РАЗРАБОТКА ИГРЫ В UNITY

с нуля и до реализации

Дмитрий Денисов

12+

Дмитрий Денисов

Разработка игры в Unity. С нуля и до реализации

«ЛитРес: Самиздат»

2021

Денисов Д. В.

Разработка игры в Unity. С нуля и до реализации /
Д. В. Денисов — «ЛитРес: Самиздат», 2021

ISBN 978-5-532-94186-1

Это руководство по разработке игры, в котором шаг за шагом описывается процесс ее создания с применением языка программирования C# и среды разработки Unity. Материал книги составлен таким образом, что практически каждый ваш шаг будет сопровождать скриншот с понятным описанием последовательности действий. Вы научитесь создавать игровые объекты и описывать логику их работы, создавать элементы ландшафта, настраивать камеру, игровые сцены, графический интерфейс, спецэффекты и звуковые эффекты. Материал практикума завершается публикацией игры на открытой онлайн платформе. После каждой главы в книге даны рекомендации по возможным модификациям игры таким образом, чтобы она получилась не похожей на ту, которую создаем мы. Издание предназначено для тех кто только знакомится с основной разработки игры в Unity. Неважно, как давно вы знакомы с Unity, это руководство, под силу выполнить любому человеку с базовыми навыками работы на компьютере.До встречи за пределами матрицы! Welcome.

ISBN 978-5-532-94186-1

Содержание

Дмитрий Денисов
Разработка игры в Unity.
С нуля и до реализации

Посвящается моей любимой жене Насте за ее ум и веру в наши победы, дочери Вике и сыну Паше за их позитив и любовь. Моим родителям Вадиму и Елене, маме моей жены Татьяне, за их заботу и поддержку.

I Trust in God

Введение

Компьютерные игры давно стали частью нашей культуры, через них можно донести мысль до широких масс, они развивают интеллект, реакцию и позволяют пережить целый спектр самых разнообразных эмоций.

В этой книге дано полноценное руководство по разработке игры на Unity с нуля. Без лишней воды и теории, только практика, потому что лучше один раз сделать что-то самостоятельно (даже по руководству), чем прочитать множество гайдов и посмотреть сотни видео на YouTube, но ничего не сделать. В отличие от большого количество книг по разработке и программированию, в этой вы найдете только ту информацию, которая непосредственно относится к созданию вполне конкретного игрового прототипа.

Это значит, что в книге не будет раздела по основам разработки на C# – языке программирования, который используется для написания сценариев в Unity. С другой стороны, вы получите всю информацию по созданию необходимых скриптов в создаваемой вами игре, а в приведенных листингах будут даны подробные комментарии. Это позволит сконцентрироваться на изучении только тех функций среды разработки, или возможностях языка программирования, которые нужны для выполнения поставленной задачи.

Автор убежден, что важнее заложить базовые знания, дав реализовать свой первый проект. А копнуть глубже и разобраться в тонкостях поможет множество других гайдов, статей, учебников и руководств из официальной документации, “разбросанной” по просторам интернета.

В практикуме предлагается разработать прототип игры «Dragon Picker», под который может быть адаптировано достаточно большое количество игровых механик и процессов. Смысл работы прототипа достаточно прост – нужно ловить различные предметы. Практикум разбит на основные разделы, в каждом из которых будут описаны основные этапы разработки игры, буквально пара абзацев теории, после чего мы сразу перейдем к разработке (программированию). Вид стартовой страницы готовой реализации приведен на рисунке ниже.

Цель практикума заключается в том, чтобы не только дать информацию о разработке прототипа, но и донести до читателя основные подходы к разработке игр такого типа. Другими

словами, поняв, как создается Picker, вы без труда сможете сделать свою игру, непохожую на “исходный” прототип, но со схожей механикой. В конце каждой главы будут даны рекомендации по вариантам изменения и доработки базового прототипа. Следуя им вы сможете не только глубже изучить работу с Unity, но и создать свою уникальную игру на основе предложенной базовой версии.

Иногда, по ходу выполнения практикума, будет предложено несколько вариантов реализации какой-либо задачи. Это сделано преднамеренно, так как гибкость выбора решения в различных ситуациях является крайне важным навыком для разработчика. Изучая и тестируя различные способы реализации одной и той же задачи, можно достичь наиболее оптимизированной работы игры.

По ходу изучения материалов практикума вы увидите, что программный код приводится в виде скриншота и в виде текста (листинга) одновременно. По убеждению автора эта избыточность необходима для простоты восприятия того, что требуется сделать. С одной стороны – вы сможете скопировать код пряма из материалов книги, с другой – будет приведен скриншот из среды разработки с удобным для восприятия представлением структуры кода.

Материалы книги не претендуют на гениальность изложенных подходов и решений в области разработки игр, однако они однозначно дадут вам базовые представления о порядке разработки компьютерной игры на Unity.

В практикуме показано, как создать собственную игру по принципу конструктора, используя бесплатно доступные текстур-паки, немного строк кода и фантазии. В любом случае процесс создания игры крайне сложен. Достаточно редки случаи, когда один разработчик способен создать достойную реализацию с приятной графикой и звуковыми эффектами. Но мы попробуем.

Структура практикума и как с ним работать

Практикум разбит на семь основных разделов. Каждый начинается с введения, в котором приводится общее описание того, что будет сделано в разделе. Каждый раздел будет приближать нас к созданию готовой игры и публикации ее на внешнем ресурсе, на котором с игрой сможет ознакомиться большое количество пользователей. Кроме того, каждый раздел завершается выводами, в которых помимо подведения итогов даются рекомендации по доработке игры. При желании вы даже можете вносить в каждом пункте модификации в игру на свое усмотрение и опубликовать решение, не похожее на то, которое создаем мы в рамках этого практикума.

В первом разделе мы установим необходимое программное обеспечение для разработки игры на Unity, а также создадим несколько тестовых примеров для проверки корректности работы установленного программного обеспечения.

Благодарности

При подготовке материалов данного практикума мне помогали несколько человек. Им всем я выражаю признательность и благодарю за потраченное время на поиск и устранение недочетов.

Фадеев Виталий, ведущий инженер платформы виртуализации VMware ПАО Сбербанк. Виталий принимал участие в подготовке разделов, связанных с установкой необходимого программного обеспечения и выгрузки игры на внешнюю платформу. Его рекомендации были ценны и позволили сделать пошаговые инструкции более доступнымми для понимания начинающим разработчикам игр.

Коровин Илья, Junior C#/Unity программист. Илья проделал весь практикум в самой сырой версии от начала и до конца, дал ряд ценных рекомендаций по объяснению некоторых важных функций, реализованных в разрабатываемой игре.

Тагатов Альберт, UX/UI дизайнер, графический дизайнер, да и просто хороший, отзывчивый человек. Альберт сделал так, чтобы печатный экземпляр книги было приятно держать в руках, а электронная версия притягивала своим внешним видом. Благодарю Альберта за разработку прекрасной обложки для издания, с которым вы работаете.

Об авторе

Автор просто человек, который любит Unity, делать игры и играть в чужие игры. Каких-то больших наград и достижений пока не имеет, но очень к этому стремится. Делает самую разную работу, начиная с создания междисциплинарных физических моделей в программном пакете ANSYS, заканчивая разработкой игр на Unity и работой в должности доцента Уральского Федерального Университета. Стремится объять необъятное и объяснить как можно большему количеству людей, что IT-сфера это просто и интересно, а самое главное – она меняет наш мир с невероятной скоростью.

Сообщество

В этом разделе приведены некоторые ссылки на внешние ресурсы, которые будут полезны при изучении принципов разработки игры на Unity и станут хорошим дополнением к материалам, данным в этом практикуме.

Сайт автора: https://sciencepub.ru/unity-book/. На сайте размещена информация о книге, на нем вы найдете ссылки на ресурсы, которые использовались при создании игры, а также исходные файлы к игре. Если после издания будут обнаружены неточности в описании или опечатки, то о них будет также сообщаться на указанном сайте.

Telegram-канал: https://t.me/EpicUnity. В телеграмме можно задать вопросы автору, или просто обсудить реализацию игры в кругу единомышленников.

YouTube-канал: https://www.youtube.com/sciencepub. На Youtube-канале со временем появится play-лист с разбором дополнительных заданий, которые даются в конце каждого раздела. Пока что вы найдете на канале достаточно много увлекательных видеоматериалов научно-популярной направленности.

GitHub: https://github.com/Den1sovDm1triy. В отдельном репозитории вы найдете исходники к игре.

SIMMER.io: https://simmer.io/@Den1sov. Мини-игра, разработанная в этом практикуме размещена на web-портале, ссылка указана на профиль автора.

Свои отзывы и предложения, а также информацию о найденных неточностях и опечатках вы можете отправить на личный e-mail автора: mr.denisov.dv@gmail.com.

Посмотреть на готовую реализацию игры Dragon Picker можно по ссылке: https://simmer.io/@Den1sov/dragon-picker.

Часть 1. Установка необходимого программного обеспечения

Введение

Чтобы создать игру нужны инструменты разработки. В качестве основного инструмента мы будем использовать Unity, а для написания программного кода понадобится среда разработки, например Microsoft Visual Studio. Unity – межплатформенная среда разработки компьютерных игр, которую выпустила и активно продвигает американская компания Unity Technologies. Unity позволяет создавать приложения, работающие на более чем 25 различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие.

В этом разделе вы:

– Пройдете регистрацию на портале Unity.com и получите свой Unity ID.

– Скачаете Unity Hub и установите среду разработки Unity.

– Скачаете Microsoft Visual Studio.

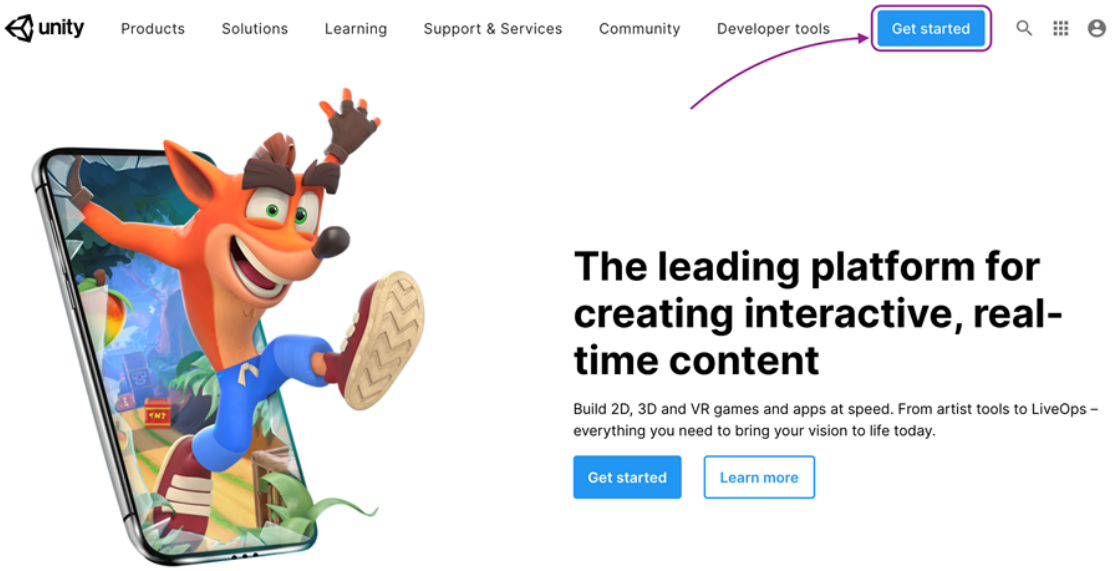
– Создадите тестовый проект, в котором проверите корректность работы скачанного и установленного программного обеспечения.

1.1 Установка среды разработки

Как было указано ранее, Unity – это среда разработки компьютерных игр и мультимедийных приложений. Для того чтобы начать с ней работу необходимо выполнить действия, описанные в пунктах ниже.

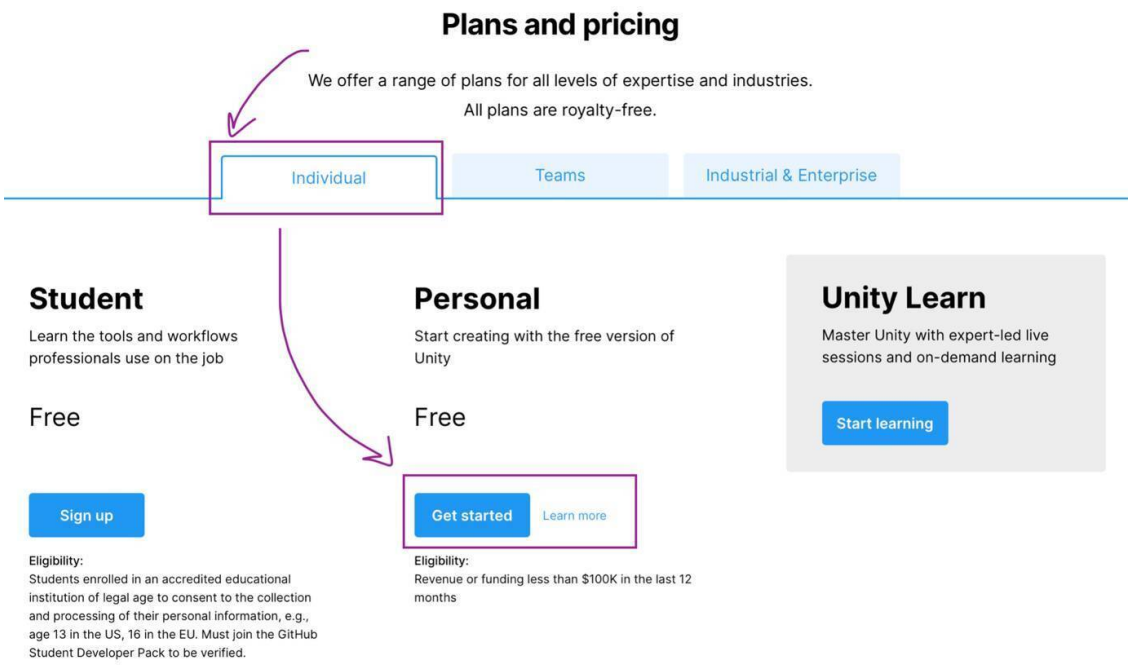
1. Откройте официальную страницу Unity (unity.com) в браузере. Вы сразу попадете на главную страницу компании Unity, где можете ознакомиться с актуальными новостями, поддерживаемыми платформами и предстоящими релизами продуктов компании.

2. На странице найдите кнопку Get Started и нажмите ее:

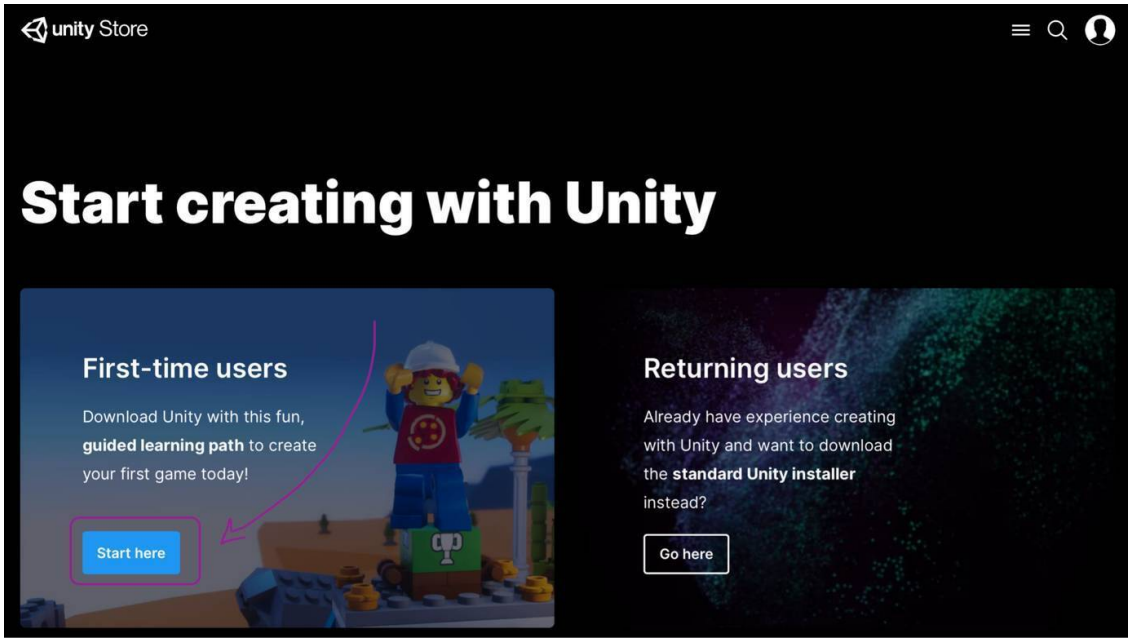


Crash Bandicoot: On the Run! – King

3. На открывшейся странице вам будет предложено выбрать план подписки. У компании Unity Technologies очень гибкая система, благодаря этому каждый разработчик может подобрать тарифный план под свои задачи. До тех пор, пока доход с вашего проекта за последние 12 месяцев не превышает $100 000 вы можете выбирать подписку Individual Personal (вкладка Individual – Personal – Get Started):



4. После этого вы попадете на страницу Unity Store. Если вы являетесь новым пользователем, нажмите кнопку Start Here:

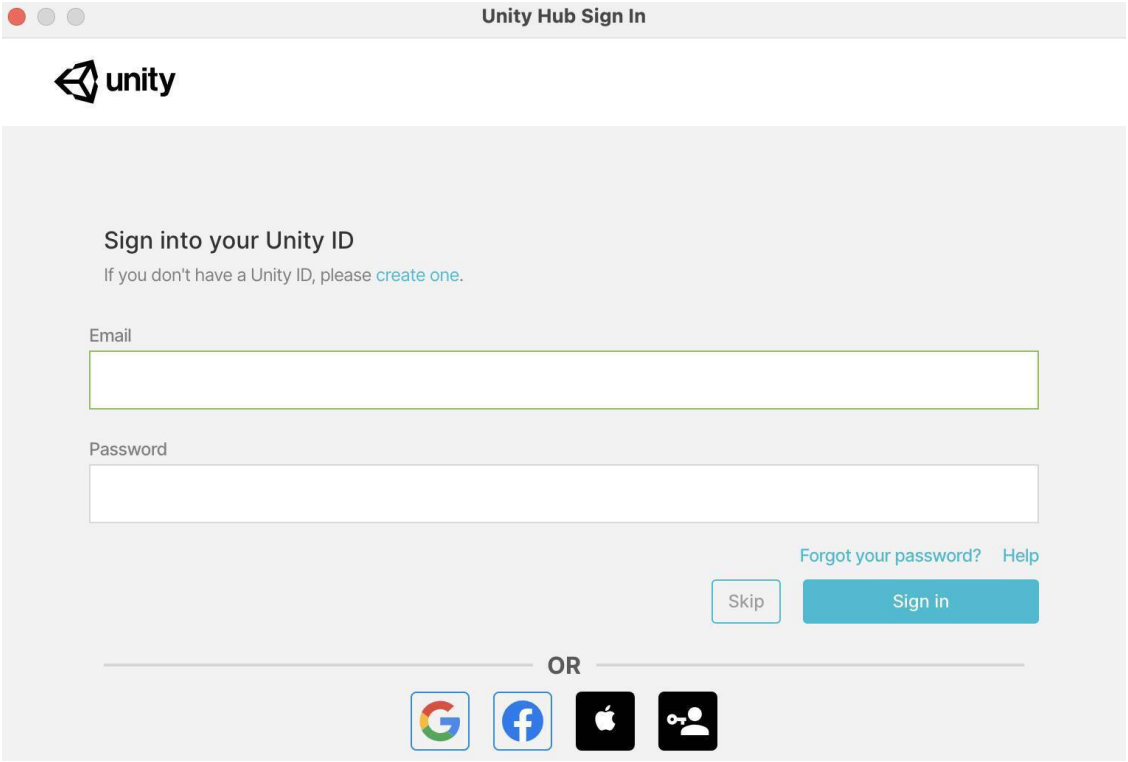


5. Далее появится всплывающее диалоговое окно, на котором нужно принять лицензионное соглашение. Нажмите кнопку Agree and download.

