цього компонента там ще немає. Якщо на об'єкті вже є компонент **Audio Source**, тоді новий кліп, який ви перетягнули, замінить той, який вже використовується джерелом звуку.

6. Робота з диспетчером **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. 4.6).

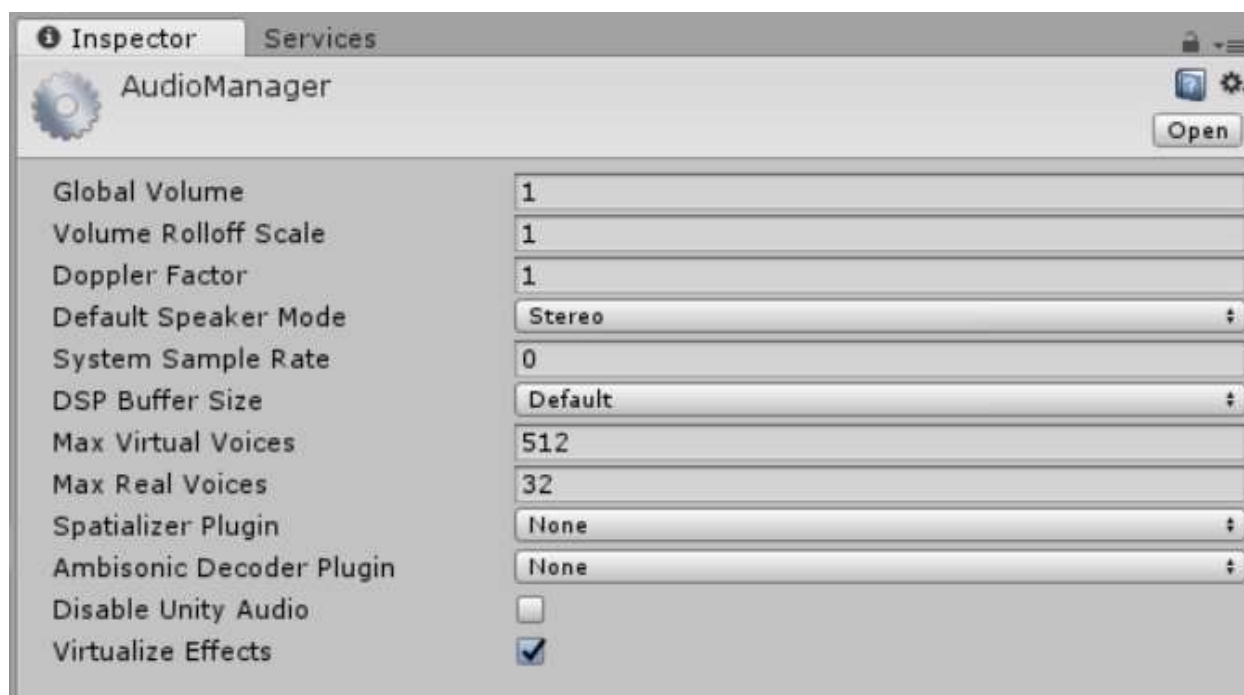


Рисунок 4.6 – Диспетчер **AudioManager**

Властивості диспетчеру **AudioManager** подано у табл. 4.3.

Таблиця 4.3 - Властивості диспетчеру **AudioManager**

Властивість	Функція
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 чути 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.
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

Завдання до лабораторної роботи №4

1. Завантажити проект Unity.
2. Обрати необхідні аудіофайли.
3. Здійснити імпорт колекції аудіофайлів в проект Unity та задати параметри імпорту аудіофайлів.
4. Додати до проекту необхідні компоненти для роботи з аудіокліпами AudioClip, AudioSource та AudioListener.
5. Здійснити налаштування компоненту AudioSource.
6. Здійснити відтворення аудіокліпу.
7. Здійснити налаштування максимальної гучності всіх звуків, які буде відтворено в сцені за допомогою диспетчеру **AudioManager**.

Контрольні запитання

- Назвіть основні функції центрального диспетчера управління звуком AudioManager.
- 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?
- Назвіть необхідні компоненти для роботи з аудіокліпами.

ЛАБОРАТОРНА РОБОТА № 5

ТЕМА: «АНІМАЦІЯ СПРАЙТАМИ. РОБОТА З СИСТЕМОЮ АНІМАЦІЇ MECANIM»

Анотація

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

Мета лабораторної роботи

Сформувати у студентів знання щодо методів розробки анімованих персонажів, технології анімації за допомогою інструменту 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.

Анімація спрайтами

Імпорт та налаштування спрайтів. Спрайт (**sprite**) - це спеціальний 2D-об'єкт Unity. Він дозволяє відображати в сцені імпортовану плоску, регулярну текстуру. В іграх з горизонтальним слайдингом спрайти використовуються і для анімованих персонажів, і для реквізитів.

Застосування 2D-спрайтів можна проілюструвати на прикладі книги з рухомими картинками (flip-book), де на кожній сторінці зображений ескіз одного ключового кадру і читач, швидко гортаючи сторінки, спостерігає ефект анімації.

Є два способи імпортування та налаштування анімованих спрайтів:

Окремі спрайти

Якщо ви створили анімаційний спрайт, використовуючи окремі файли зображень для кожного кадру анімації, то для їх імпорту ви можете просто перетягнути їх всі відразу в панель проекту Unity.



Anim_01.png



Anim_02.png



Anim_03.png



Anim_04.png

Рисунок 5.1 - Анімаційні слайди з окремих файлів

При цьому кожен файл буде імпортовано як звичайну текстуру (рис. 5.2).

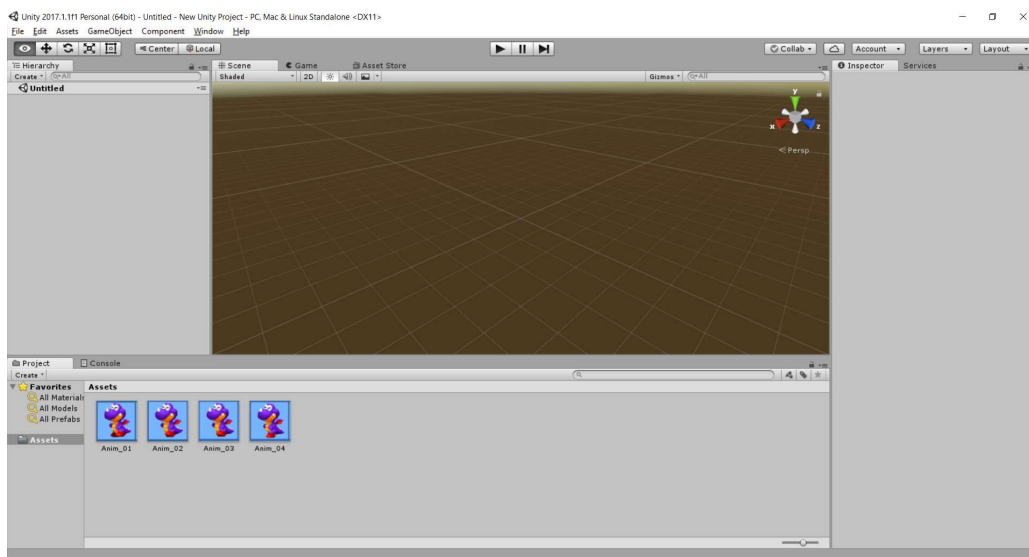


Рисунок 5.2 - Імпорт анімаційних слайдів з окремих файлів в панель проекту Unity

Для перетворення їх в спрайти виберіть всі імпортовані текстури в панелі проекту. У інспекторі об'єктів виберіть опцію **Sprite** для параметра **Texture Type**. Переконайтеся, що режим спрайтів **Sprite Mode** встановлений в **Single**, так як кожне окреме зображення є одним кадром для загального спрайту (одного і того ж персонажа).

Опція **Pivot** для спрайту повинна бути встановлена в **Bottom**. Такий вибір дозволить правильно розташувати спрайт в межах сцени. Після цього натисніть на кнопку **Apply** для підтвердження (рис. 5.3).

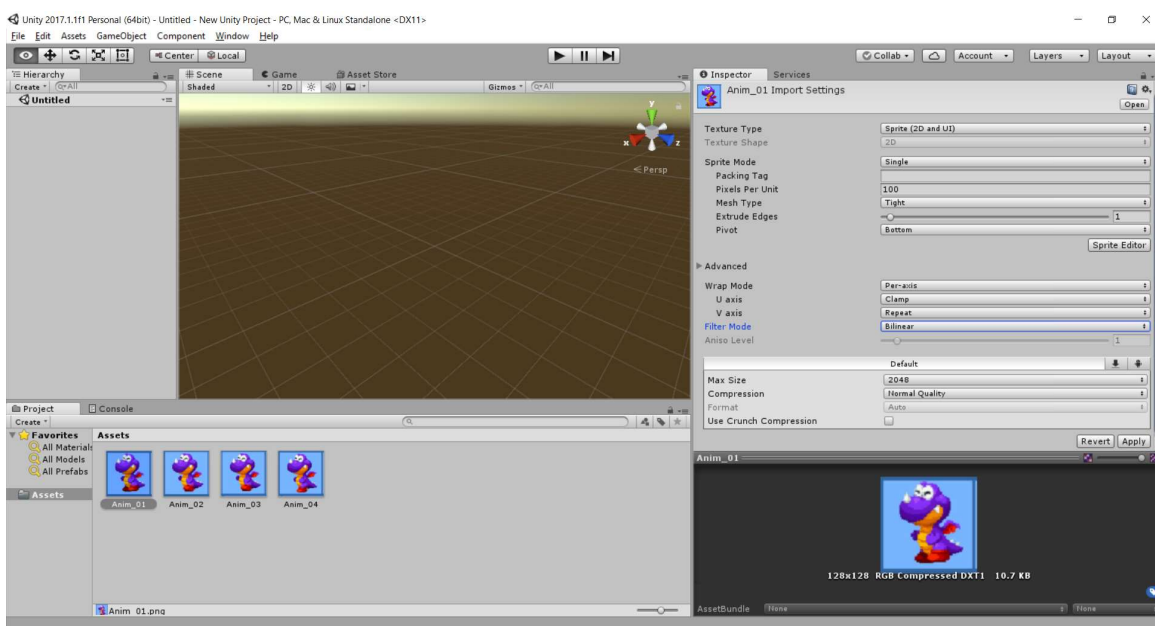


Рисунок 5.3 - Налаштування імпортованих текстур спрайтів

Імпортовані кадри тепер налаштовані і готові до анімації спрайтами в сцені.