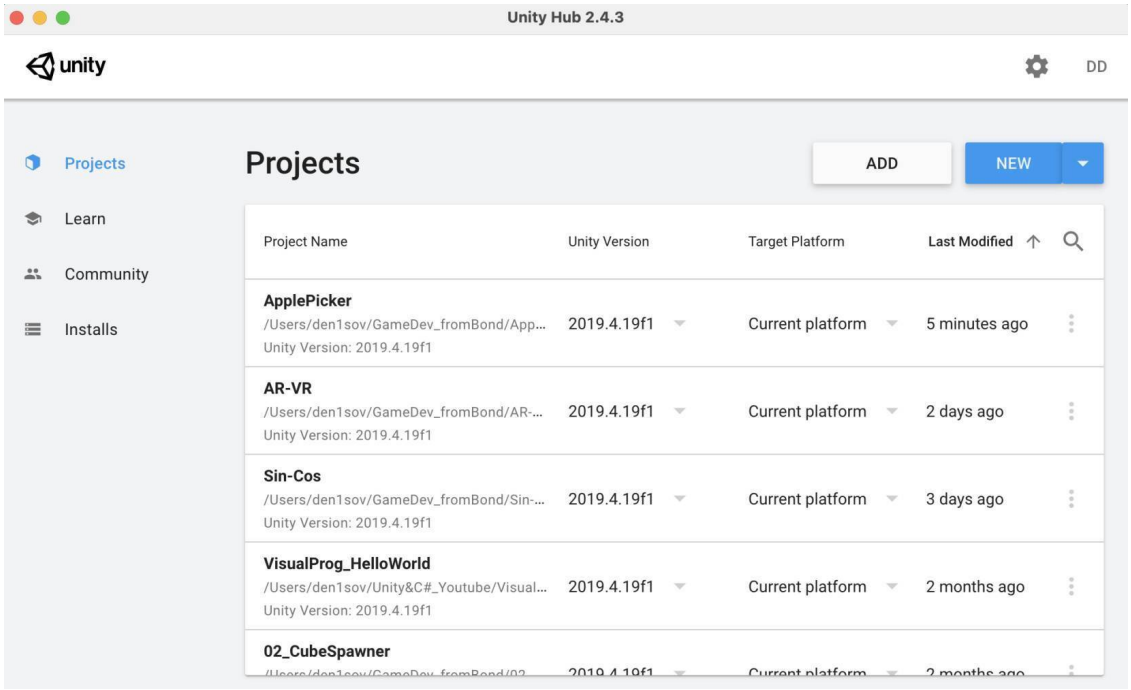
6. Начнется автоматическая загрузка файла UnityHubSetup.

7. После завершения загрузки файла, запустите его. Начнется процесс установки приложения Unity Hub. Unity Hub – это десктопное приложение, спроектированное для удобной работы пользователей. Из него происходит доступ к экосистеме игрового движка Unity, работа с менеджером проектов созданных в Unity, управление лицензиями и установка дополнительных компонентов.

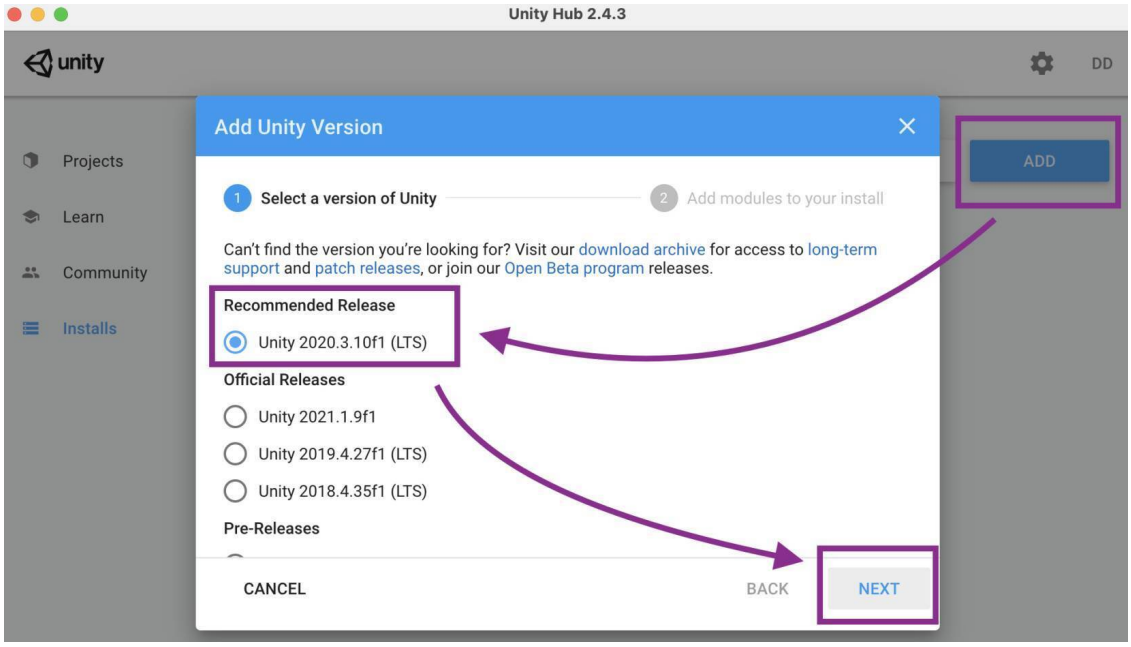
8. После завершения установки запустите Unity Hub. При первом запуске система предложит вам войти или создать свой аккаунт. Это можно сделать также при помощи популярных социальных сетей:



9. После того, как вы зашли в свой аккаунт Unity, откроется окно приложения Unity Hub. В центральной части приложения указаны проекты (Projects), с которыми вы работаете. Если вы используете Unity впервые, то это окно у вас должно быть пустым, однако очень скоро в нем начнут появляться созданные вами проекты, и Unity Hub будет выглядеть примерно следующим образом:



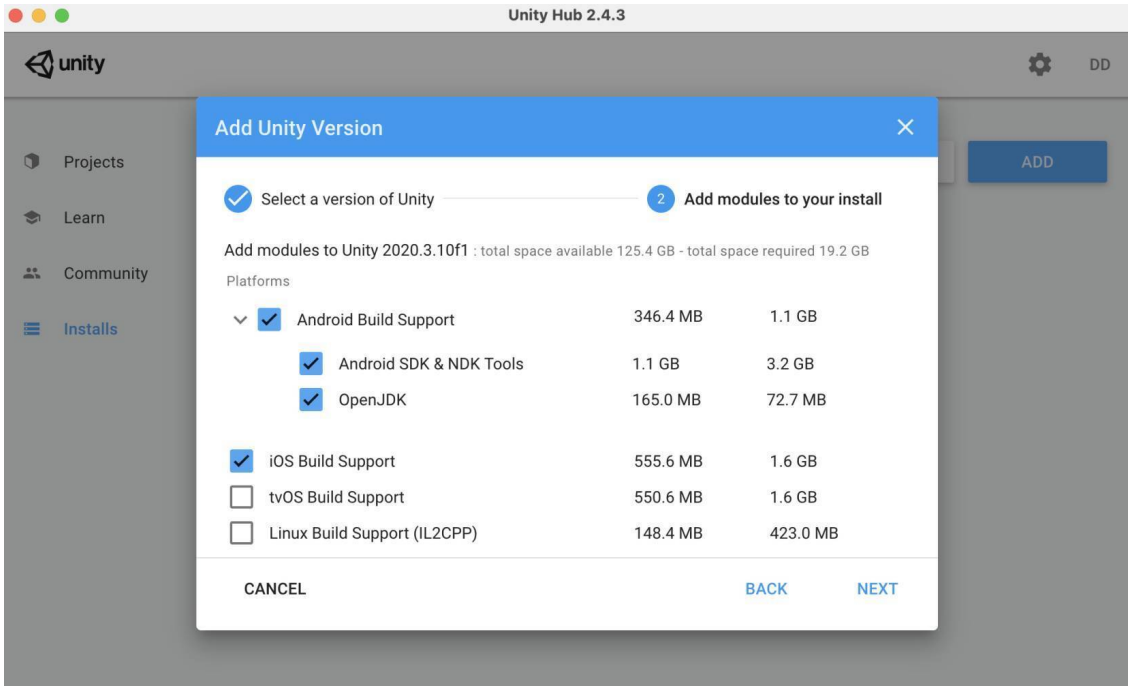
10. Оставаясь в Unity Hub нажмите кнопку Installs в левом меню и после того, как перейдете в новое окно, нажмите кнопку Add. После этого откроется окно выбора версий Unity для установки. Для начинающих пользователей лучше устанавливать рекомендованный релиз (Recommended Release) последней версии (как правило выбран по умолчанию). В книге используется версия Unity 2020.3.14f1:



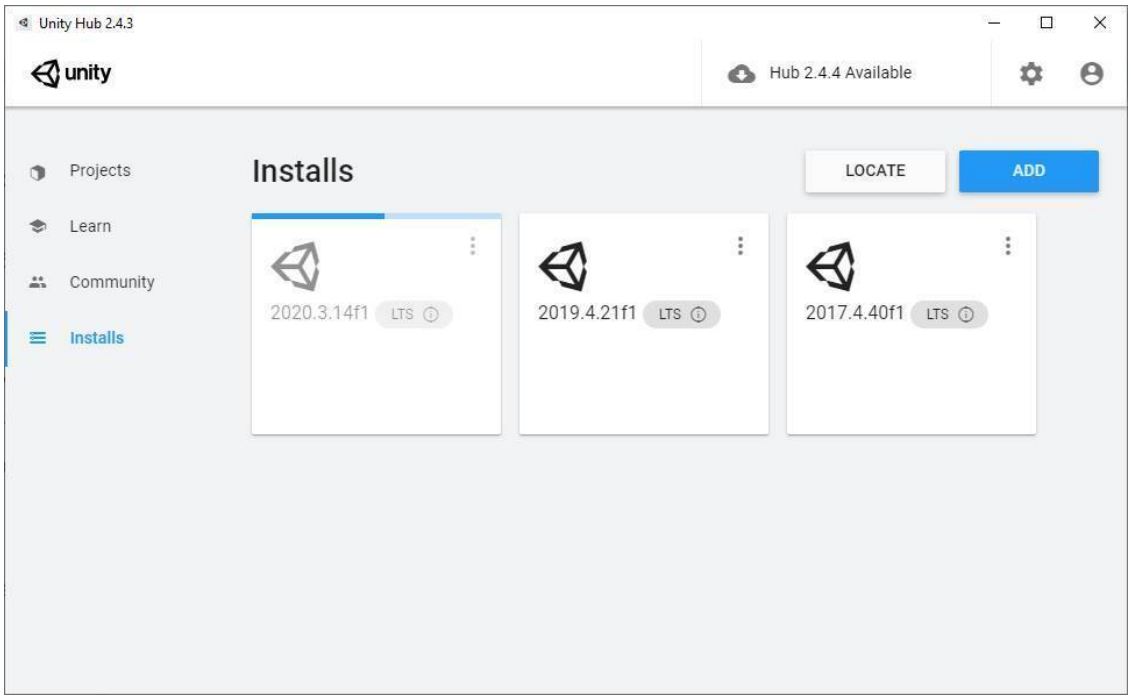
11. После выбора последней рекомендованной версии, нажмите кнопку Next.

12. В следующем окне нам предлагается выбрать компоненты редактора (модули). Модули также удобно разделены по платформам, на которые мы хотим создавать игры. Мы будем делать игру для выгрузки на Web-сервис, поэтому установите дополнительно модуль WebGL (поставьте флажок напротив модуля с таким названием). Кроме этого, Unity позволяет

делать игры под самые разные платформы. Например, если в дальнейшем вы захотите сделать игру под мобильное устройство, то все что вам потребуется – это установить модули Android Build Support и iOS Build Support:

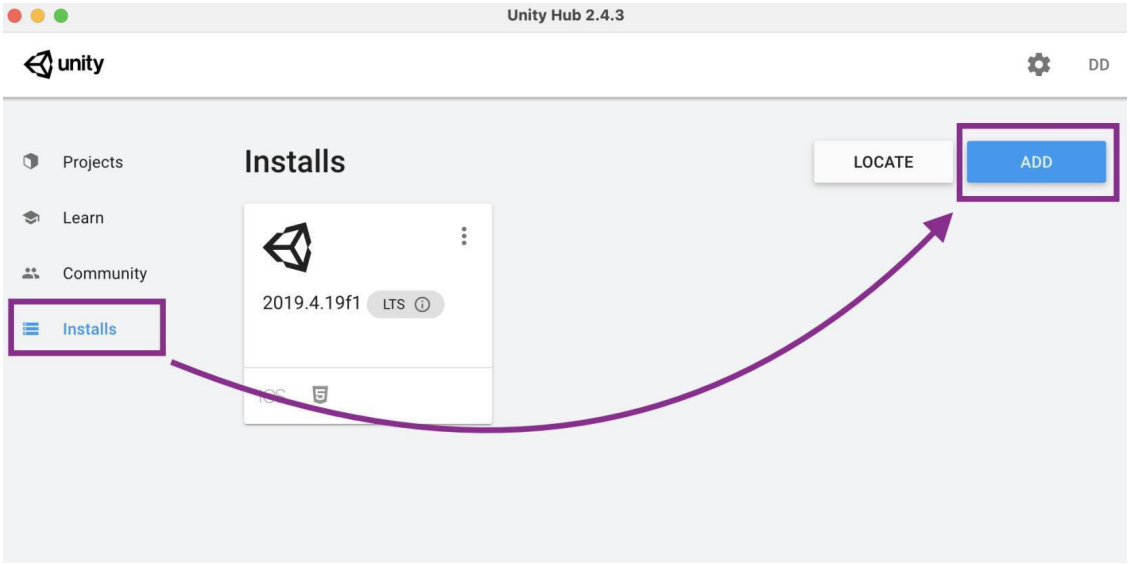


13. Жмем кнопку Next, на следующей странице принимаем соглашение, поставив галочку. Жмем Done и ожидаем скачивания и установки редактора. На компьютере можно держать несколько разных версий Unity (см. пример ниже):



14. На этом процесс установки закончен.

15. Если в дальнейшем вам понадобятся другие версии среды разработки Unity (например, вы найдете и захотите посмотреть готовые проекты, сделанные под более ранние версии среды разработки), – то вы всегда сможете открыть Unity Hub, перейти во вкладку Install и скачать недостающие версии Unity и модули:



Таким образом, Unity Hub является своего рода “точкой старта”, в которой происходит создание новых проектов (вкладка Projects), установка различных версий Unity (вкладка Installs) и т.д. Отмечу, что в Unity Hub можно держать несколько различных версий Unity, если у вас есть потребность работы с проектами, разработанными на ранних версиях.

По итогу пошагового выполнения всех указаний из данного раздела, у вас должна быть установлена среда разработки Unity. Также у вас должна быть создана учетная запись на сайте Unity.com. Не теряйте ее, так как через эту учетную запись происходит синхронизация проектов на вашем компьютере и на официальном сайте Unity. Кроме того, учетная запись нужна для работы с некоторыми полезными ресурсами, такими как Asset Store (подробнее об этом см. в разделе 2.2 Импорт игровых персонажей из Unity Asset Store).