Атлас спрайтів

Спрайти можуть бути зібрані разом на одному аркуші спрайтів, який називається текстурою атласу (Texture atlas). Іншими словами, всі кадри в анімації можуть бути поміщені в рядки і стовпці одного файлу текстури, а не розміщуватися в окремих файлах (рис.5.4).



Рисунок 5.4 - Текстура атласу спрайтів

Цей вид листів спрайтів також підтримується Unity, але він вимагає деяких додаткових налаштувань, в тому числі і на етапі імпорту.

Щоб імпортувати атлас спрайтів, просто перетягніть текстуру в панель проекту Unity. Атлас буде імпортовано як звичайний файл текстури. Після імпорту він з'явиться в панелі проекту (рис. 5.5).

Після імпорту ви повинні будете налаштувати текстуру для роботи в якості спрайту. Щоб зробити це, виберіть текстуру в панелі проекту. У інспекторі об'єктів виберіть **Sprite** для опції типу текстури **Texture Type**. Потім виберіть **Multiple** для опції режимів спрайту **Sprite Mode**, тому, що файл текстури містить кілька кадрів, а не один. Наприкінці натисніть на кнопку **Apply**.

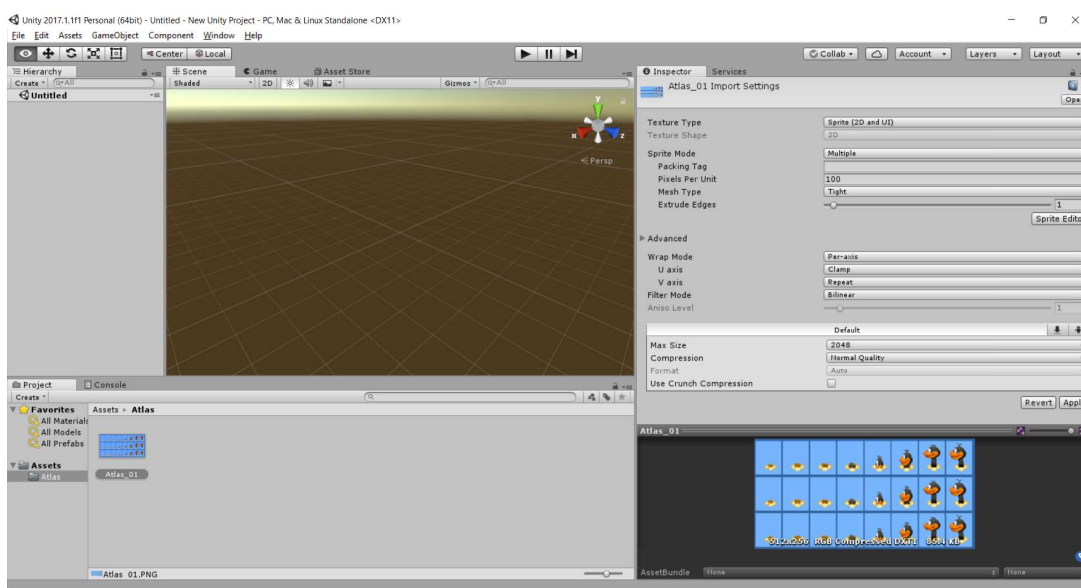


Рисунок 5.4 - Налаштування імпортованої текстури атласу спрайтів
Виберіть **Multiple** для опції **Sprite Mode**, давши зрозуміти Unity, що в текстурі присутні кілька кадрів, але Unity ще не знає, де саме вони знаходяться в текстурі. Щоб визначити це, можна використовувати редактор спрайтів (**Sprite Editor**). Цей інструмент доступний за допомогою кнопки **Sprite Editor** в інспекторі об'єктів, коли текстура обрана в панелі проекту (рис. 5.5 – 5.6).

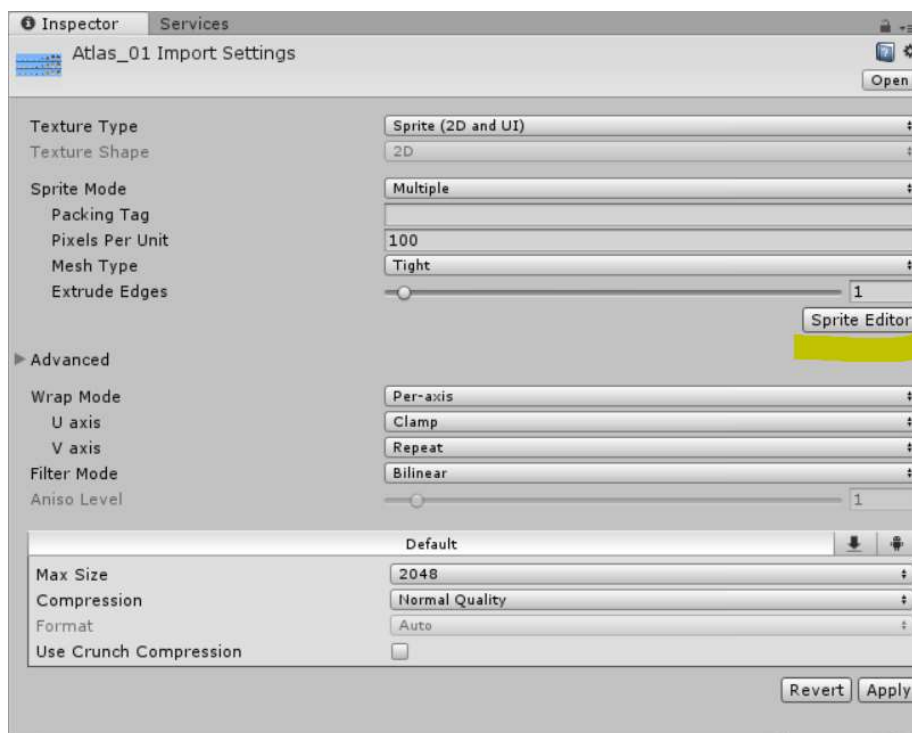


Рисунок 5.5 - Інспектор об'єктів з кнопкою **Sprite Editor**

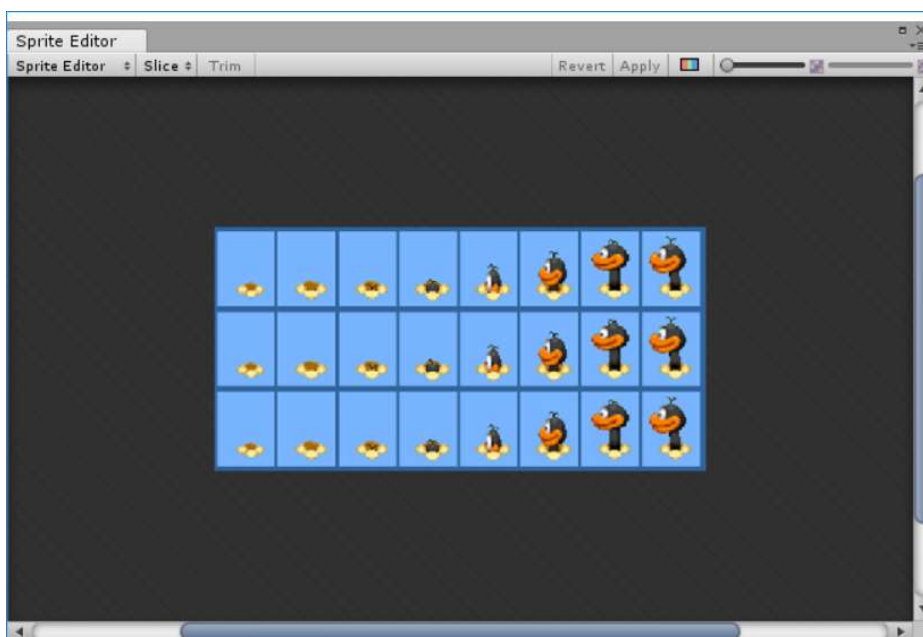


Рисунок 5.6 - Редактор **Sprite Editor**

У діалоговому вікні редактора спрайтів **Sprite Editor** ви зможете вручну вказати кордони для кожного спрайту, просто перетягуючи рамку навколо кожного з них в зображенні.

Ви можете скористатися альтернативним підходом, автоматично згенерувати фрагменти спрайту, задавши розміри плитки. У текстурі з файлів на рис.5.6, кожен спрайт має розміри 62×80 . Для їх поділу натисніть на кнопку **Slice** в верхньому лівому кутку вікна редактора спрайтів **Sprite Editor** (рис. 5.7).