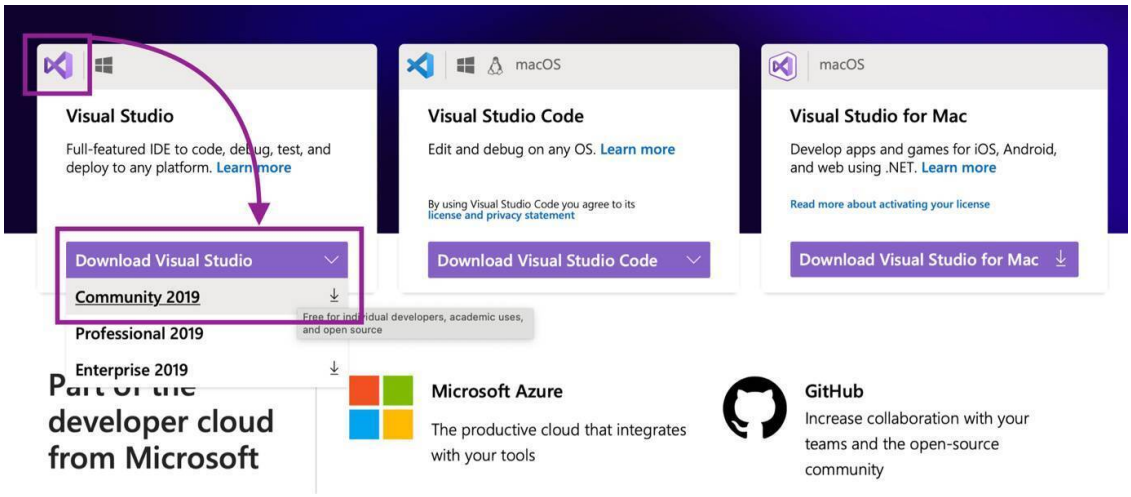
1.2 Установка редактора кода

Несмотря на то, что Unity является полноценной средой разработки компьютерных игр, вам понадобится отдельное приложение для работы с кодом (для написания скриптов на языке программирования). Написать код для Unity можно даже в обычном блокноте, главное, чтобы он был написан корректно на языке C#. Однако, удобнее использовать специализированные среды разработки. Мы будем использовать Microsoft Visual Studio, установить которую можно как стандартное приложение в системе Windows, либо с помощью Unity Hub.

Воспользуйтесь одним из двух предложенных способов установки среды для работы с кодом. Чтобы перейти к скачиванию и установке Microsoft Visual Studio, выполните следующие действия:

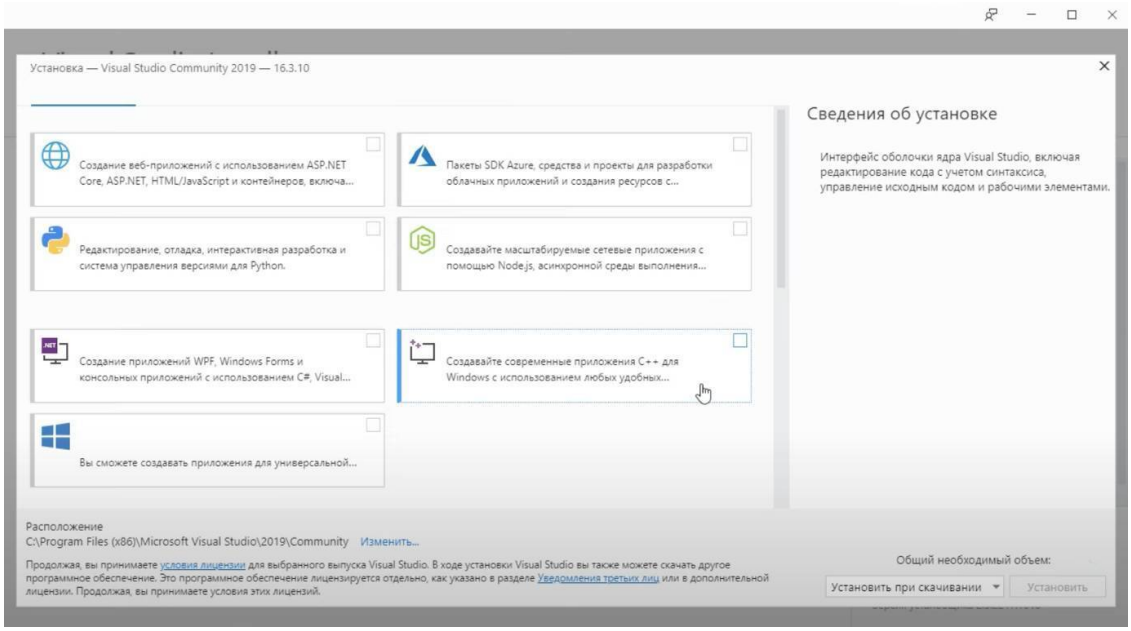
1. Перейдите на сайт https://visualstudio.microsoft.com/ru/

2. На сайте вы увидите несколько ссылок на скачивание среды разработки под различные операционные системы. Ниже будет показан пример установки для Windows. Выберите из выпадающего списка Download Visual Studio и выберите версию Community 2019:

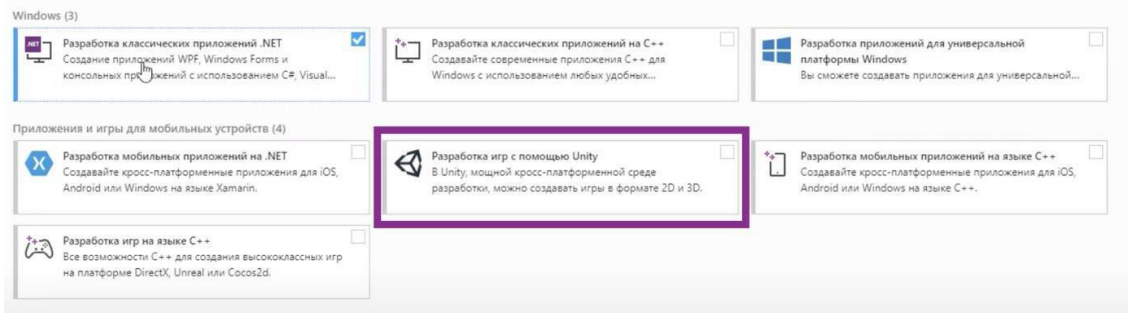


3. После этого автоматически начнется скачивание файла установки. После завершения скачивания запустите скачанный установочный файл.

4. Запустится Visual Studio Installer, который некоторое время будет скачивать необходимые файлы. Далее откроется окно с выбором компонентов, необходимых для установки:



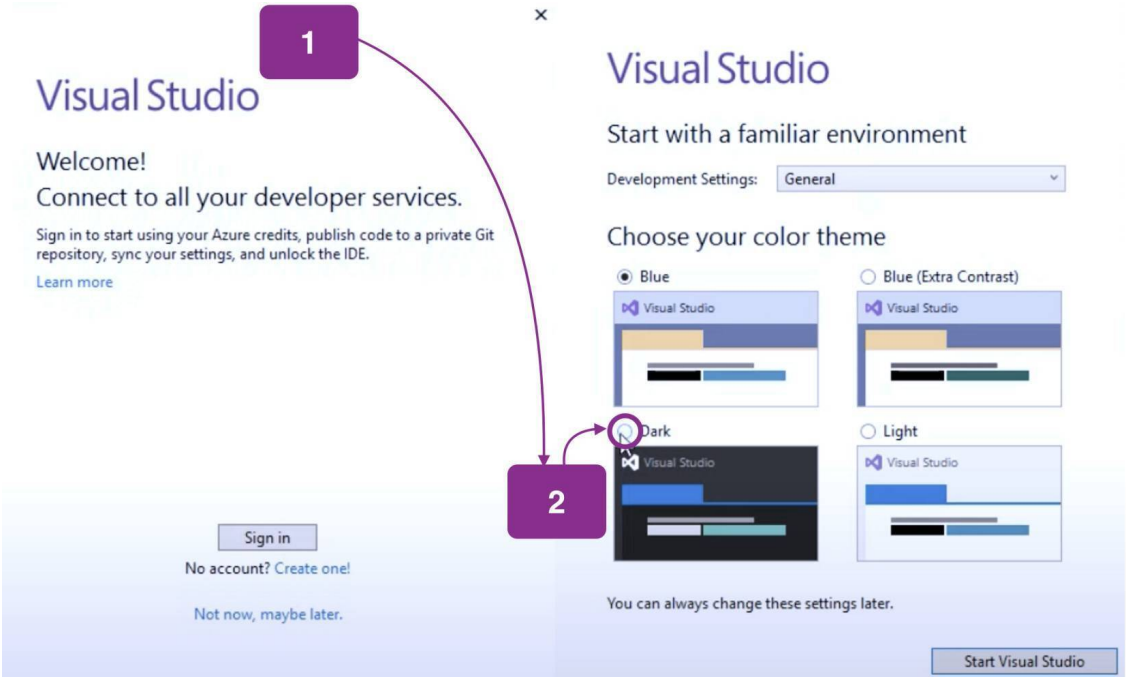
5. Среди множества компонентов найдите “Разработка игр с помощью Unity”:



6. После выбора необходимого компонента нажмите “Установить”. Скачивание и установка займет определенное время, которое зависит от производительности вашего компьютера и скорости Интернет-соединения.

7. После завершения установки, возможно, потребуется перезагрузка компьютера (в этом случае рекомендуется согласиться на столь заманчивое предложение и перезагрузиться).

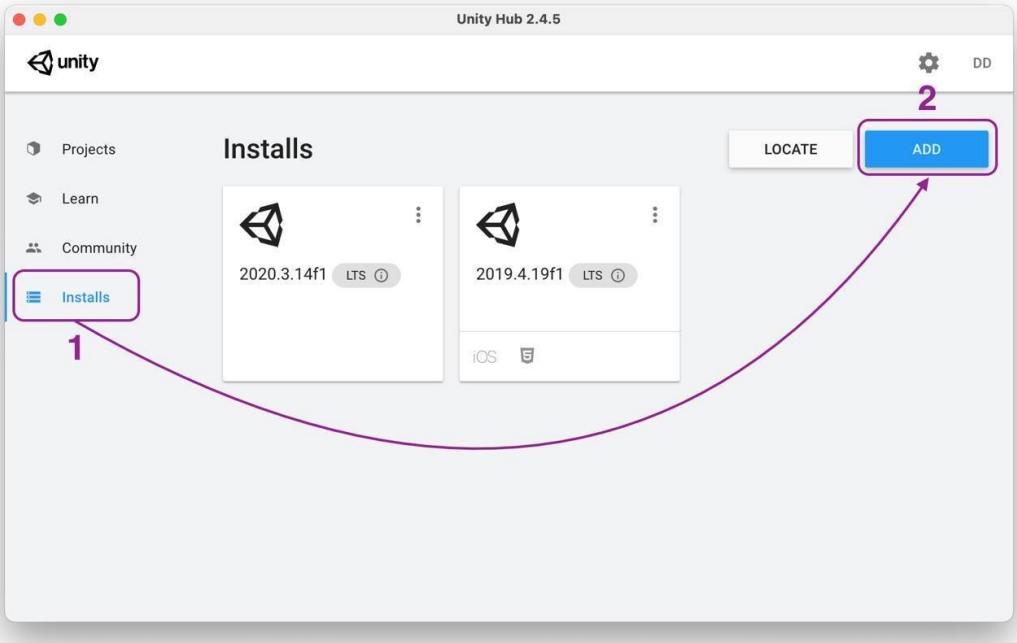
8. После перезагрузки запустите Visual Studio. При первом запуске среда разработки предложит вам выполнить вход под своей учетной записью (см. рисунок ниже слева) и выбрать вид темы (см. рисунок ниже справа):



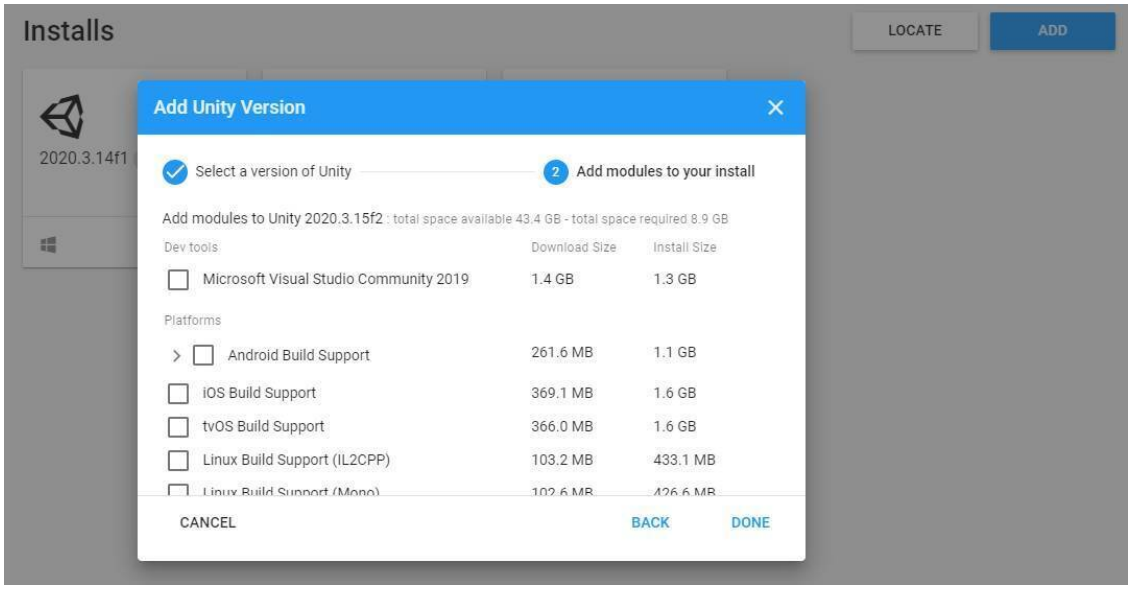
9. Указанные в предыдущем пункте действия необходимо сделать только один раз, далее среда разработки будет запускаться без каких-либо дополнительных всплывающих окон.

10. На этом установку Visual Studio можно считать законченной. Если среда разработки открыта, то вы можете ее просто закрыть. В дальнейшем мы будем открывать проекты напрямую из Unity и использовать среду разработки Visual Studio для работы с кодом.

11. Установить Visual Studio можно также из Unity Hub. Чтобы это сделать выберите вкладку Installs – Add (можете пропустить этот и следующие два шага, если среда разработки уже была установлена):



12. В появившемся окне сначала предлагается выбрать нужную версию Unity (окно Add Unity Version), а если нажать кнопку Next, то при переходе на вторую страницу (Add modules to your install) можно выбрать установку Visual Studio:

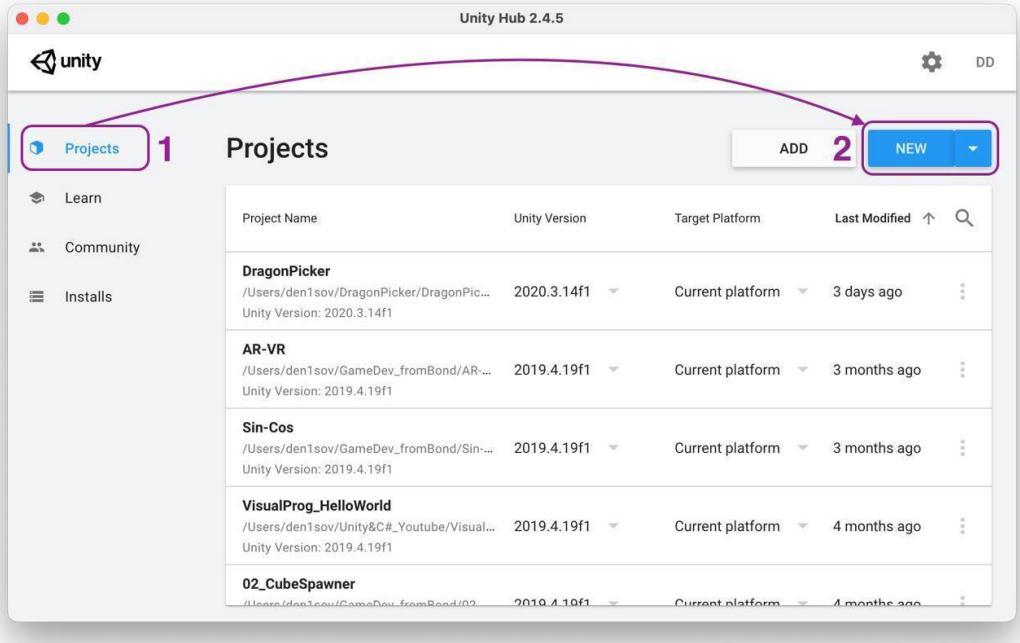


13. Поставьте галочку напротив Microsoft Visual Studio Community 2019. Эта опция доступна как для iOS, так и для Windows. Обратите внимание, если у вас уже установлена Visual Studio, она не будет отображаться в качестве опции с дополнительной установкой. Нажмите Done, начнется скачивание и установка в автоматическом режиме среды разработки Visual Studio. Это программное обеспечение вы будете использовать для редактирования кода.

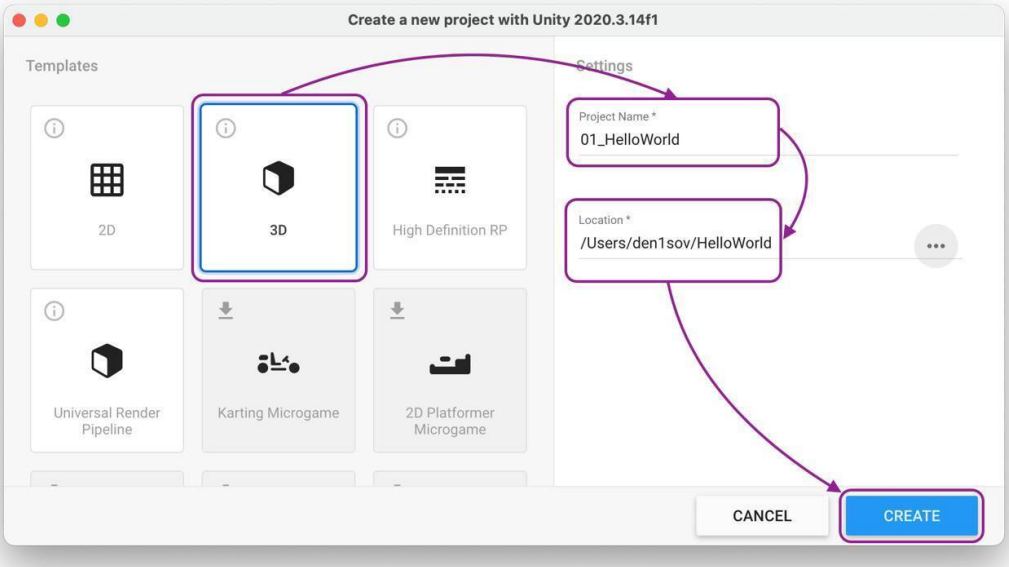
1.3 Программа “Hello World” и принципы работы в Unity

Проверить корректность работы всех установленных программных пакетов можно, написав простейшую программу. По традиции принято создавать программу, которая выводит сообщение Hello World. В нашем примере мы не просто выведем сообщение, но и научимся взаимодействовать с объектами в среде Unity.

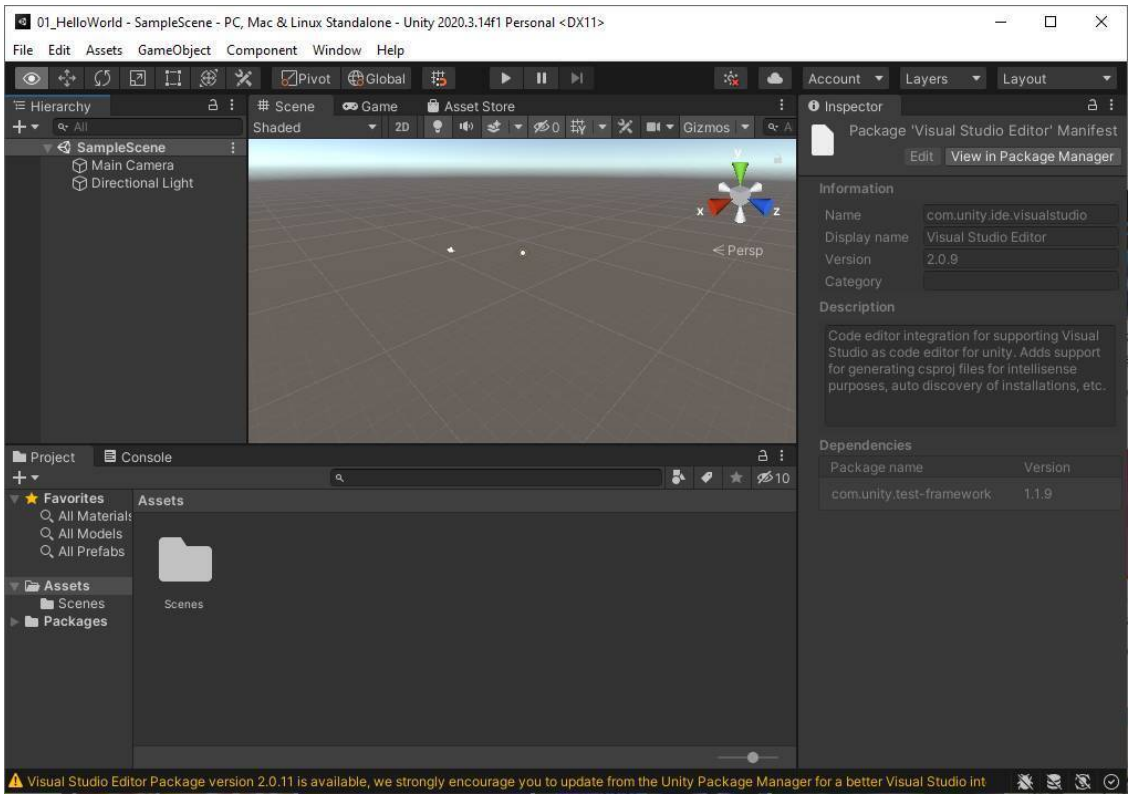
1. Чтобы создать первый проект на Unity, откройте Unity Hub и перейдите во вкладку Project. Нажмите New чтобы перейти в окно создания нового проекта:



2. В появившемся новом окне выберите тип проекта – 3D, дайте имя новому проекту, например 01_HelloWorld. Проверьте путь к папке, в которой будет создан проект (здесь скорее важно, чтобы вы осознанно указали папку для проекта и не потеряли его в дальнейшем). После этого нажмите Create:

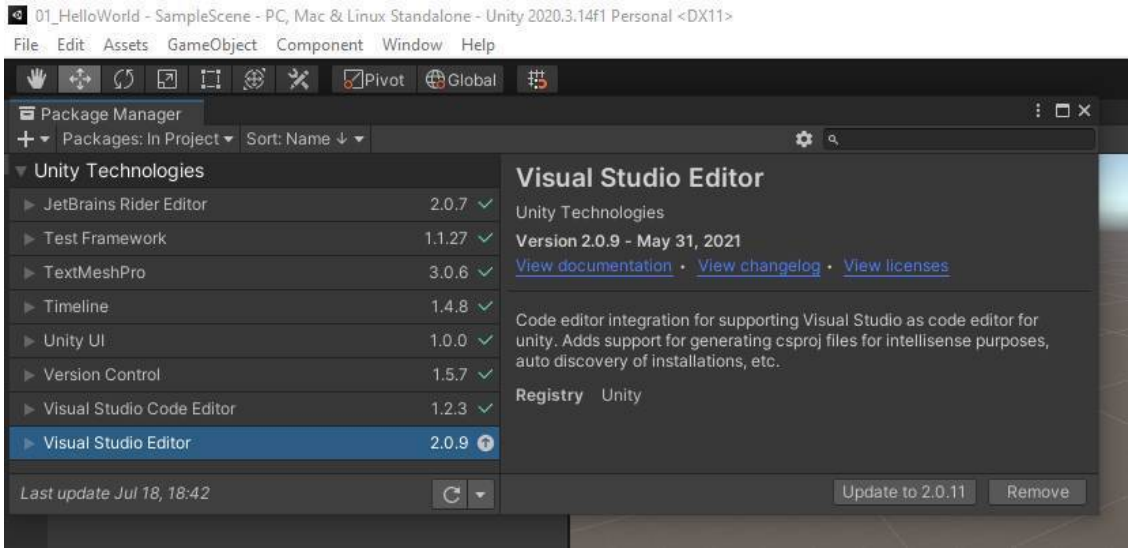


3. Проект будет создан и открыт в новом окне Unity. На рисунке ниже показано, как выглядит запущенная среда разработки Unity, в данный момент вам может показаться, что она содержит довольно большое количество разнообразных и непонятных окон, но в дальнейшем мы разберемся в том, как они устроены и за что отвечают отдельные элементы среды разработки:

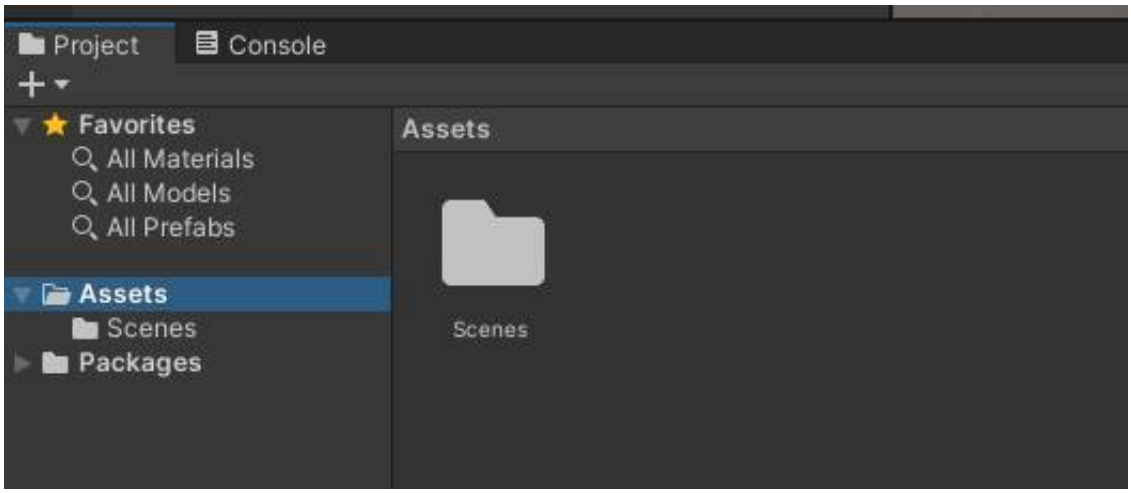


4. Возможно при первом запуске Unity появится сообщение о необходимости обновить встроенный Visual Studio Editor до последней версии (сообщение с желтым восклицатель-

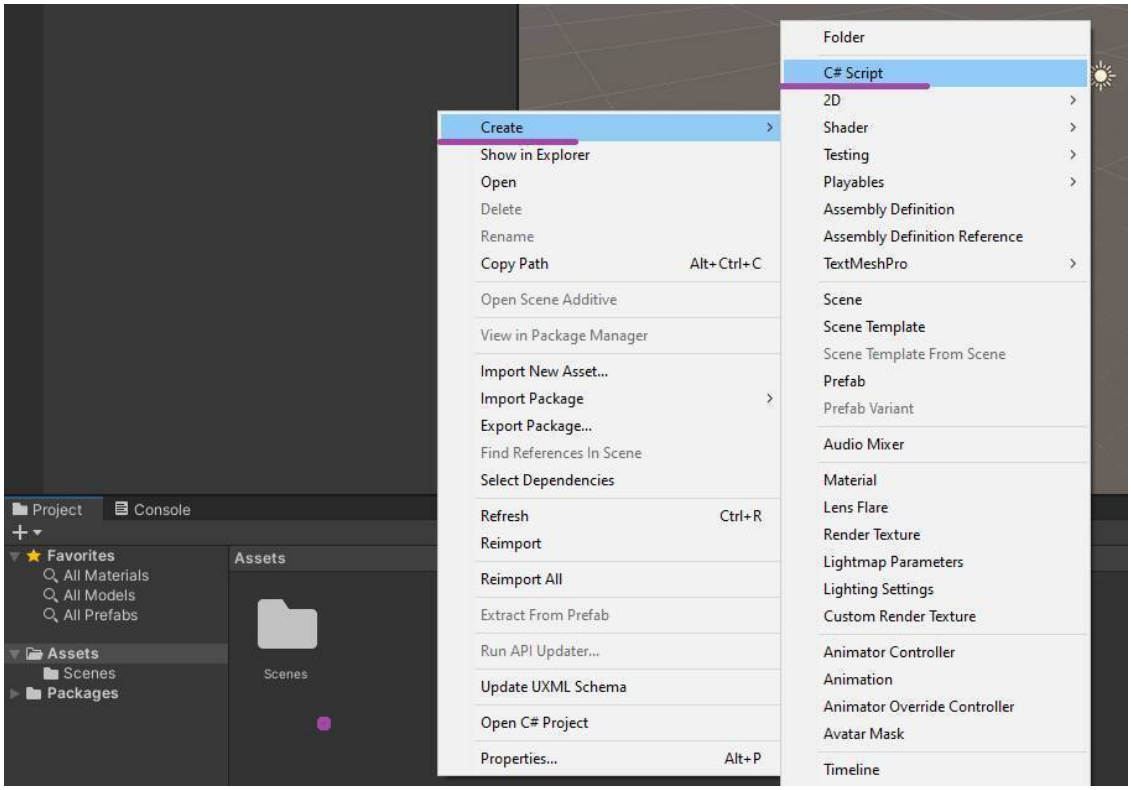
ным знаком внизу среды разработки). В этом случае перейдите во вкладку Window – Package Manager, выберите Visual Studio Editor и нажмите Update:



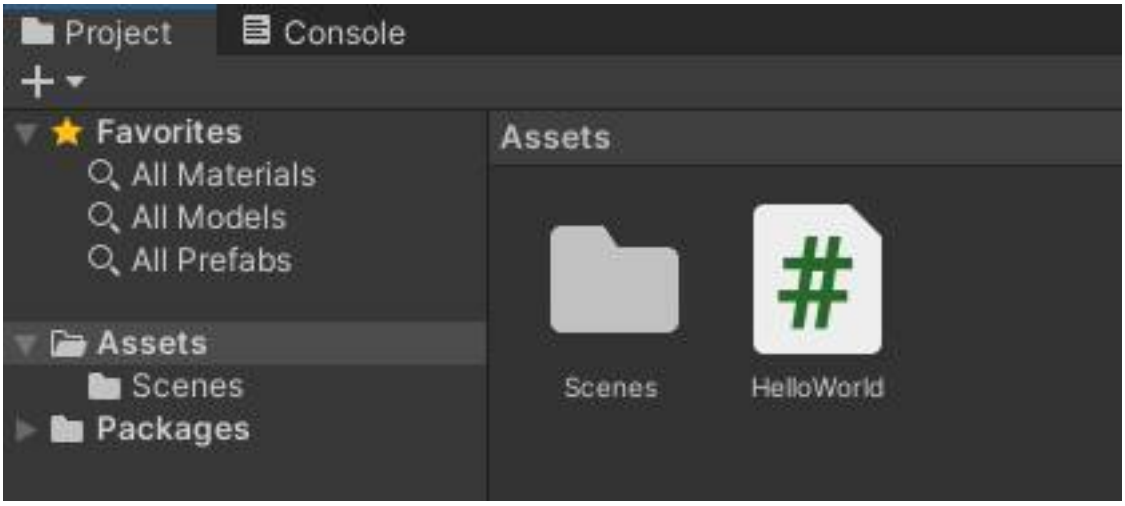
5. Создадим новый C# Script-файл с простой командой, которая выводит сообщение “Hello World”. Для этого на панели Project перейдите в папку Assets, в данный момент в ней находится только одна папка с названием Scenes:



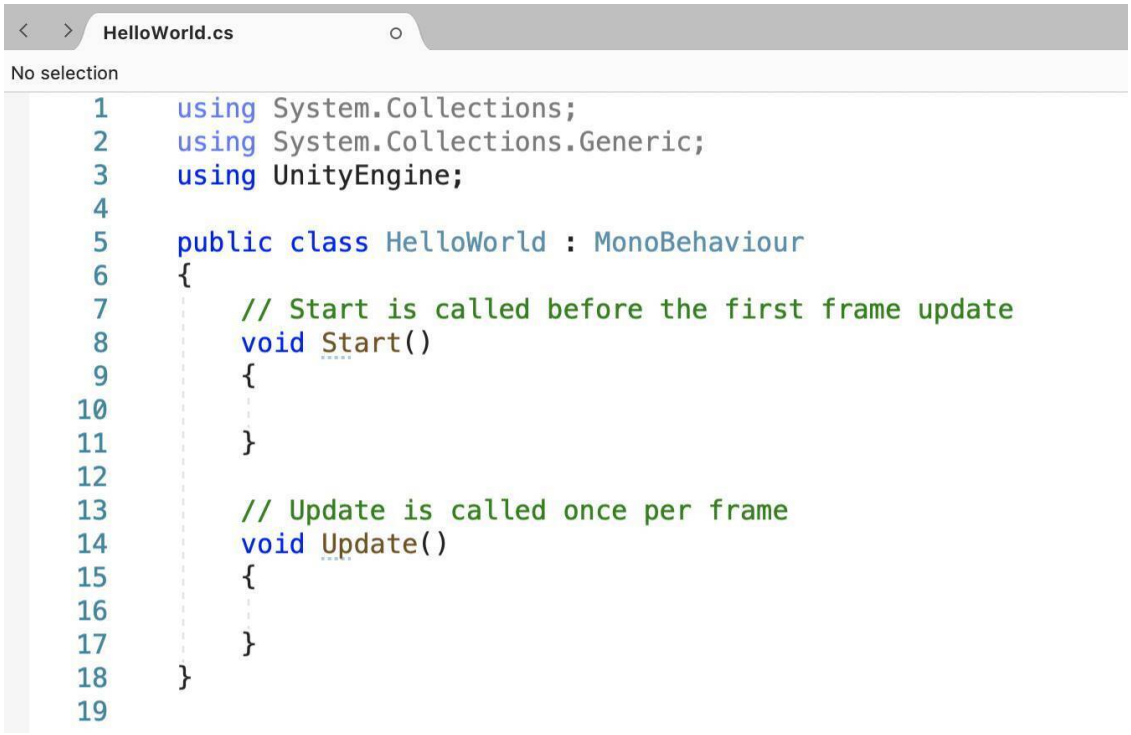
6. Кликните правой кнопкой мыши внутри папки Assets и выберите из контекстного меню Create – C# Script, как показано ниже:



7. Назовите созданный скрипт-файл HelloWorld. Содержимое папки Assets после этого должно выглядеть так, как показано на рисунке ниже:



8. Откройте файл HelloWorld.cs, кликнув по нему дважды. Файл автоматически откроется в Visual Studio Community 2019. Содержимое файла и вид среды разработки показаны на рисунке ниже:



9. В дальнейшем мы будем приводить программный код и в виде скриншотов, и в виде листинга (текстом). Его будет удобнее читать, а в случае использования электронной версии издания – копировать и вставлять части кода в свой проект. В дальнейшем мы разберемся с каждой строкой приведенного выше программного кода. Пока лишь обратите внимание на то, что внутри кода содержится два метода: void Start() и void Update().

– void Start() – это метод, который запускается при старте игры в Unity. Это значит, что команды, написанные внутри фигурных скобок этого метода, отработают один раз при запуске сцены в Unity.

– void Update () – это метод, который запускается каждый кадр на сцене. Другими словами, в метод Update() следует писать тот функционал, который требует регулярного обновления в процессе игры.