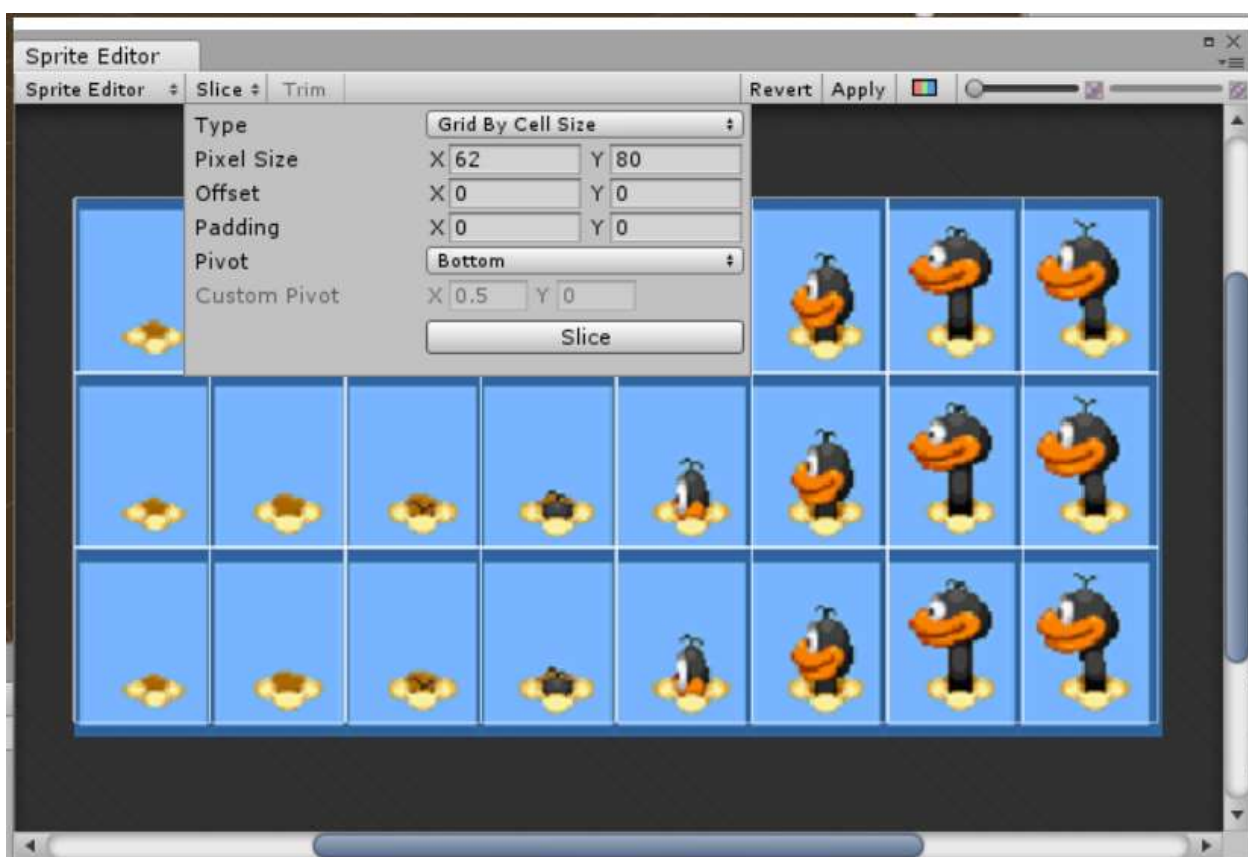


Рисунок 5.7 - Редактор **Sprite Editor**. Генерація слайдів

Натискання кнопки **Apply** в вікні редактора спрайтів призведе до генерації послідовності об'єктів спрайтів, які з'являться в панелі проекту в якості незалежних суб'єктів, готових до анімації (рис. 5.8).

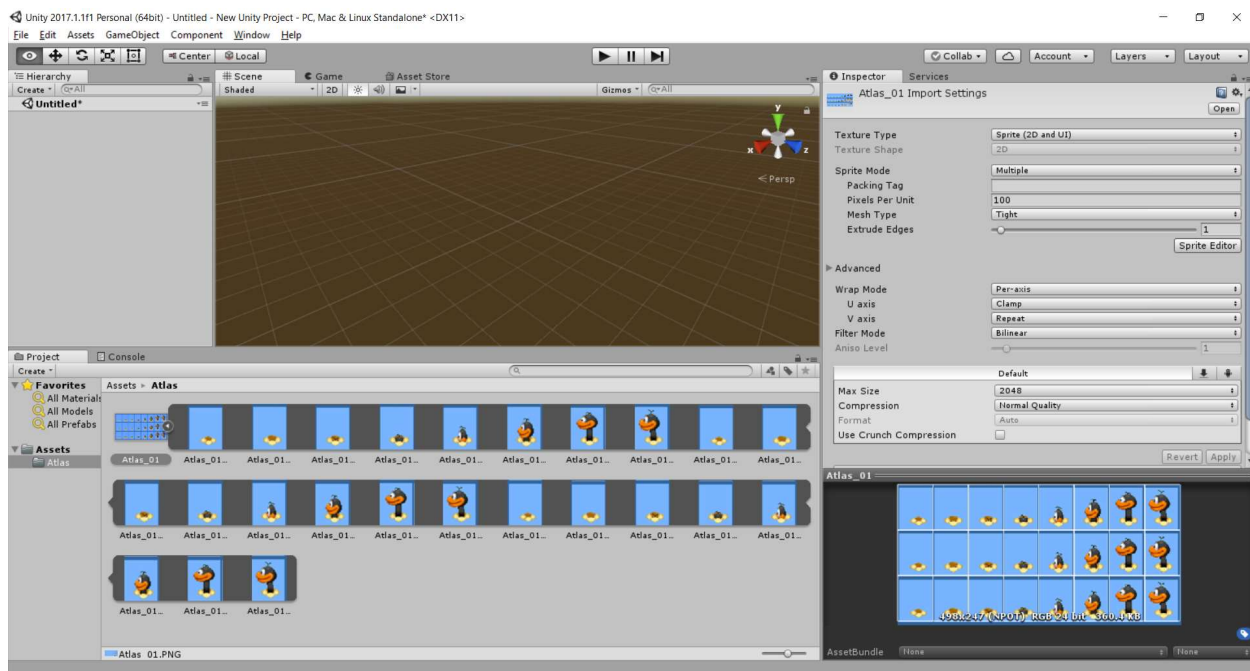


Рисунок 5.8 - Згенерована послідовність об'єктів спрайтів

Анімація за допомогою спрайтів

Після того як ви імпортували набір об'єктів спрайтів (не має значення, як окремі зображення або як текстуру атласу), ви готові запусити анімацію з їх участю.

На панелі проекту Unity виберіть всі спрайти, що входять в повну послідовність анімації, а потім перетягніть їх всі разом на панель ієрархії сцени. Якщо ви зробите це, з'явиться діалогове вікно збереження **Save**, пропонуючи вам створити нову послідовність анімації (файл з розширенням **.anim**). Задайте ім'я файлу анімації (наприклад, **PlayerRun.anim**) і виберіть кнопку **Save** (рис. 5.9).

У сцені буде створено об'єкт спрайту, він повинен стати видимим на вкладках сцени і гри. Якщо ви протестуєте вашу сцену, то ваш персонаж буде анімаційним, ви повинні побачити всі зображення послідовності. Зазначимо, що Unity створило новий актив Animation Clip (.anim), який визначає ключову послідовність кадрів. Крім того, Unity створило контролер Mecanim для запуску відтворення анімації при завантаженні рівня і для управління швидкістю відтворення, додало компонент аніматора до об'єкта спрайту в сцені, щоб зв'язати об'єкт з його даними по анімації (рис. 5.10).

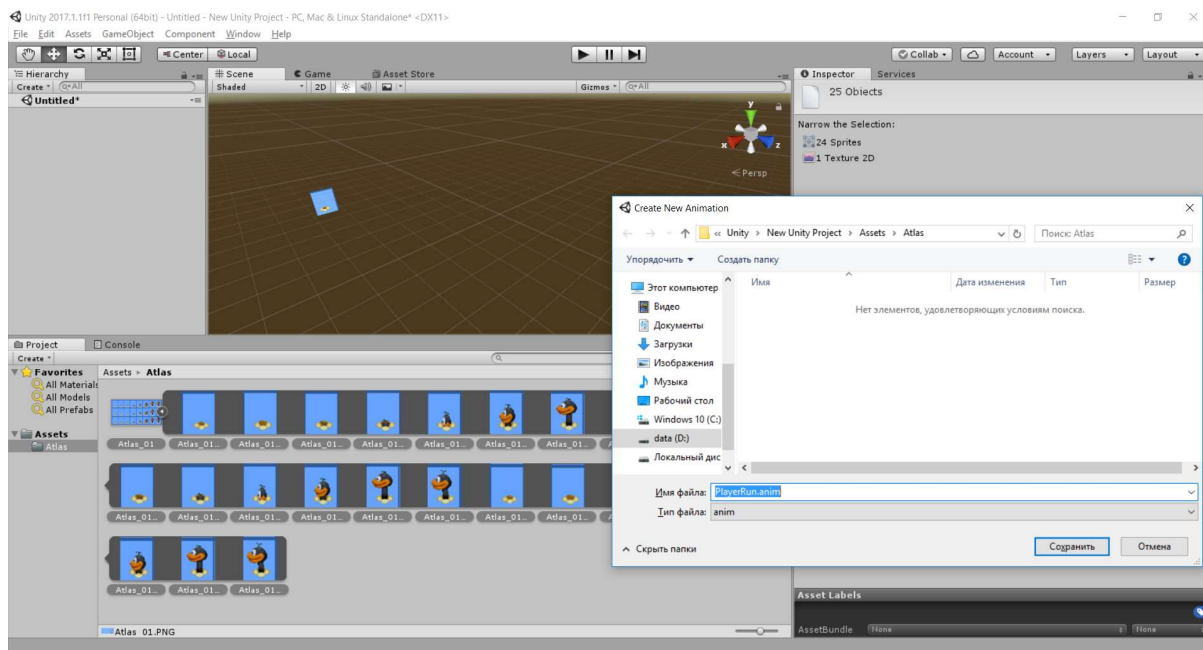


Рисунок 5.9 – Створення файлу анімації

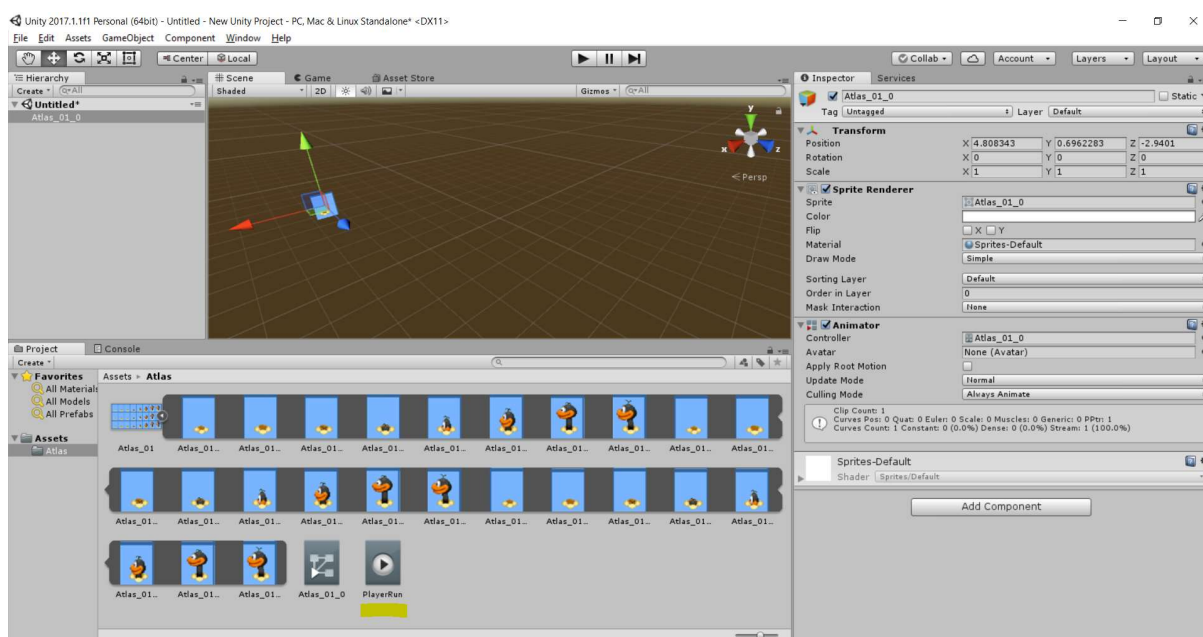


Рисунок 5.10– Створення нового активу Animation Clip (.anim) та контролеру Mecanim

Налаштування анімації за допомогою інструменту Mecanim

Якщо анімація спрайтами відтворюється занадто швидко або занадто повільно, то ви повинні будете відредагувати діаграму спрайту інструменту Mecanim.

Для досягнення цієї мети виберіть об'єкт спрайту в сцені. В інспекторі об'єктів двічі клацніть на актив **Animation Controller** всередині слота **Controller** компонента **Animator** (рис. 5.11).

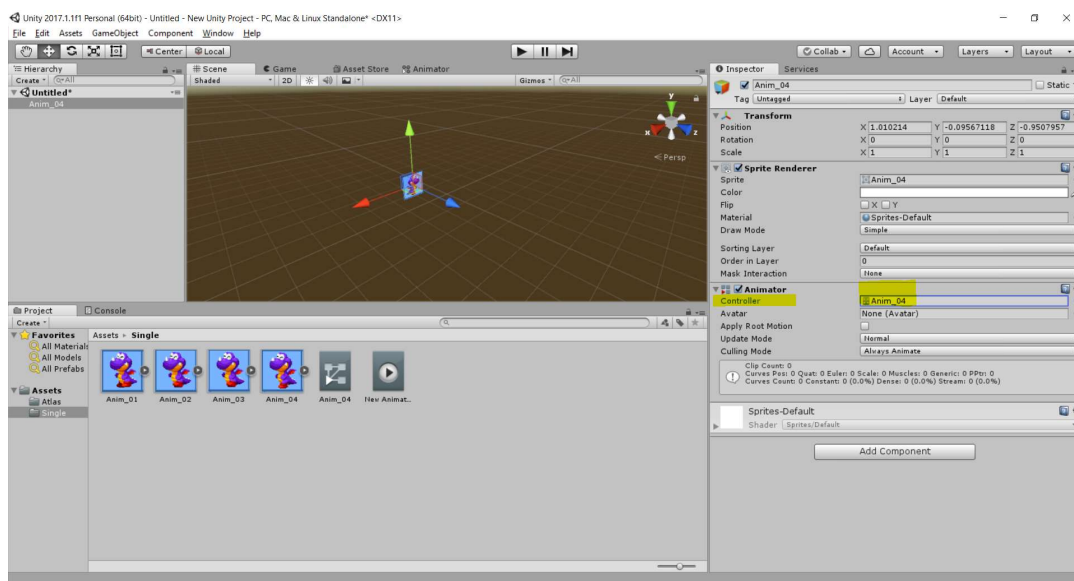


Рисунок 5.11– Налаштування анімації

Подвійний вибір активу **Animation Controller** спрайту викликає діаграму інструменту **Mecanim** для спрайту. З неї можна керувати швидкістю анімації. Ця діаграма містить кілька вузлів, пов'язаних між собою.

У діаграмі натисніть на вузол **Anim_04**, який представляє в діаграмі спрайт анімації, щоб вибрати його і переглянути його властивості в інспекторі об'єктів (рис. 5.12).

Швидкість анімації регулюється за допомогою параметра **Speed**. Значення 0 - зупинка, значення 1 - швидкість за замовчуванням, значення 0,5 - половинна швидкість, 2 - подвійна швидкість і т.д. Якщо анімація занадто повільна, то збільште значення швидкості, і якщо вона занадто швидка, то зменште значення швидкості. Після того як ви закінчите, просто запустіть гру, щоб побачити ефект.

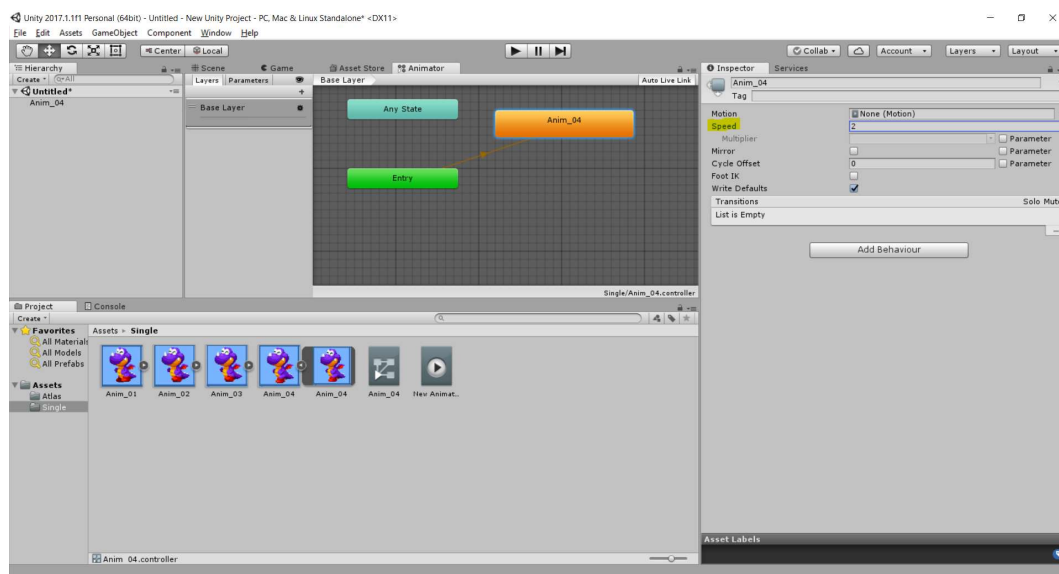


Рисунок 5.12– Діаграма інструменту Mecanim

За замовчуванням спрайт анімації зациклений, тобто відтворюється знову і знову без кінця. Коли відтворення анімації завершує цикл, воно просто повертається до початку, і відбувається нове відтворення.

Іноді потрібно відтворити анімацію тільки один раз, а потім зупинити. Для цього потрібно отримати доступ до даних анімації (вони знаходяться всередині активу `.anim`) і відкоригувати їх властивості. Щоб зробити це, виберіть актив спрайту анімації в панелі проекту. Активи анімації помічені іконкою **Play** і мають ім'я, яке ви присвоїли їм при їх створенні. Після вибору активу скиньте прапорець **Loop Time** в інспекторі об'єктів (рис. 5.13). Тепер анімацію буде відтворено тільки один раз.