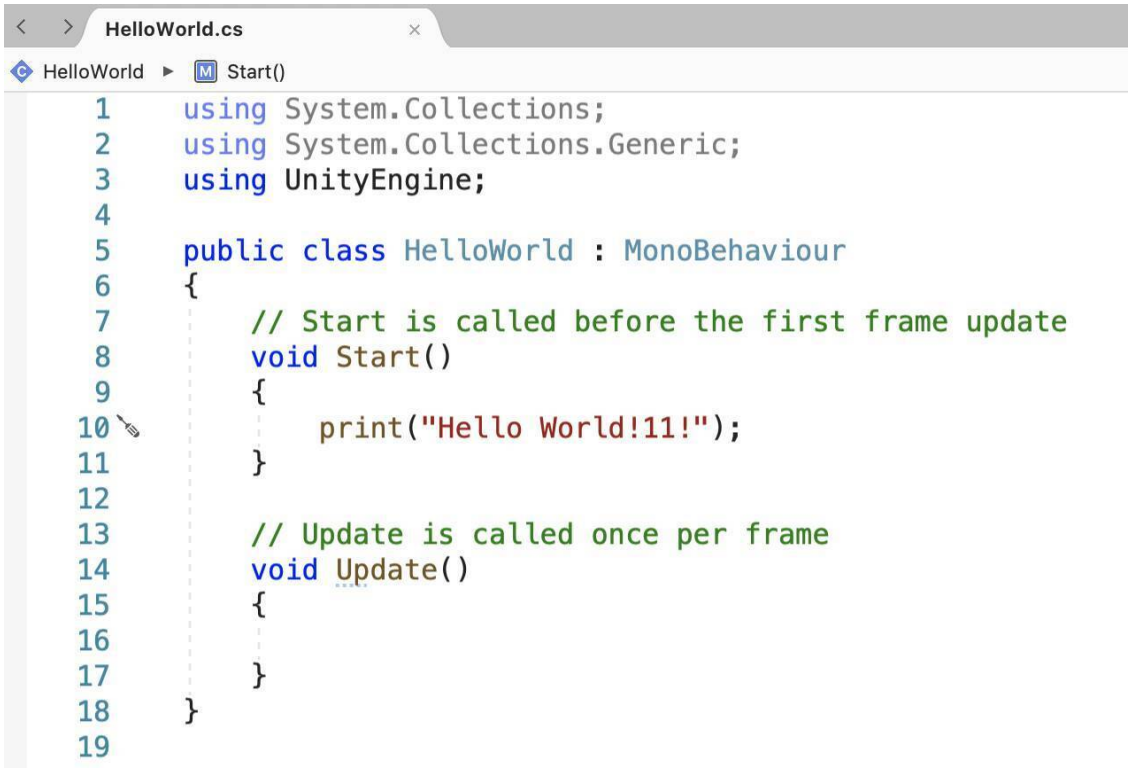
10. Добавьте строку кода, которая будет выводить сообщение «*Hello World!11!*». Для этого внутрь фигурных скобок метода void Start(), как показано в листинге ниже (листинг приводится целиком), нужно написать команду print:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HelloWorld : MonoBehaviour
{
    void Start()
    {
        print("Hello World!11!");
    }
    void Update()
    {
    }
```

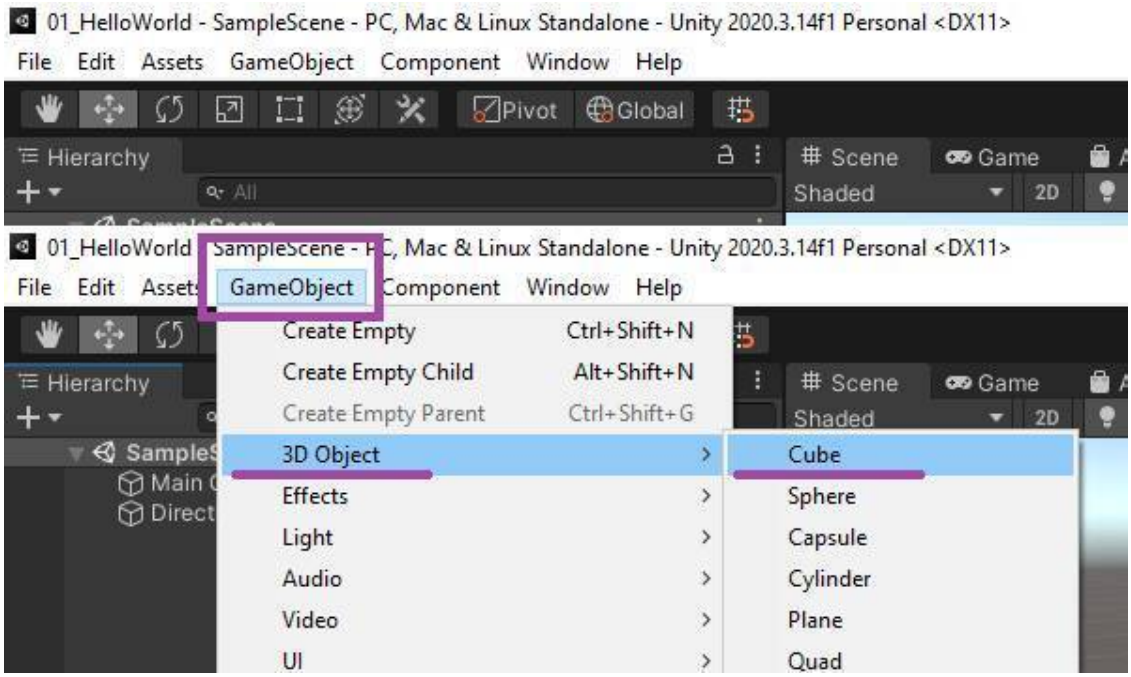
```
}
// End Code
```

Листинг дублируется ниже в виде скриншота из MS Visual Studio.

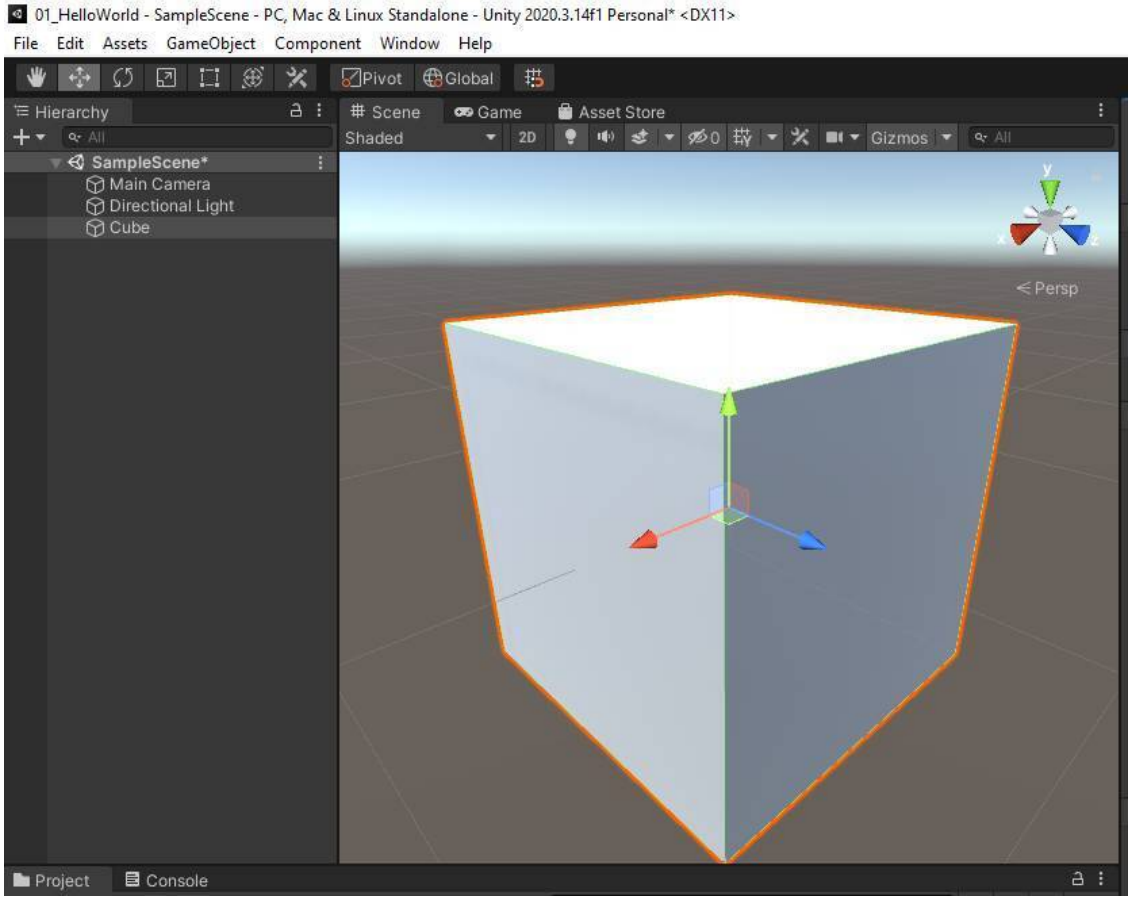


11. Скрипт-файл с названием HelloWorld.cs написан. Однако, чтобы он начал работать, нам следует его подключить к одному из игровых объектов внутри сцены Unity. Давайте создадим такой объект, например, простейший куб.

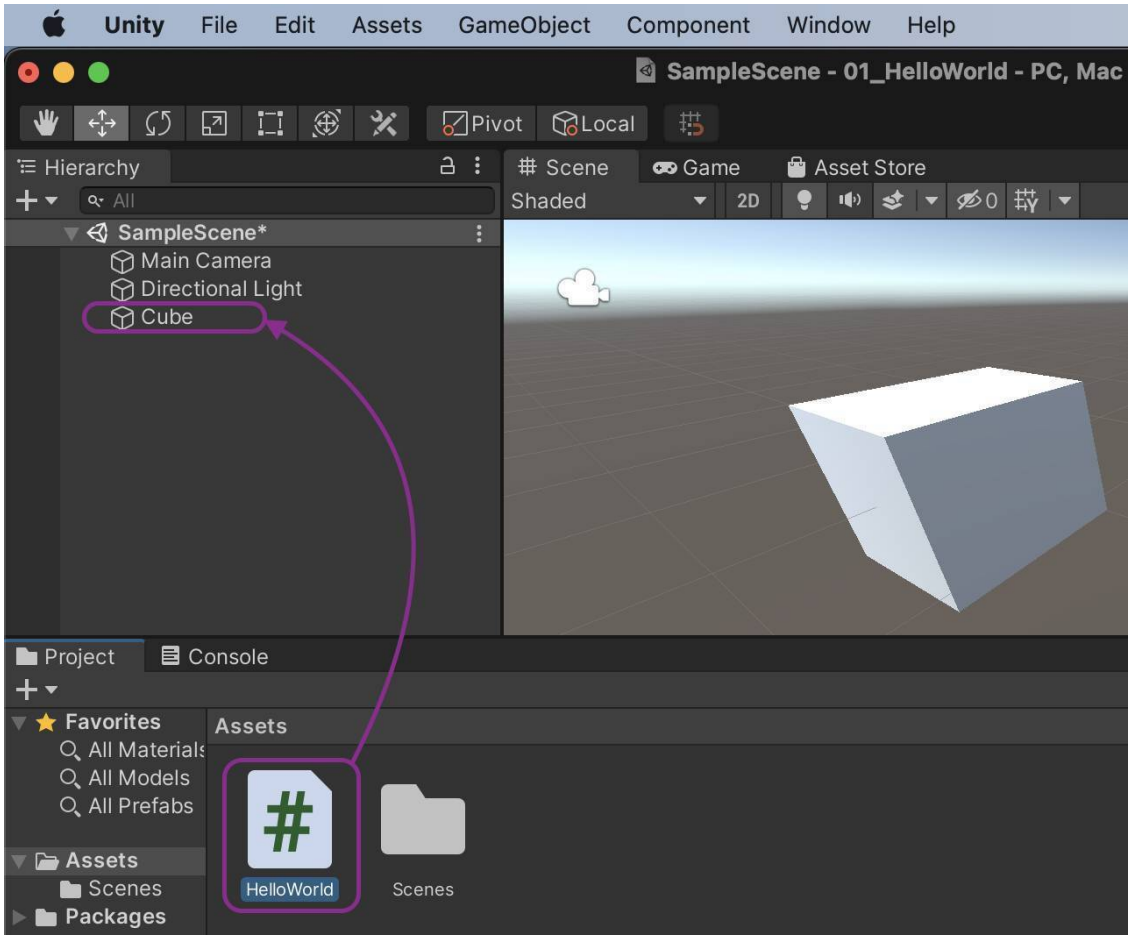
12. Чтобы создать игровой объект “Куб”, на верхней панели быстрого доступа в среде Unity выберите GameObject – 3D Object – Cube:



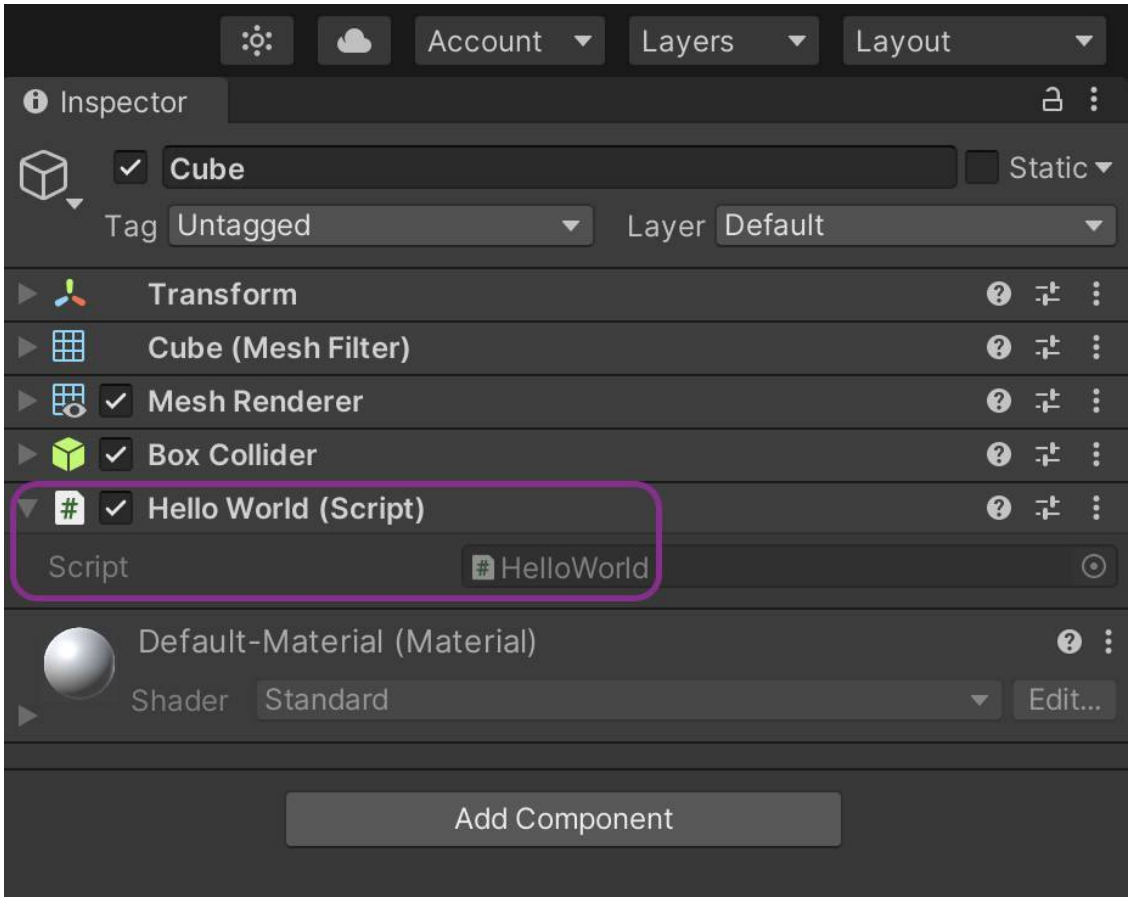
13. Таким образом, на сцене появится новый игровой объект:



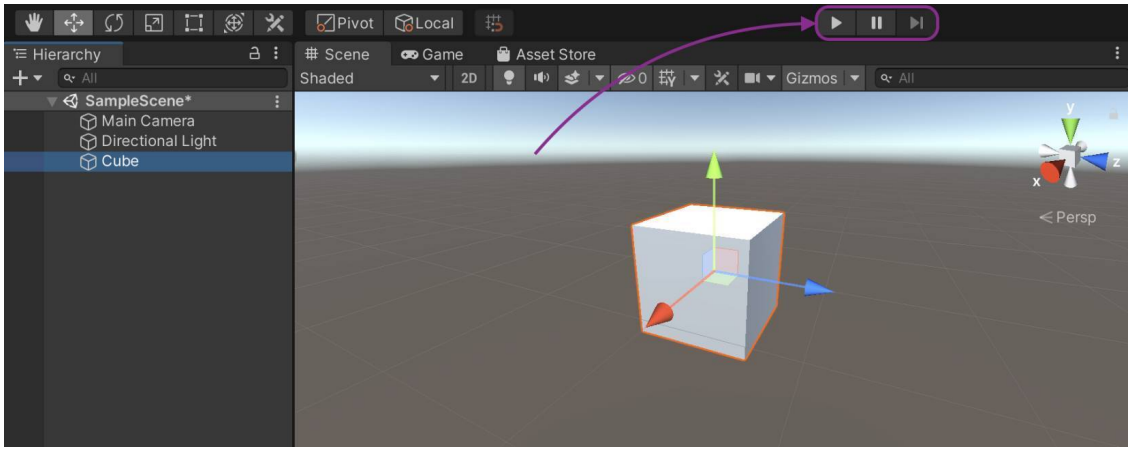
14. Чтобы подключить скрипт HelloWorld.cs к объекту Cube, можно просто перетащить (зажав левую кнопку мыши) скрипт-файл на куб.



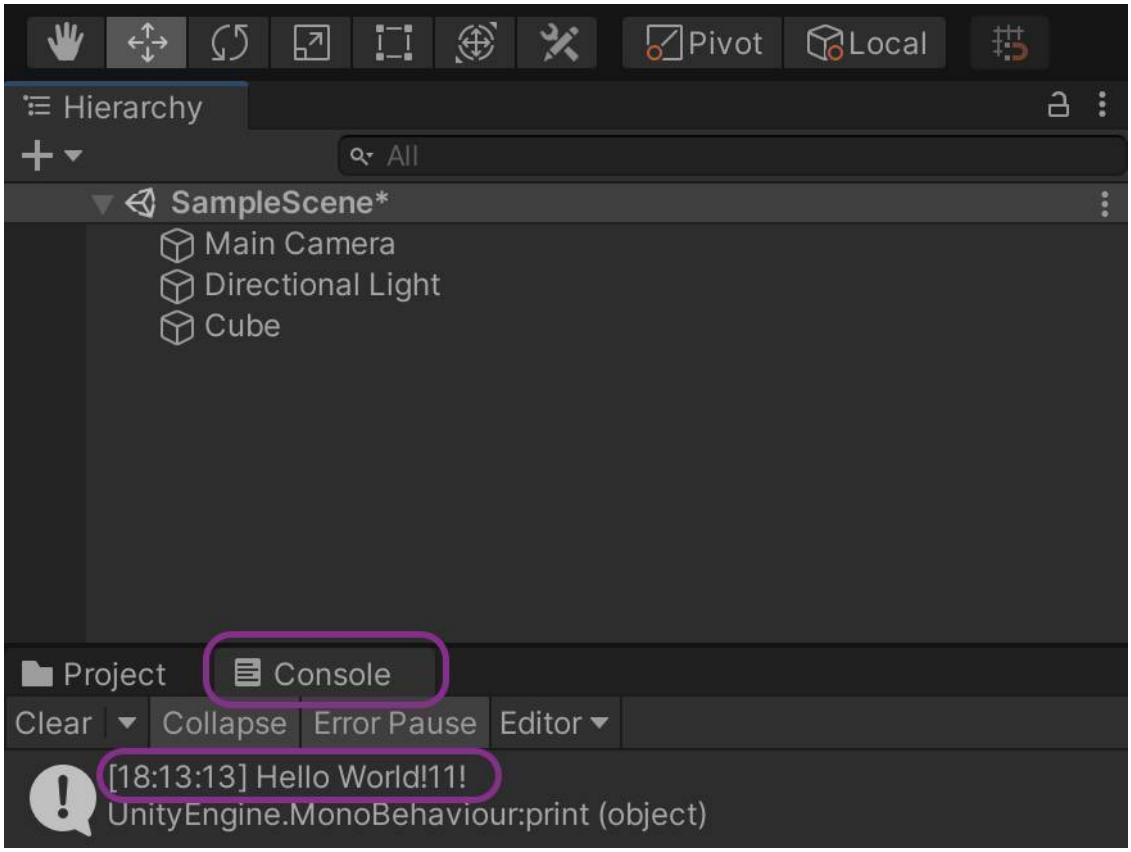
15. Теперь, если выделить объект Cube, кликнув по нему левой кнопкой мыши, то можно увидеть, что в правой части среды разработки (окно Inspector) к кубу подключился файл HelloWorld.cs (Script-файл):



16. Теперь, можно запустить сцену и проверить ее работу. Для этого нужно нажать кнопку Run в верхней центральной части среды разработки.



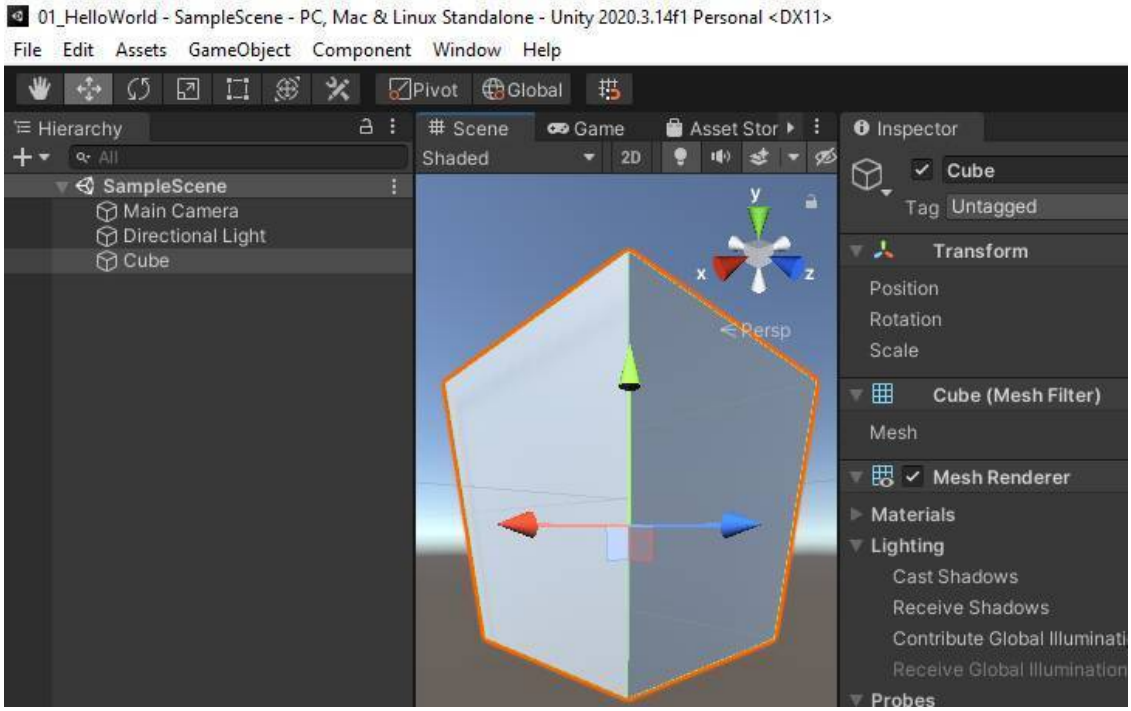
17. После этого сцена запустилась. На ней статично висит куб и кажется, что ничего не происходит, но если перейти в окно Console (в нижней части среды разработки), то можно заметить, что при старте было отправлено сообщение в чат:



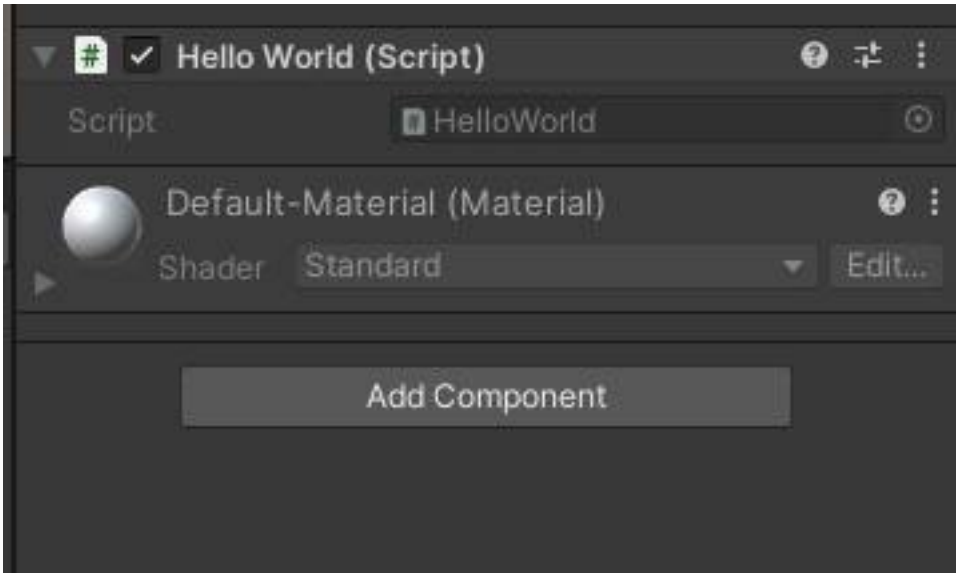
Вместо функции print, можно использовать функцию Debug.Log(), которая является частью движка Unity. Отличие функции Debug.Log() от функции print() заключается в том, что print() не позволяет увидеть какую-либо информацию, после сборки проект. То есть print() выводит информацию только в консоль среды разработки Unity, тогда как функция Debug.Log() выводит сообщение в специальный файл в папке проекта при запуске готовой игры, содержимое которого потом можно просмотреть. По сути обе эти функции делают одно и тоже, но рекомендуют использовать именно Debug.Log(). В качестве эксперимента вы можете заменить функцию print(“Hello World”) в листинге выше на функцию Debug.Log(“Hello World”).

Обычным сообщением в окне консоли сложно удивить, особенно если речь идет о разработке игры. Поэтому давайте добавим еще немного функций для наглядности. Следует отметить, что многие моменты, связанные с разработкой игры, на себя берет Unity без необходимости написания какого-либо функционала. Иногда даже очень сложный функционал в игре можно реализовать просто настройками внутри среды разработки Unity. Продемонстрируем это на примере ниже.

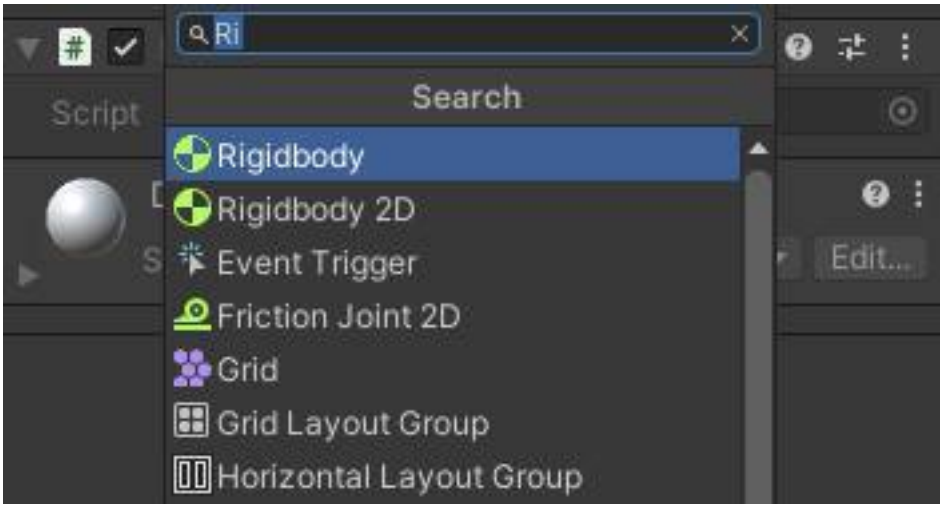
18. Сделаем так, чтобы созданный 3D объект Cube при запуске сцены падал вниз. Для этого выделите объект Cube (клик левой кнопкой мыши в окне Hierarchy), после этого в правой части среды разработки станет активно окно Inspector:

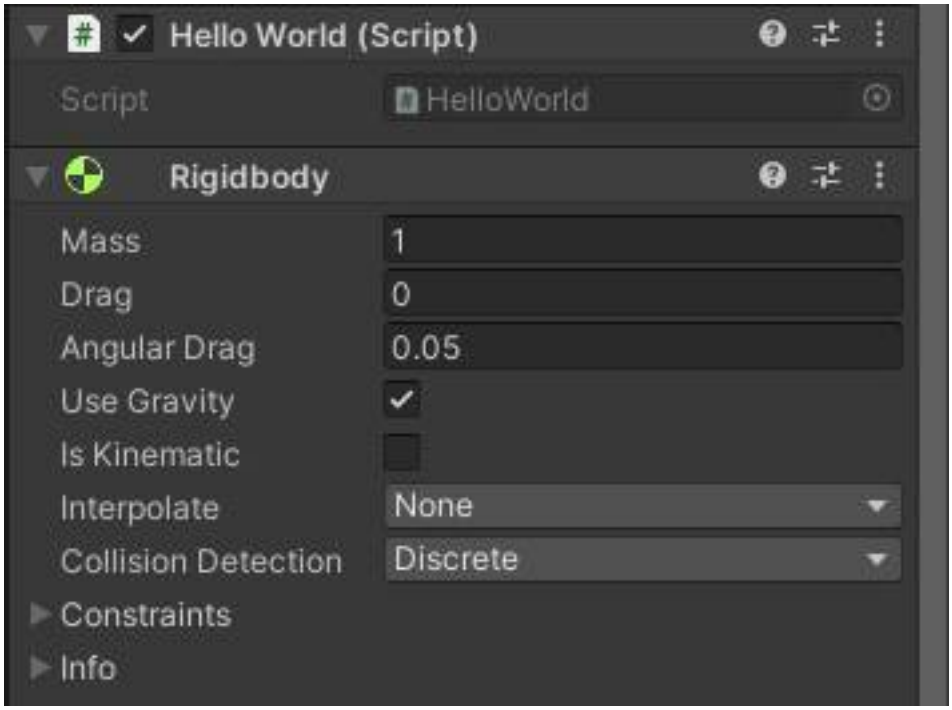


19. Содержание окна Inspector зависит от типа выбранного объекта. В нем содержатся свойства объекта, его параметры, подключаемые Script-файлы и т. д. Нажмите в нижней части окна Inspector кнопку Add Component:



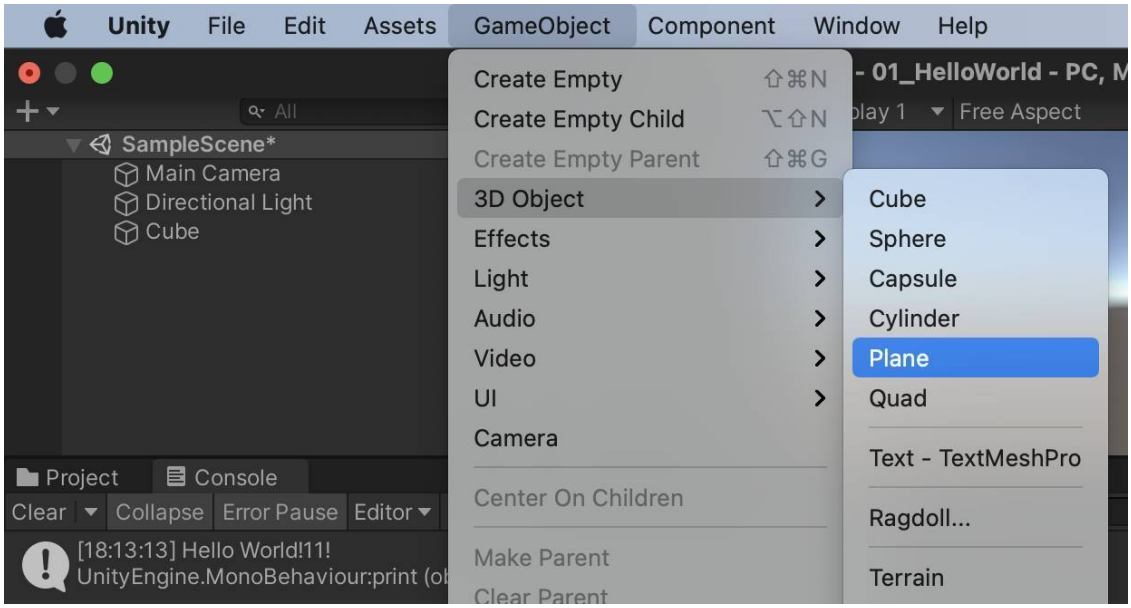
20. После этого появится список с перечнем компонентов, которые могут быть подключены к выбранному объекту Cube. Найдите с помощью поиска компонент Rigidbody и кликните по нему левой кнопкой мыши так, чтобы он добавился в окно Inspector.



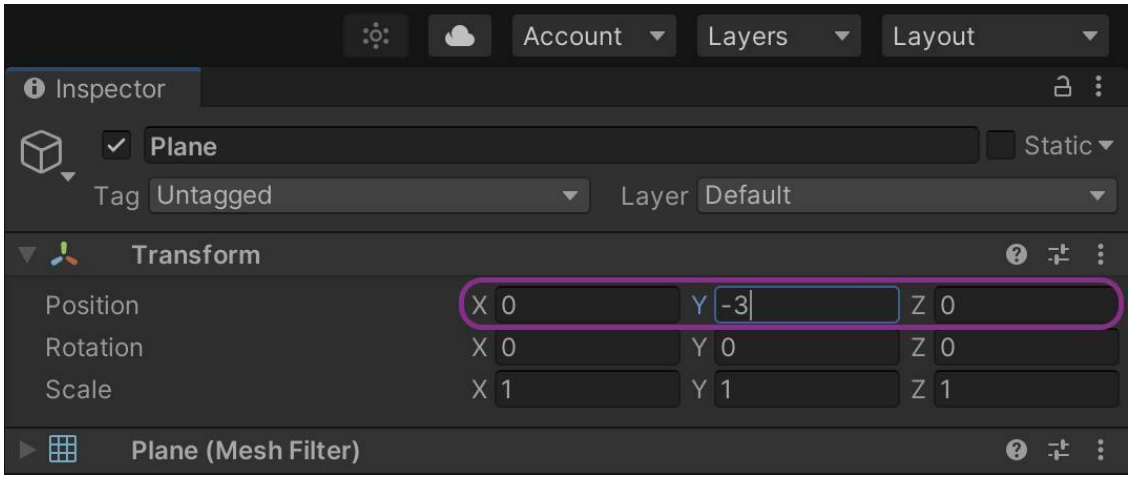


21. Rigidbody – это механическое свойство объекта, определенное в базовом движке Unity. Другими словами, если назначить этот компонент объекту, то он начнет вести себя в соответствии с механикой абсолютно твердого тела. Запустите сцену еще раз (нажмите Run) и убедитесь, что теперь объект Cube начинает падать вниз.

22. Создадим еще один объект – плоскость (Plane), которая будет ограничивать падение куба за пределы начальной сцены. Для этого выполните действия, которые уже выполнялись при создании объекта Cube, – в верхней части меню выберите GameObject – 3D Object – Plane:



23. После создания плоскости переместим ее немного ниже уровня объекта Cube. Для этого выделите объект Plane в окне Hierarchy (клик левой кнопкой мыши) и в окне Inspector в верхней части установите значения напротив свойства Transform – Position [X:0, Y: -3, Z:0]:



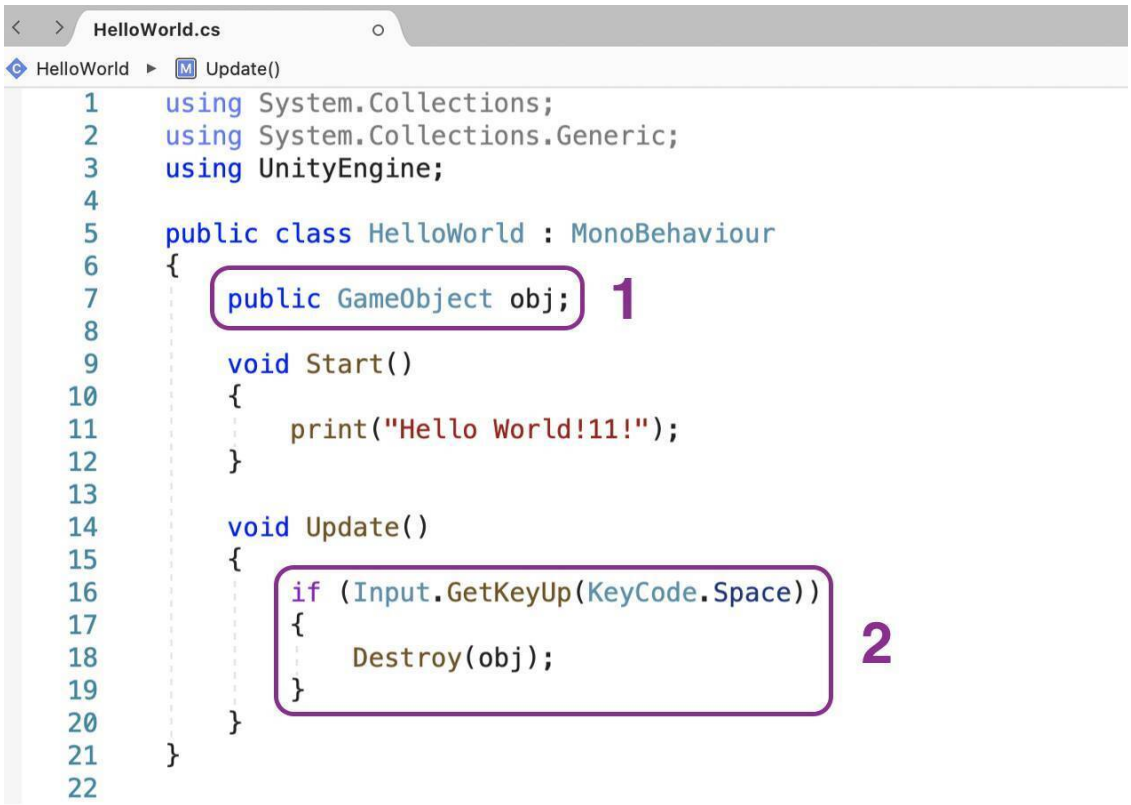
24. Запустите сцену еще раз. Теперь объект куб (Cube) падает на плоскость (Plane) при старте сцены.

25. Теперь добавим немного интерактивности. Откройте скрипт-файл, который мы создали ранее с именем HelloWorld.cs и напишите туда небольшой функционал, который будет уничтожать объект Cube при нажатии клавиши пробел. В программном коде ниже показано содержимое файла HelloWorld.cs, а жирным шрифтом в комментариях показаны новые строки кода, которые нужно ввести дополнительно:

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HelloWorld : MonoBehaviour
}
```

```
    public GameObject obj;
    void Start()
    {
        print("Hello World!11!");
    }
    void Update()
    {
        if (Input.GetKeyUp(KeyCode.Space))
        {
                Destroy(obj);
        }
    }
}
// End Code
```