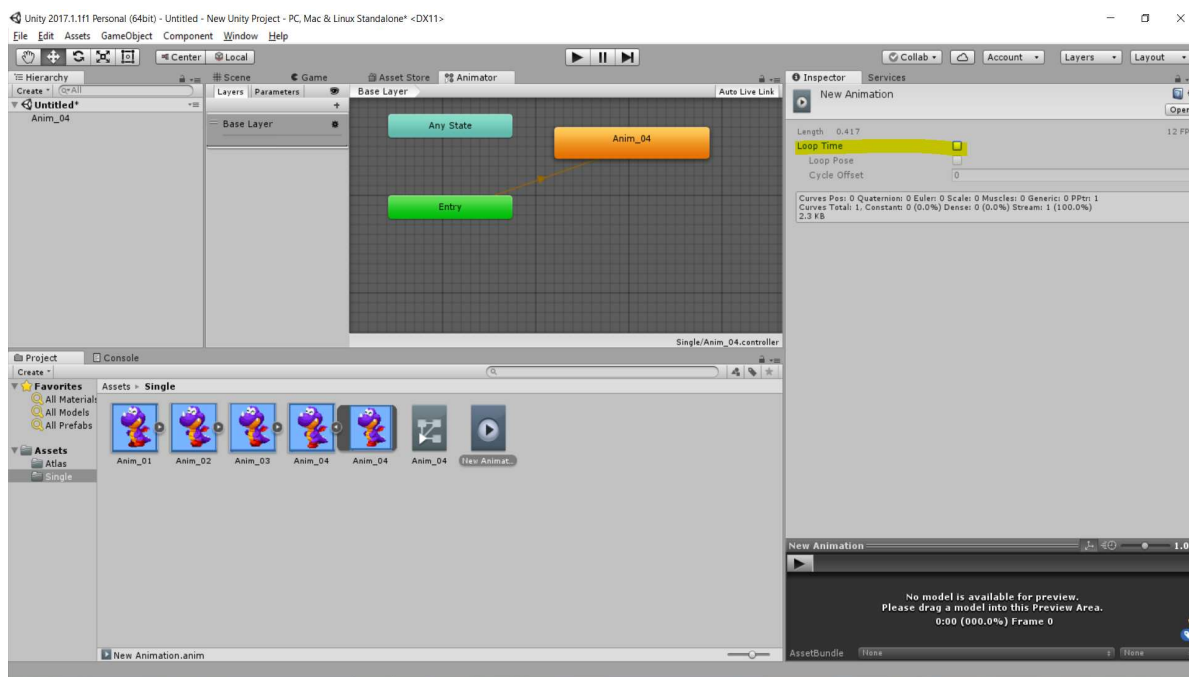


Рисунок 5.13– Налаштування анімації. Властивість **Loop Time**

Редагування кадрів анімації. Вікно **Animation**.

Якщо спрайт анімації має багато кадрів, то, можливо, при генерації анімації вони будуть розташовані в неправильному порядку, в результаті чого деякі кадри стануть з'являтися раніше або пізніше потрібного моменту.

Редагувати кадри анімації можна у вікні **Animation**. Це вікно можна отримати, перейшовши до **Window** → **Animation** в головному меню (рис. 5.14).

При відкритому вікні **Animation** виберіть об'єкт спрайту в сцені, і його анімаційні дані автоматично будуть відображені на часовій шкалі. Графік містить в собі весь період анімації, від початку і до кінця. Ромбовидні символи, рівномірно розподілені по шкалі, є ключові кадри, в яких змінюються зображення спрайтів. Ви можете вибрати конкретний ключовий кадр на часовій шкалі, обравши його. Потім ви можете переглянути його спрайтові властивості в інспекторі об'єктів (рис. 5.15).

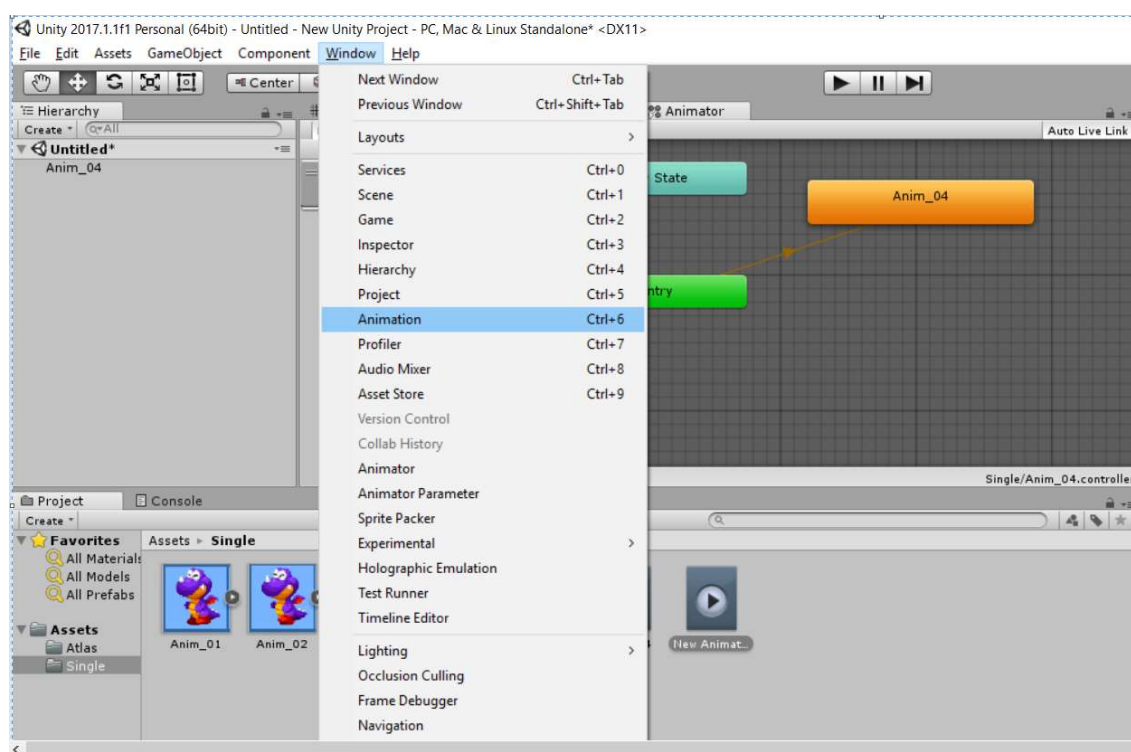


Рисунок 5.14– Вікно **Animation**

Вибравши ключовий кадр графіка, в якому знаходиться те зображення, яке не повинно бути показано в цей час, ви можете це легко виправити, натиснувши на поле **Sprite** в інспекторі об'єктів і вибравши новий спрайт в браузері спрайтів. Unity автоматично внесе потрібні зміни, і обраний вами спрайт буде відображатися в цьому ключовому кадрі (рис. 5.16).

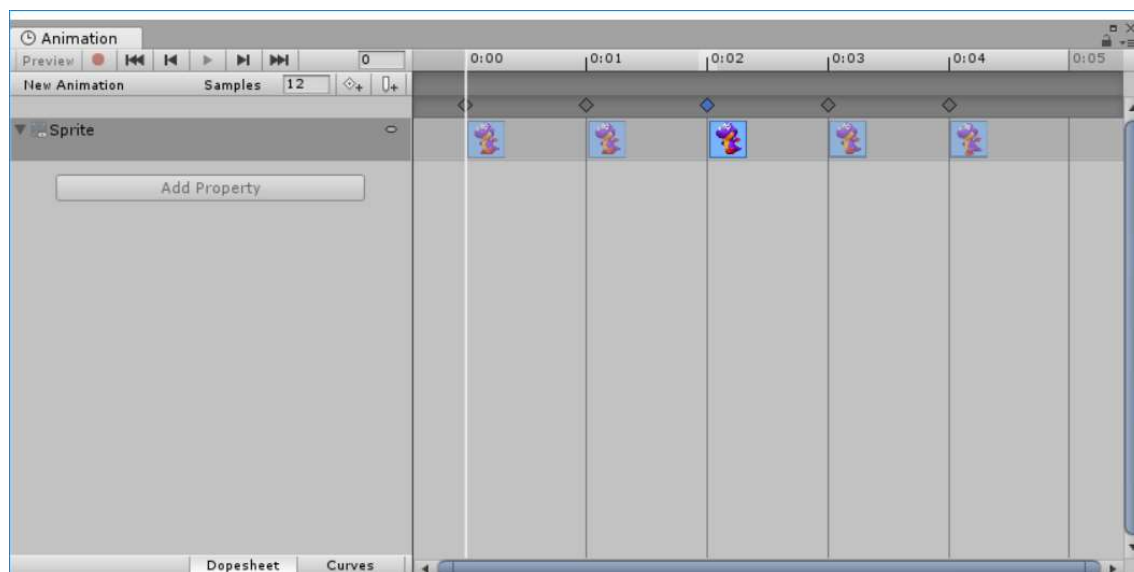


Рисунок 5.15– Ключові кадри анімації

Завдання до лабораторної роботи №5

1. Створити новий проект Unity.
2. Здійснити імпорт спрайтів кадр за кадром з окремих файлів.
3. Здійснити імпорт спрайтів як текстуру атласу, що містить набір кадрів.
4. Зібрати кадри в анімаційну послідовність, використовуючи інструменти створення анімації Unity (такі як **Mecanim** і **Animation**).
5. Здійснити налаштування швидкості анімації, зациклення і переміщення кадрів анімації.
6. Використовуючи отримані активи, створити анімований персонаж.

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 інспекторі об'єктів?

ЛАБОРАТОРНА РОБОТА № 6

ТЕМА: «РОЗРОБКА КРОСПЛАТФОРМНИХ 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 кросплатформних 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.

Створення ігрового проекту для ОС Windows

Unity 3D Personal (базова безкоштовна версія) дозволяє створювати додатки для великої кількості ігрових платформ: Windows, Mac OS X, Linux, Web, iOS, Android.

А за допомогою особливих ліцензованих модулів Unity 3D дозволяє

с
т Під створенням додатків для великої кількості ігрових платформ мається на увазі генерація прикладних пакетів, які запускаються на цих платформах. На кожній платформі (Windows, Android та ін.) своя форма пакета, але як тільки ви згенерували виконуваний файл, з'являється можливість поширювати гру і грати в неї без прив'язки до Unity.

в Один проект Unity можна розгорнути на різних платформах - його не потрібно кожного разу генерувати заново.

т Подивитися всі варіанти платформ можна у вікні «**Build Settings**». Відкрити вікно «**Build Settings**» можна за допомогою меню «**File**→**Build Settings**» (рис. 6.1).

д Вікно «**Build Settings**» містить мінімальні налаштування, необхідні для збірки проекту. Перше, що необхідно зробити - додати в проект ігрову сцену. Зробити це можна за допомогою кнопки «**Add Open Scenes**», або просто перемістити файл сцени з вікна проекту в поле «**Scenes In Build**».

т
к
и

д
л
я

XBox 360, XBox One, PlayStation 4, Windows Phone 8 та ін.

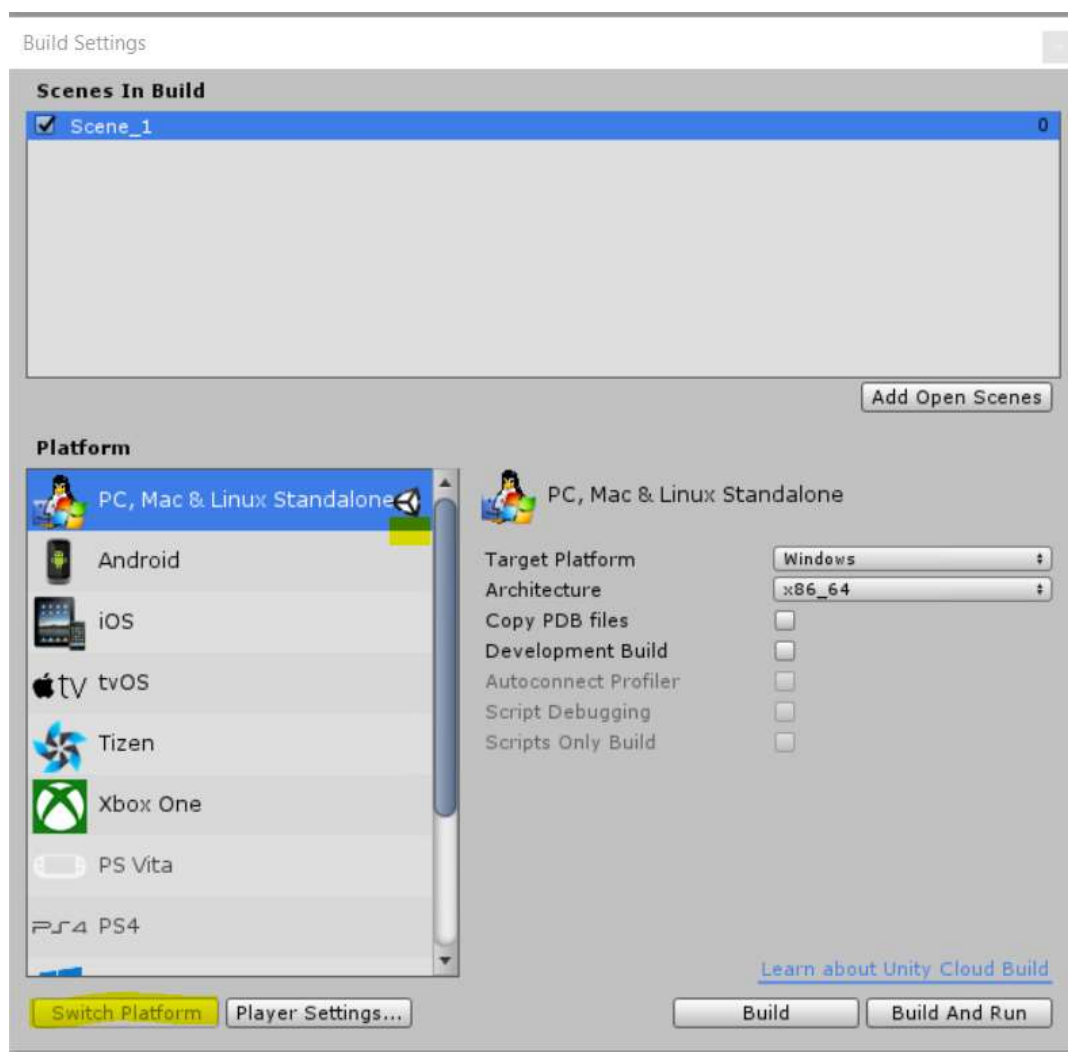


Рисунок 6.1– Вікно «**Build Settings**»

Зверніть увагу на цифру нуль, праворуч від назви файлу сцени. Це її порядковий номер. Якщо сцен в вашому проєкті кілька - пам'ятаєте, що сцена, яка повинна буде завантажуватися першою, повинна отримати ідентифікатор нуль. Інакше, Unity автоматично завантажить замість вступного ролика сцену, що випадково отримала головний ідентифікатор.

Перелік платформ, які підтримує **Unity**, подано у лівій частині вікна. Активні платформи позначено значком **Unity**. Достатньо виділити платформу з цього переліку та натиснути кнопку «**Switch Platform**».

Далі праворуч на вкладці «**Target Platform**» вибираємо платформу «**Windows**», на вкладці «**Architecture**» вибираємо архітектуру, яка буде підтримуватися проєктом.

Для генерації ігрового проєкту у нижній частині вікна знаходяться кнопки «**Build**» і «**Build and Run**». Кнопка «**Build and Run**» відрізняється від кнопки «**Build**» тим, що автоматично запускає згенерований ігровий додаток.

Відкрити перелік налаштувань ігрового додатку на панелі «**Inspector**» можна за допомогою кнопки «**Player Settings**» (рис. 6.2). Ці налаштування контролюють різні аспекти готового додатка.

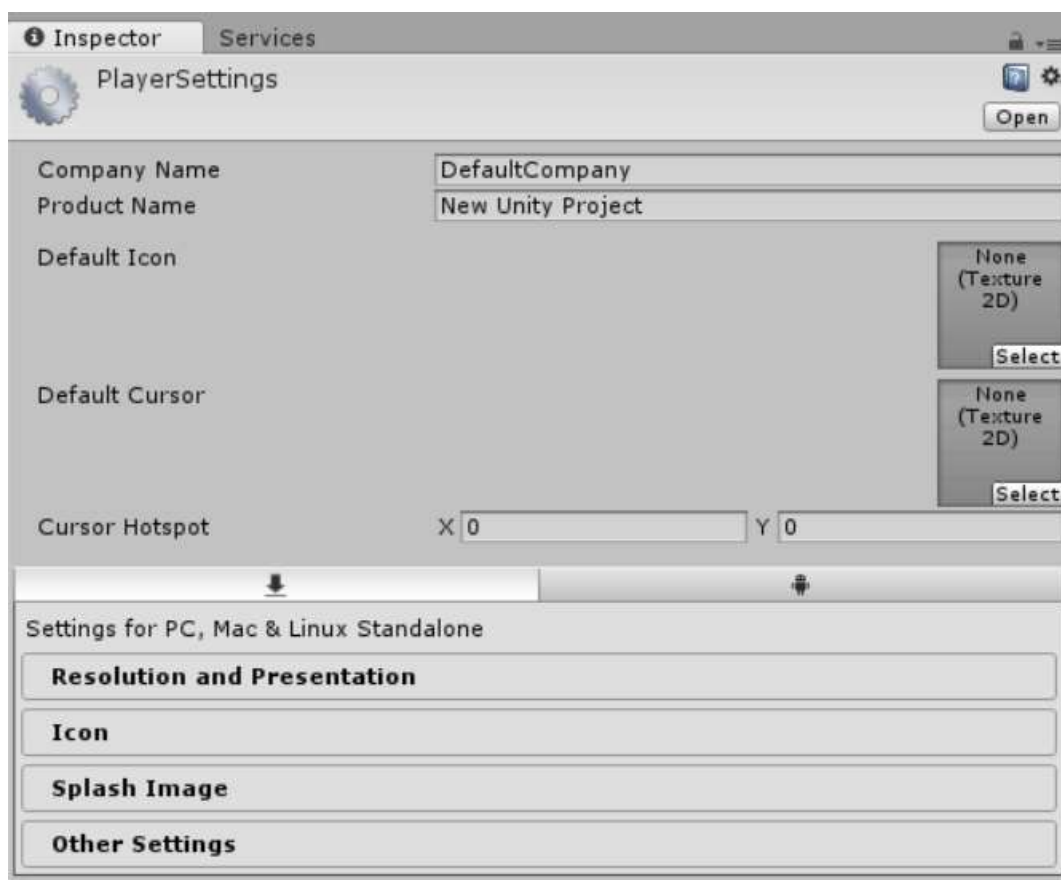


Рисунок 6.2– Вікно «**Player Settings**»

Поля «**Company Name**» і «**Product Name**» дозволять вказати назву гри і розробника. За допомогою «**Default Icon**» можна встановити іконку ігрового додатку. Для цього необхідно обрати зображення (при необхідності імпортувати його) на вкладці «**Project**» та перетягнути його у розділ «**Texture 2D**» на панелі «**Inspector**». Крім того, за допомогою «**Default Cursor**» можна також задати зображення курсора.