Листинг дублируется ниже в виде скриншота из MS Visual Studio.



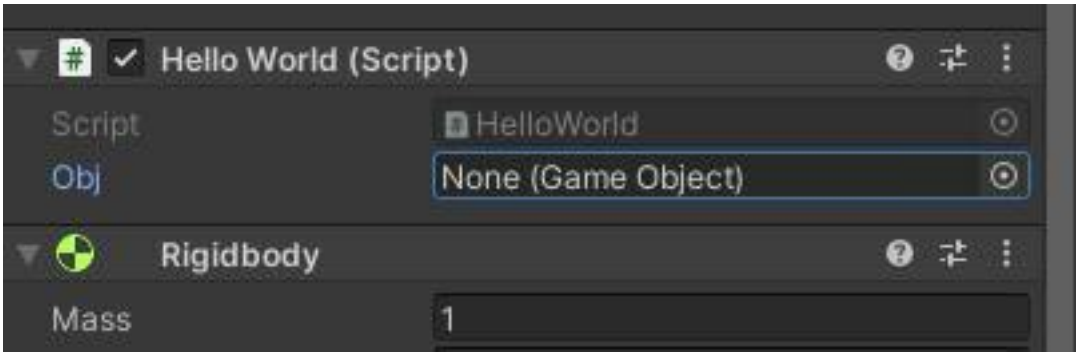
В листинг были добавлены следующие строки кода:

– создается переменная класса GameObject с именем obj, в которую можно будет поместить объект (как это сделать будет показано далее). Класс GameObject – это встроенный класс Unity, в дальнейшем мы познакомимся с множеством встроенных классов и их методов;

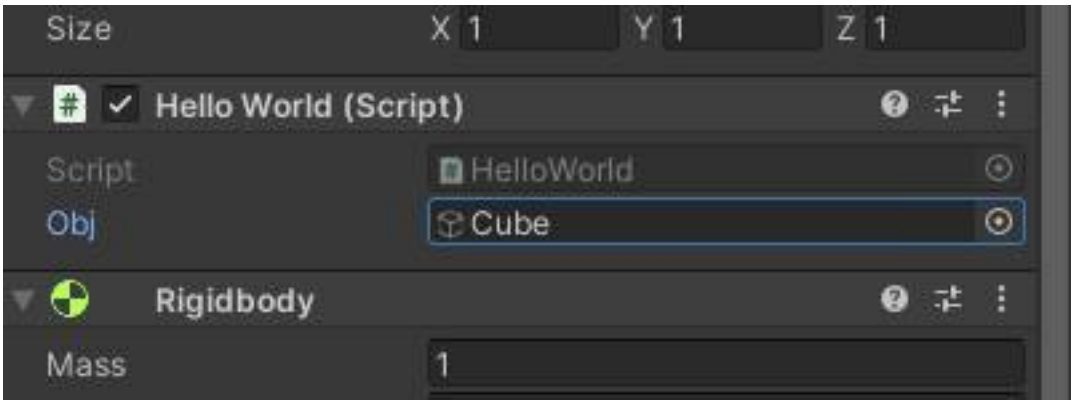
– создается условие if, которое уничтожает объект (Destroy(obj)) при нажатии клавиши Space. При этом используется метод Input.GetKeyUp, который срабатывает, после того как игрок отпустил клавишу. Также есть похожий метод Input.GeyKeyDown, который срабатывает сразу при нажатии клавиши. В качестве эксперимента можете изменить метод на Inputy.GetKeyDown.

26. Вернитесь в среду разработки Unity и снова выберите объект Cube. обратите внимание, что теперь в окне Inspector внутри скрипт-файла HelloWorld.cs появилась новая перемен-

ная Obj. Стоит отметить, что если бы мы создали приватную (private) переменную объекта, то в инспекторе мы бы ее не увидели (приватную переменную можно увидеть, если сверху строки public GameObject obj добавить [Serialize.Field]):



27. Новая переменная появилась благодаря тем самым строкам кода, которые мы добавили выше. Теперь, если кликнуть на значок “мишени” рядом с надписью None (Game Object) то можно выбрать любой из существующих на сцене объектов, который будет удаляться при нажатии клавиши пробел. Например, выберите объект Cube:



28. Теперь запустите сцену и проверьте, что она работает следующим образом:
– В окно Console выводится сообщение *“Hello World!11!”*;
– Куб (Cube) начинает падать;
– Куб падает на плоскость Plane и останавливается;
– При нажатии на клавишу пробел объект Cube удаляется.

29. В качестве эксперимента замените объект Cube на плоскость Plane внутри скрипта Hello World (Script) в окне Inspector:

30. Проверьте, как теперь отрабатывает сцена после запуска. Что происходит при нажатии на клавиши “пробел”? Теперь, при нажатии клавиши, со сцены должна удаляться плоскость Plane.

Выводы

После завершения всех пунктов рекомендуется вернуться в начало раздела и еще раз внимательно просмотреть всю последовательность действий. Попробуйте самостоятельно внести модификации в некоторые пункты на свой выбор. Так вы сможете более детально разобраться в устройстве взаимосвязей между объектами, скрипт-файлами и некоторыми элементами интерфейса Unity. Ниже приведен некоторый список возможных изменений в проекте Unity, который вы можете внести, опираясь на те инструкции, которые были получены в этом разделе:

– Сделайте так, чтобы в Console выводилось сообщение “Goodbye World”.

– Добавьте на сцену больше объектов произвольной формы, измените их размер, положение и ориентацию.

– Модифицируйте скрипт-файл таким образом, чтобы разные объекты могли быть удалены со сцены при нажатии разных клавиш на клавиатуре.

– Перенесите строку кода *print("Hello World!11!");* из фигурных скобок метода Start() в фигурные скобки метод Update(), и проверьте работу сцены. Что изменилось в выводе в командной строке Console?

Часть 2. Создание игрового прототипа

Введение

Создание геймплея (игрового процесса) называют игровым дизайном (англ. Game Design). Обычно перед началом разработки следует определиться с основными требованиями, предъявляемыми к игре. Если говорить более обобщенно, то в списке требований, с которыми следует определиться при начале разработки, можно выделить следующие:

– платформа, под которую осуществляется разработка (Windows, WebGL, Android / iOS),
– экран: разрешение и ориентация,
– длительность игровой сессии,
– система управления,
– однопользовательская или многопользовательская игра,
система монетизации приложения.

Цель нашего практикума по разработке заключается в том, чтобы сделать игру и опубликовать ее на одном из онлайн-ресурсов (подробнее об этом см. Часть 7). Поэтому, наша игра должна будет удовлетворять следующим требованиям:

– платформа: WebGL,
– экран: ландшафтная ориентация, разрешение не менее 1280x1024,
– длительность игровой сессии: 3–5 минут,
– система управления: легкое управление при помощи мыши и клавиатуры,
– однопользовательская игра,
– система монетизации не предусмотрена.

В игре “Dragon Picker” центральным объектом в игре будет являться дракон, который периодически роняет драконье яйцо. Игровой процесс будет заключаться в том, чтобы ловить летящие вниз объекты. В игру могут быть добавлены разные виды объектов, одни из которых могут добавлять очки в игре, жизни, либо отнимать их. Для нас важно будет добавить один вид объектов (например, драконье яйцо), а разные виды других объектов вы сможете добавить самостоятельно по аналогии.

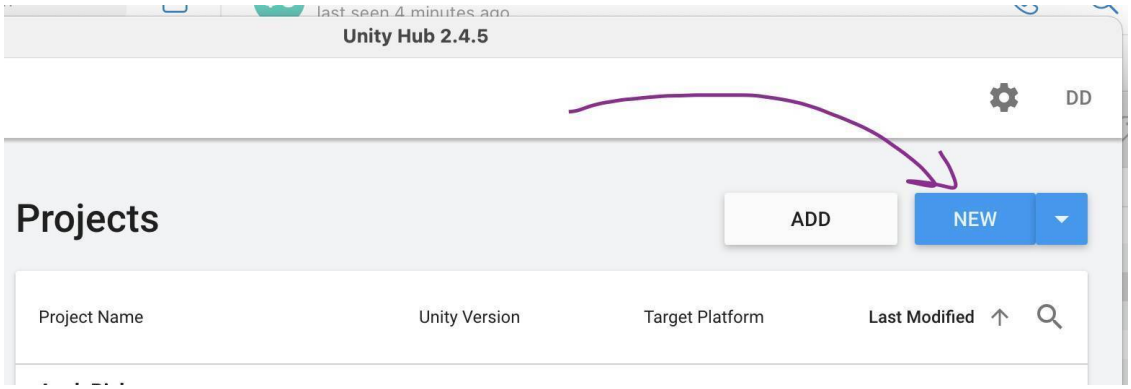
Часто для понимания концепции игры, да и просто для того чтобы передать атмосферу, полезно сделать наброски в виде скетчей. Выполнить наброски можно как с применением профессиональных средств рисования/прототипирования, так и просто на листе бумаги. В любом случае рано или поздно придется показать игру всему миру (например, выложив на google play, app store или simmer.io) и довольно полезно иметь в этом случае контент для визуального оформления страницы с игрой. Довольно много изображений и видеоматериалов можно найти в открытом доступе для бесплатного коммерческого использования. Их вполне может быть достаточно для того, чтобы вдохновиться, или использовать на первом этапе чтобы передать концепцию создаваемой игры. Ниже изображен пример скетча для игры “Dragon Picker” с сайта pixabay.com:

Важно отметить, что игровой дизайн имеет малое отношение к графическому дизайну, картинка приведенная выше призвана лишь передать общую атмосферу создаваемой игры. При создании игры именно текстуры и визуальное оформление позволяют описать общую концепцию игрового процесса. С игрой будет интереснее взаимодействовать, когда она представлена не только в виде кубиков и блоков. Для оформления игры используются текстуры, а взаимодействие с объектами описывается такими свойствами (компонентами в Unity), как Collider и RigidBody. Подробнее об этих компонентах вы узнаете в следующих главах.

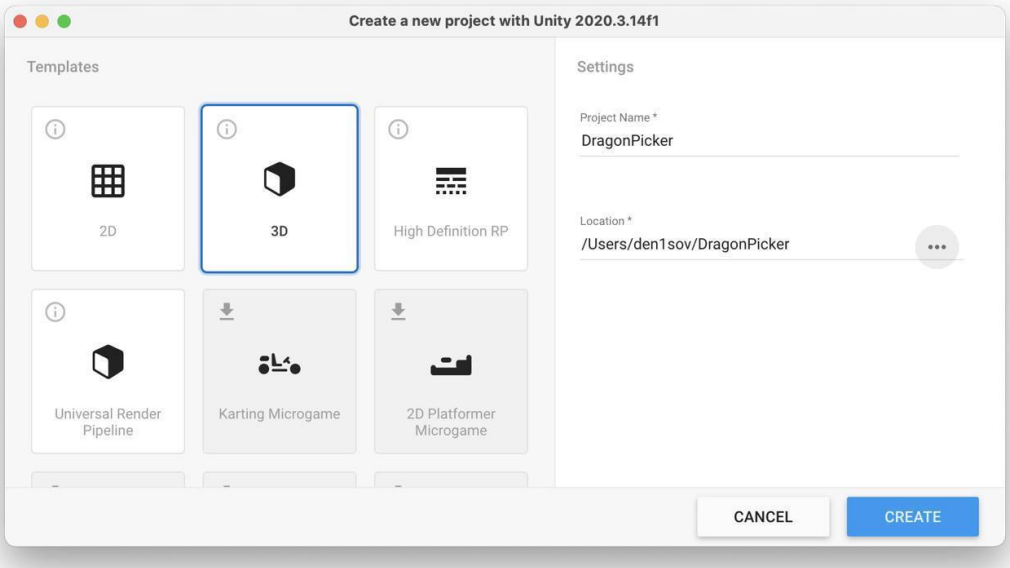
2.1 Создание проекта и первой сцены

Последовательность создания проекта и сцены внутри среды разработки практически не отличается для различных операционных систем.

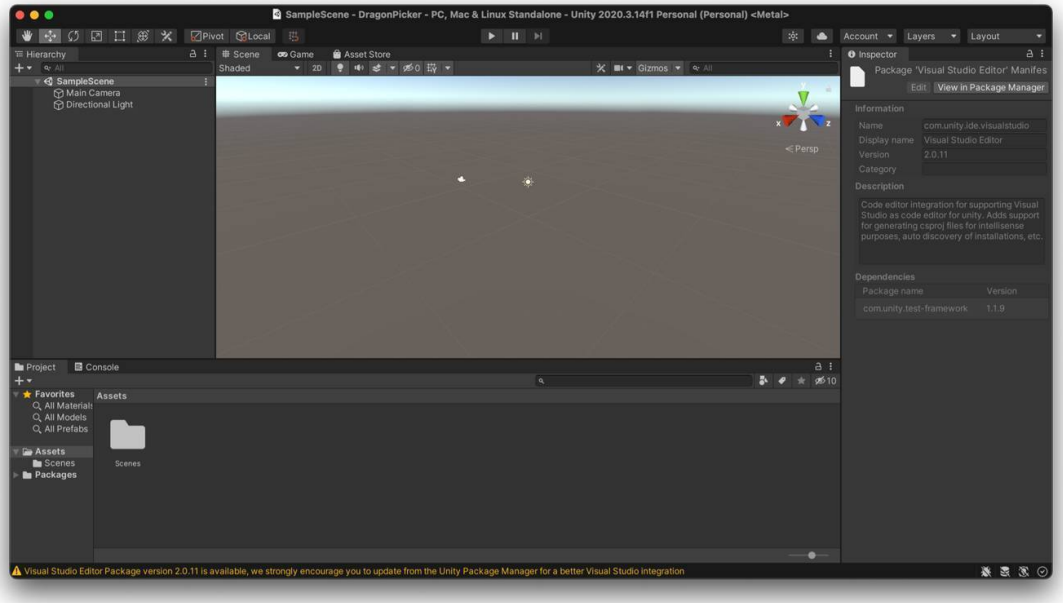
1. Запустите Unity Hub, который был скачан и установлен в предыдущей главе;
2. Создайте новый проект, для этого внутри Unity Hub нажмите New;



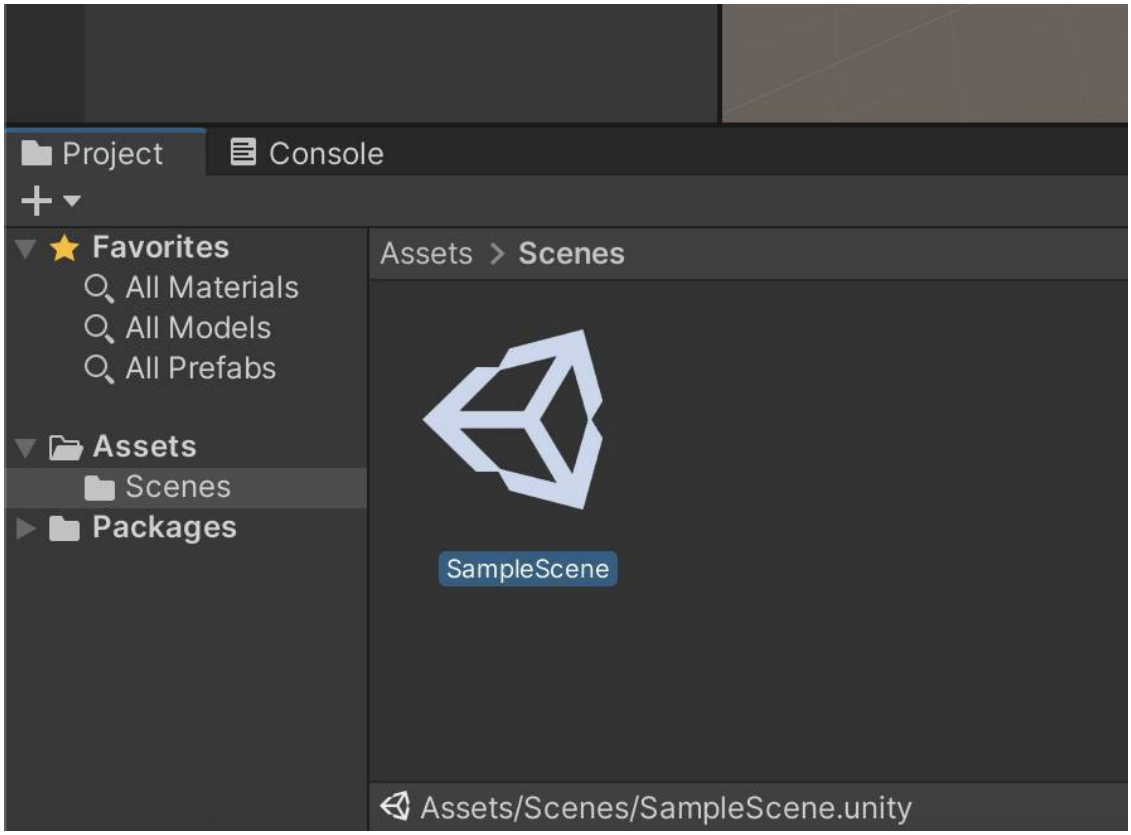
3. Выберите вид проекта Templates – 3D, в поле Project Name дайте имя проекту, например DragonPicker, в поле Location укажите путь к папке с проектом. Напомним, что проект может находиться в любом месте на вашем компьютере, (главное, чтобы вы смогли его найти позже):