Після натискання на кнопку «**Build**» відкриється вікно вибору файлу, в якому потрібно вказати адресу для генерації пакета ігрового додатку. Відразу після вказівки місця розташування почнеться процес побудови, після чого **Unity** створить виконуваний файл для активної в даний момент платформи. У нашому випадку це платформа «**Windows**».

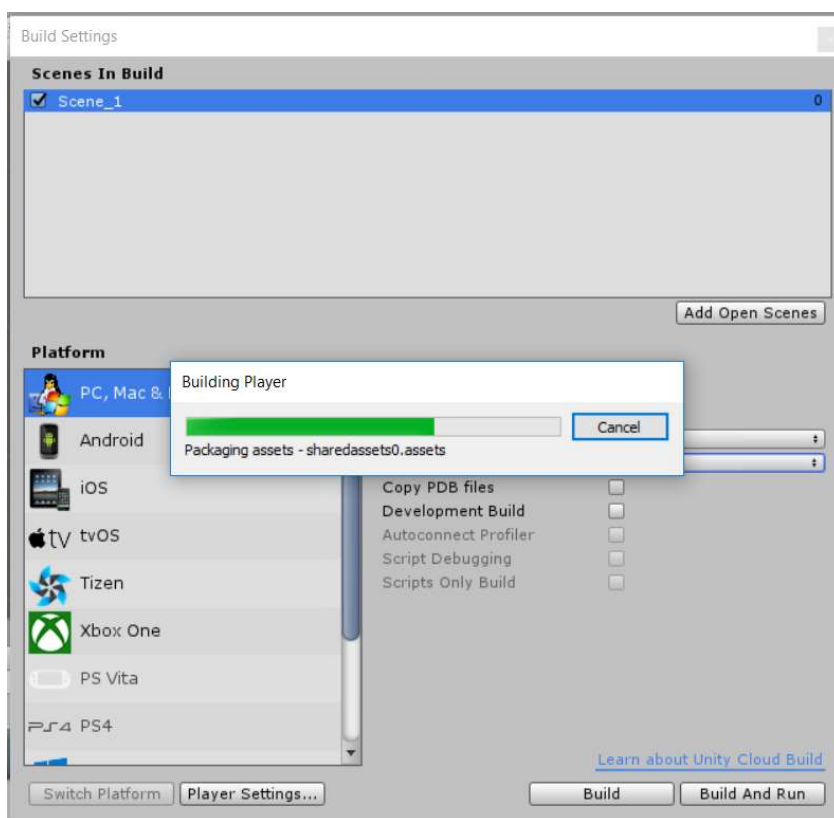


Рисунок 6.3– Процес побудови ігрового проекту

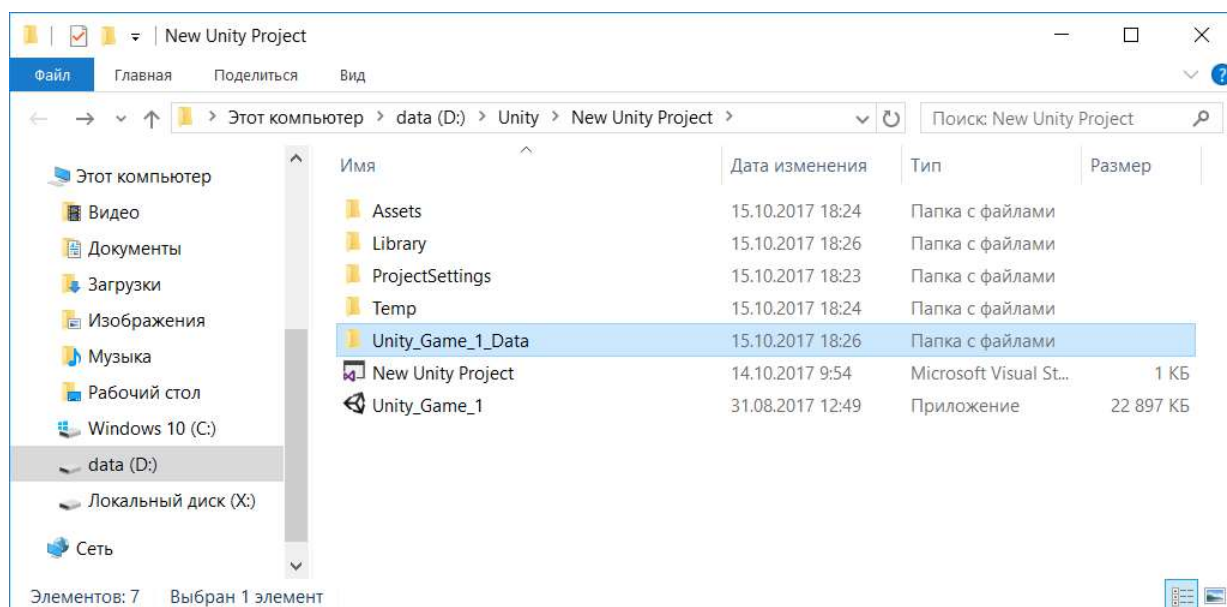


Рисунок 6.4– Файли проекту для платформи «Windows»

На додаток також впливають налаштування, доступ до яких здійснюється за допомогою меню «**Edit**». Зокрема, саме тут налаштовується візуальна якість готового додатка.

Виберіть в меню «**Edit**» команду «**Project Settings**», а потім в додатковому меню - команду «**Quality**» (рис.6.5).

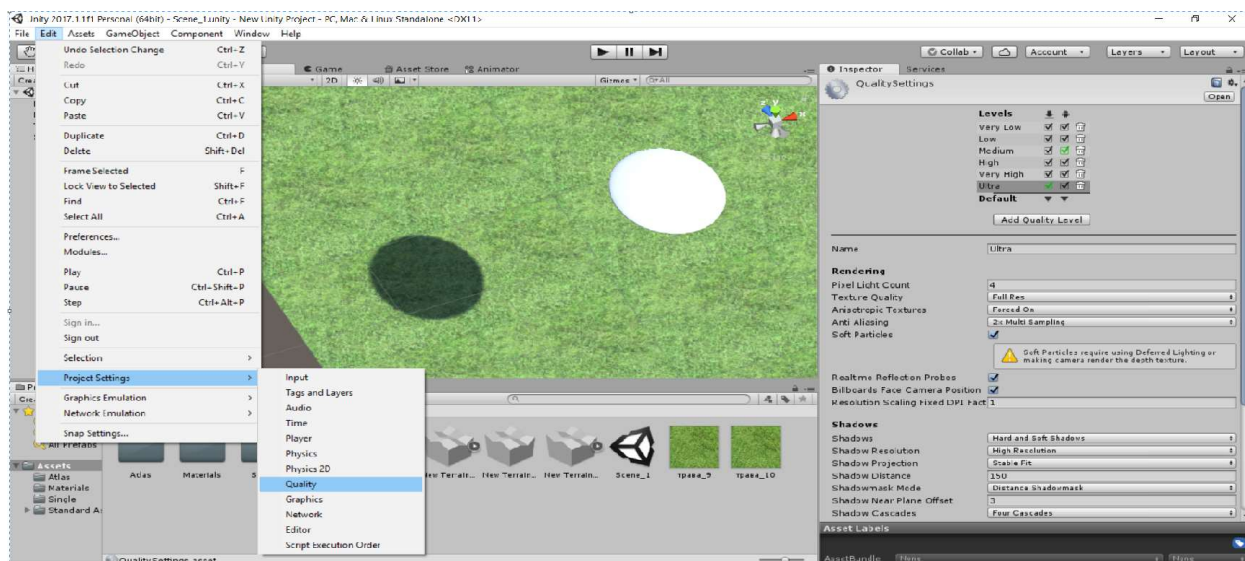


Рисунок 6.5– Меню «Edit»

На панелі «Inspector» з'являться елементи управління якістю, найбільш важливими з яких є прапорці в розташованій зверху групі. У верхньому рядку знаходяться значки можливих платформ, а збоку вказані варіанти налаштувань якості. Встановлені прапорці показують доступні для даної платформи налаштування, а ті, що виділені зеленим - поточні налаштування.

У більшості випадків за замовчуванням застосовується варіант «Very Low» (мінімальна якість), але можна вибрати варіант «Very High» або «Ultra» (максимальна якість) (рис. 6.6).



Рисунок 6.6– Елементи управління якістю

Створення ігрового проекту для ОС Android

Unity має можливість генерувати файли формату **APK** (Android Application Package). Для цього необхідно додати в **Unity** шлях до середовища розробки **Android SDK**, яке вже містить необхідний компілятор.

Android SDK – це середовище розробки додатків для операційної системи **Android**. **Android SDK** дозволяє створювати і тестувати **Android**-додатки, що використовують камеру мобільного пристрою, акселерометр, компас, дані GPS, доступ по Bluetooth, Wi-Fi, EDGE і 3G.

Android SDK підтримує роботу з мультимедійним контентом (аудіо, відео, зображення в форматах MPEG4, H.264, MP3, AAC, AMR, JPG, PNG і GIF), базами даних SQLite, інтегрованим браузером на движку WebKit, віртуальною машиною Dalvik, GSM телефонією і т.д. Користувачі **Android SDK** мають можливість тестувати створені ними додатки за допомогою вбудованого емулятора.

Завантажити **Android SDK** можна з офіційного сайту за посиланням <https://developer.android.com/studio/index.html>

Завантажте **Android SDK** та вкажіть шлях до цього файлу на вкладці «**External Tools**» у вікні «**Unity Preferences**».

Відкрити вікно «**Unity Preferences**» можна за допомогою меню «**Edit**→**Preferences**» (рис. 6.7).

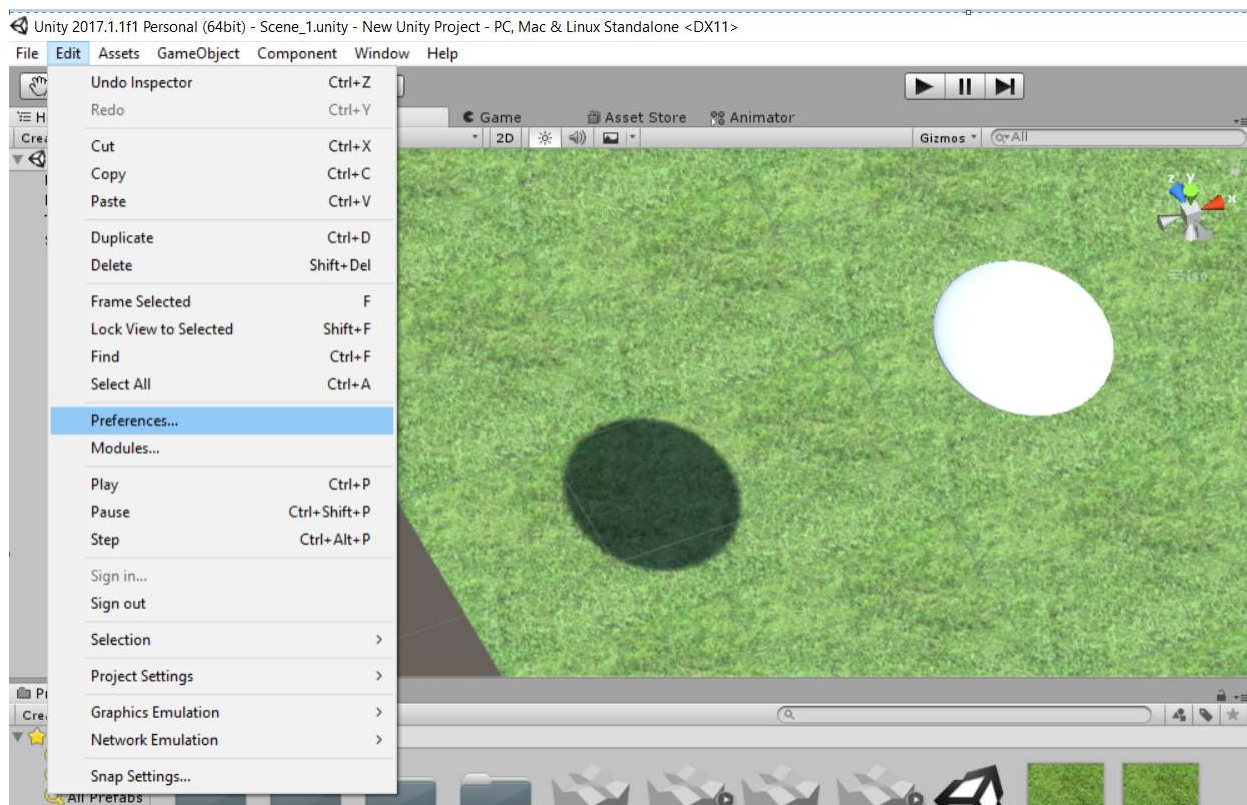


Рисунок 6.7– Меню «**Edit**→**Preferences**»

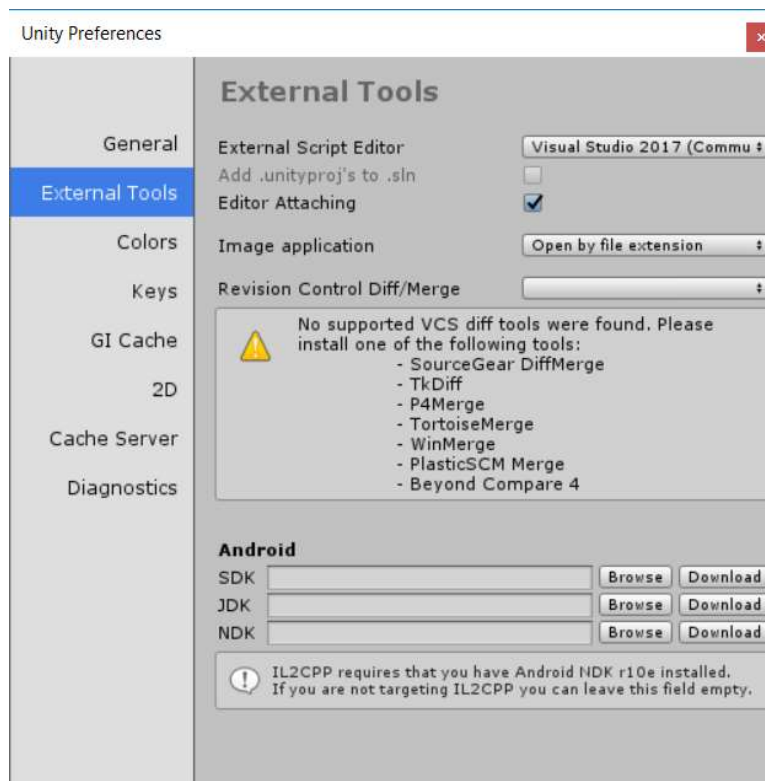


Рисунок 6.8– Вкладка «External Tools» у вікні «Unity Preferences»

Після цього необхідно задати параметр «**Package Name**» в розділі «**Other Settings**» на панелі «**Inspector**» у вигляді **com.назвакомпанії.назвапродукта** та запустить процес збірки ігрового проекту за допомогою кнопки «**Build**».

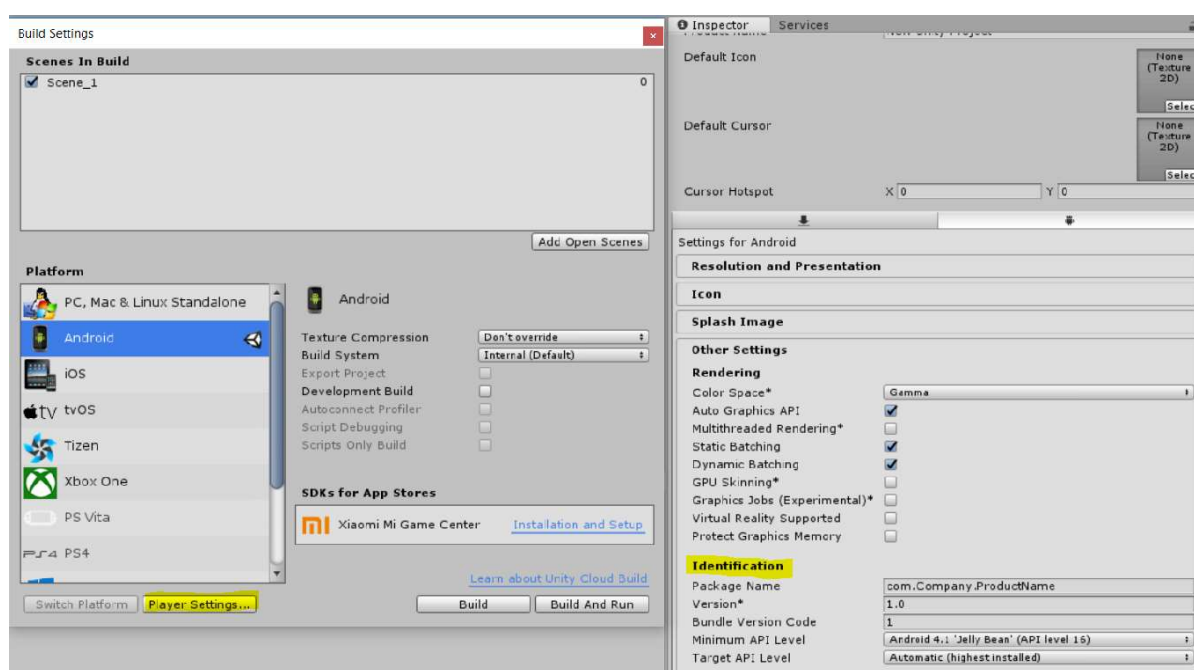


Рисунок 6.9– Налаштування параметру «Package Name»

Отриманий **АРК** пакет ігрового додатку необхідно встановити на мобільний пристрій.

Завдання до лабораторної роботи №6

1. Завантажити проект Unity.
2. Здійснити налаштування інструментів збірки ігрового проекту для платформи Windows.
3. Отримати пакет ігрового додатку для платформи Windows.
4. Здійснити налаштування інструментів збірки ігрового проекту для платформи Android.
5. Отримати пакет ігрового додатку для платформи Android.
6. Встановити **АРК** пакет ігрового додатку на мобільний пристрій.

Контрольні запитання

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

Література:

1. Горн А. Основы анимации в Unity/ пер. с англ. Р. Рагимова. – М.:ДМК Пресс, 2016. – 176 с.: ил.
2. George A Duckett Game Development With Unity: Questions and Answers. - CreateSpace Independent Publishing Platform, 2016. – 268 p.
3. Joseph Hocking Unity in Action Multiplatform game development in C# with Unity 5. - Foreword by Jesse Schell. – 352 p.
4. Официальный сайт Unity3d [Электронный ресурс] // режим доступа: <https://unity3d.com/>
5. Кенни Ламмерс Шейдеры и эффекты в Unity. Книга рецептов. - ДМК Пресс, 2014. – 274 с.
6. Хорхе Паласиос Unity 5.x. Программирование искусственного интеллекта в играх. - ДМК Пресс, 2016. – 272 с.
7. Крис Дикинсон Unity 5 Game Optimization. - ДМК Пресс, 2017. – 306 с.