4. Нажмите кнопку Create. После этого откроется среда разработки Unity. Может потребоваться некоторое время, это зависит от производительности вашего компьютера. Вид среды разработки после старта изображен на рисунке ниже:

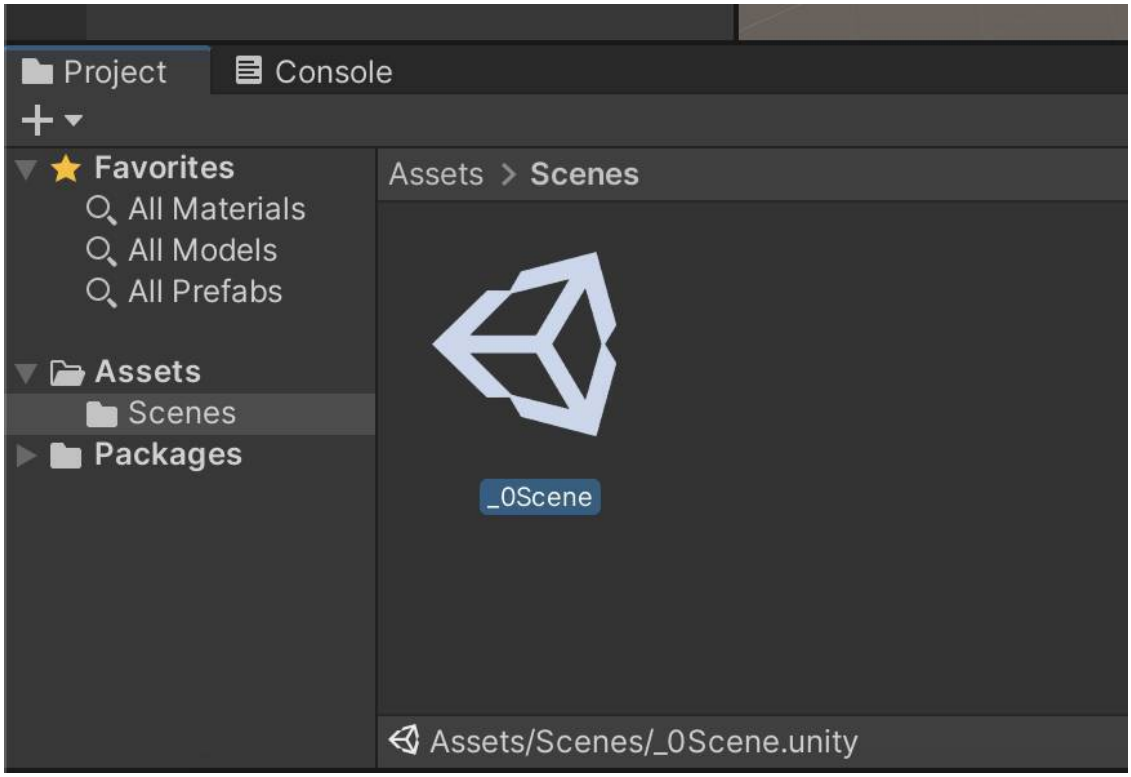


5. В левой нижней части среды разработки внутри панели Project находится структура проекта. Внутри папки Assets хранятся исходные файлы проекта. В данный момент там существует папке Scenes, в которой находится главная сцена с именем SampleScene:



6. Внутри сцен могут существовать отдельные объекты и персонажи, с которыми происходят различные действия во время игры. Сохраните сцену, а также давайте переименуем главную сцену. Это можно сделать также, как и в большинстве операционных систем при вза-

имодействии с папками. Кликните по SampleScene правой кнопкой мыши, выберите в выпадающем меню Save, а затем Rename и напишите новое имя _0Scene:



Теперь в проекте существует сцена с названием _0Scene. Внутри сцены будут происходить основные действия с созданными игровыми объектами. Позже мы добавим и другие сцены, например сцену для стартового меню и меню с настройками проекта.

2.2 Импорт игровых персонажей

В качестве основного источника визуальных моделей и визуального оформления для своих игр на первом этапе вы можете использовать Unity Asset Store – это магазин, в котором можно приобрести различные ресурсы (assets) для Unity. Для простоты понимания мы будем называть их ассетами. В Asset Store можно найти:

–3D модели,

– звуки/музыку,

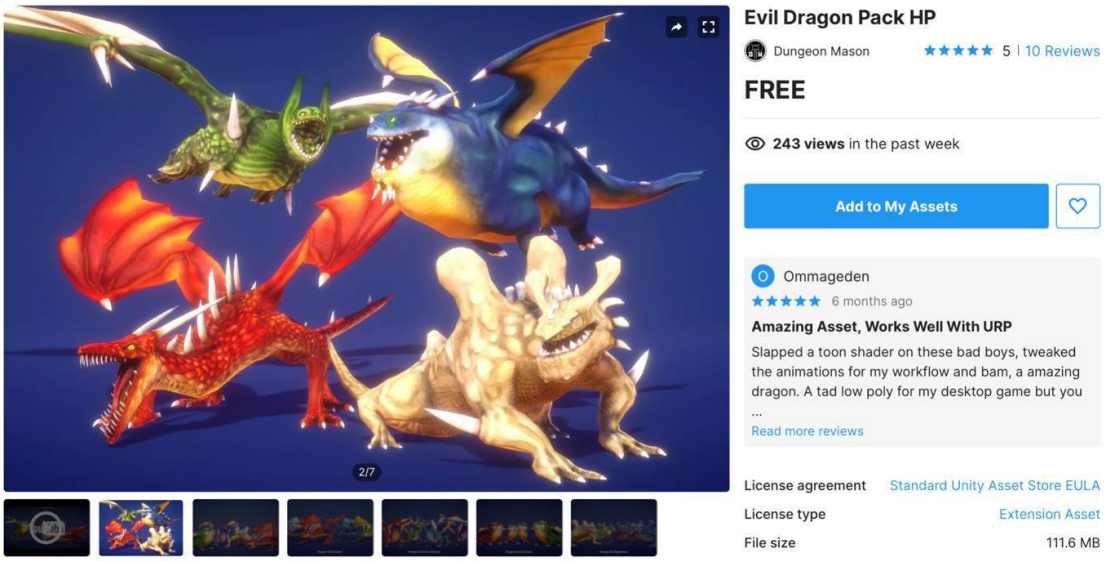
– элементы пользовательского интерфейса,

– шейдеры/частицы,

– наборы спрайтов,

– готовые скрип-файлы и многое другое.

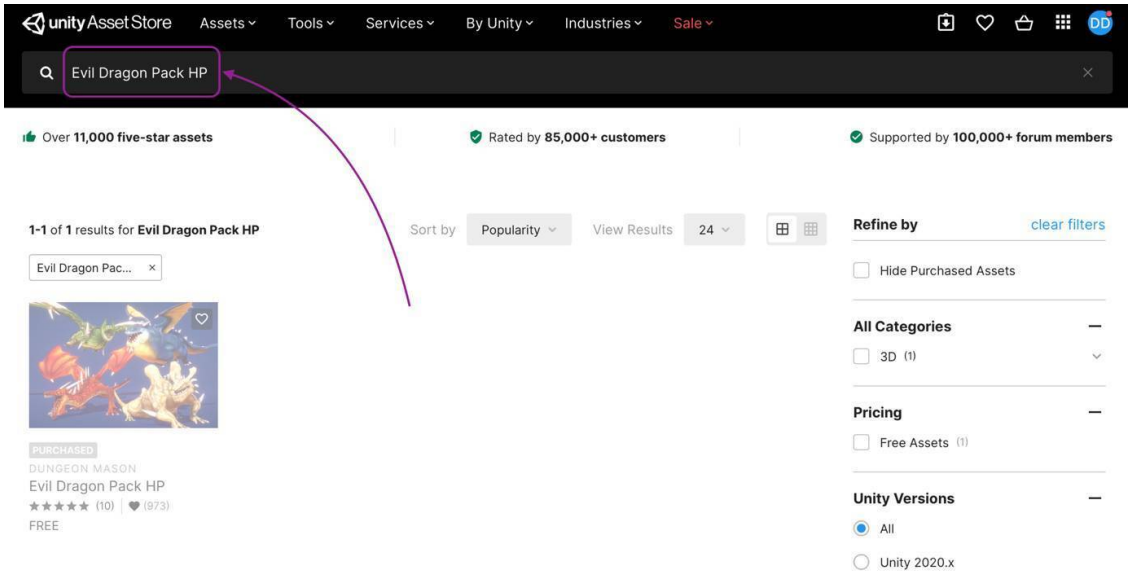
Основным источником персонажей в нашей игре станет Evil Dragon Pack HP, который распространяется бесплатно:



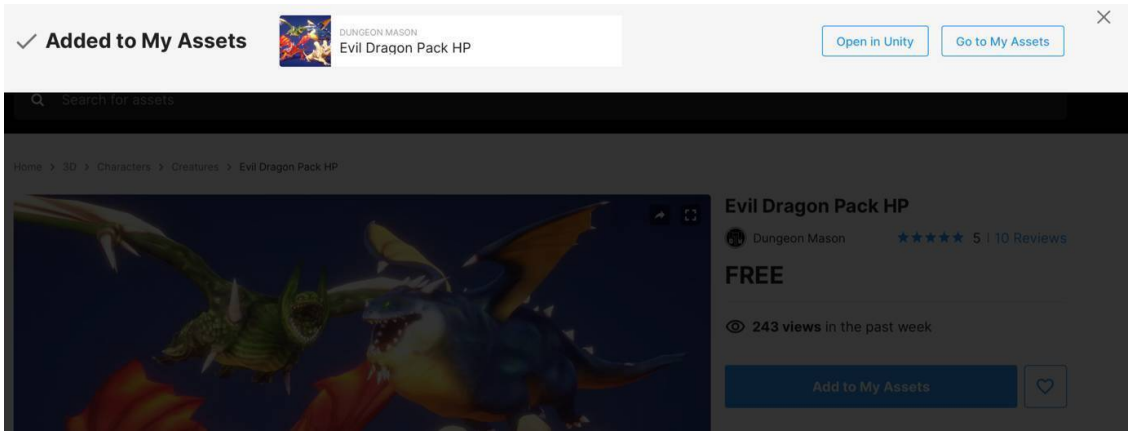
Ниже будет приведена подробная последовательность действия по добавлению ассета в Unity.

1. Зайдите на сайт assetstore.unity.com и войдите со своим Unity ID. Для добавления ассета используется личный кабинет пользователя с тем же Unity ID, который указывался при регистрации и входе в UnityHub (см. пункт 1.1).

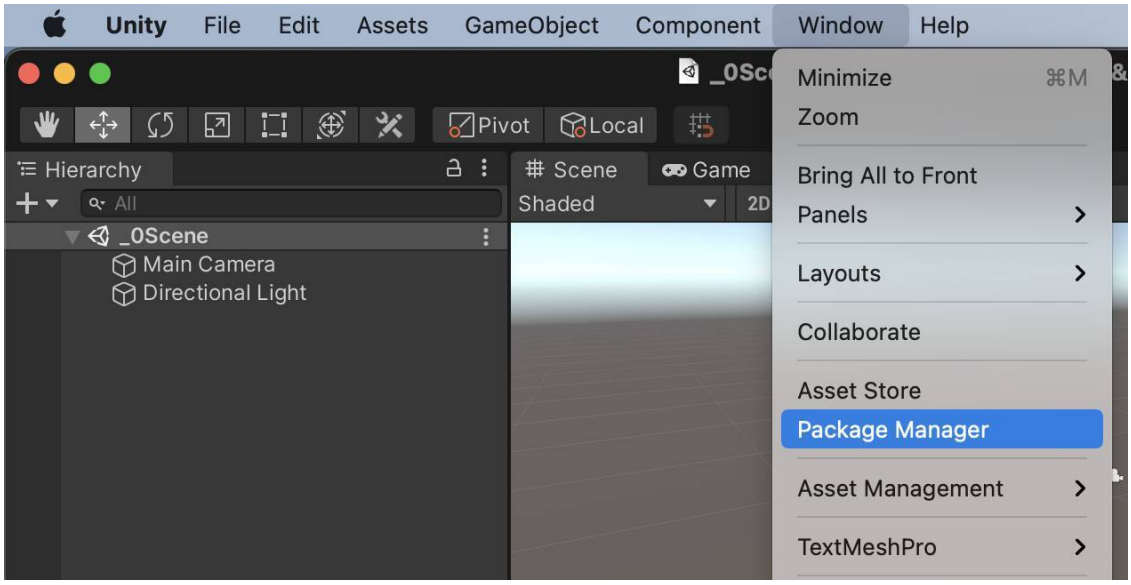
2. Используя строку поиска в верхней части сайта, найдите Evil Dragon Pack HP:



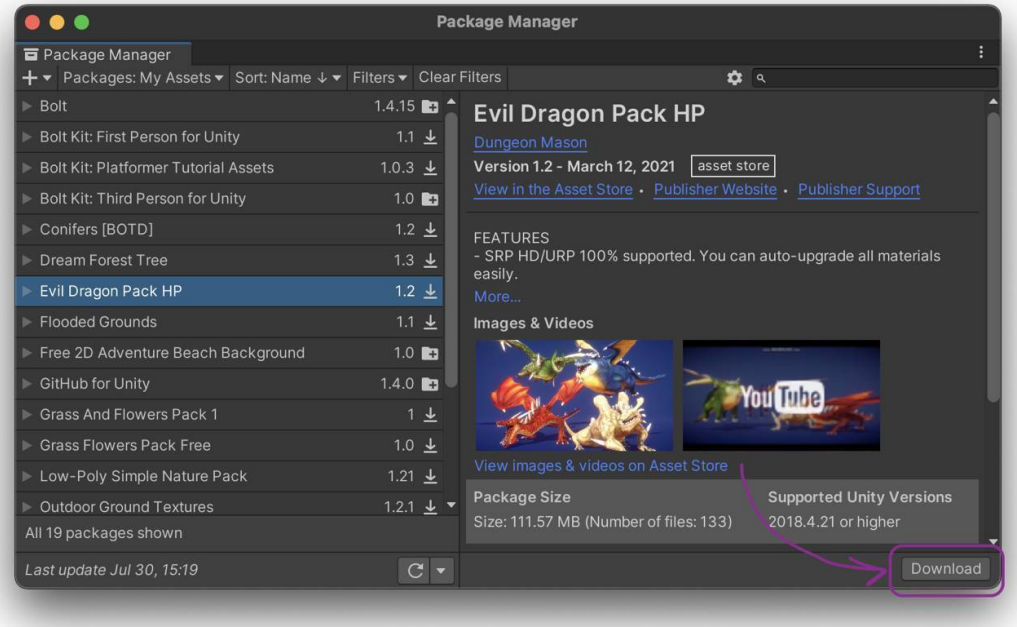
3. Откройте найденный ассет, перейдя на следующую страницу в браузере, и нажмите кнопку “Add to My Assets”. После этого ассет будет привязан к вашему Unity ID:



4. После добавления ассет-пак в своем личном кабинете, найти добавленный ассет можно в менеджере пакетов. Он находится внутри среды разработки Unity, установленной на вашем персональном компьютере. Для этого откройте Unity, на верхней панели выберите Window – Package Manager:



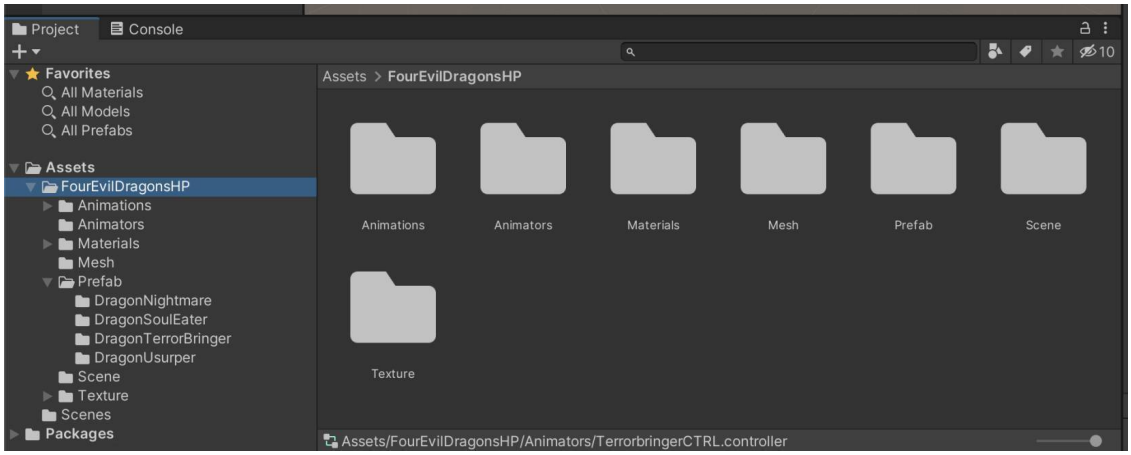
5. Среди установленных пакетов найдите Evil Dragon Pack и скачайте его, нажав кнопку Download:



6. После завершения скачивания, вместо кнопки Download появляется кнопка Import, нажмите ее. Появится новое окно со списком импортируемых файлов, еще раз нажмите Import чтобы импортировать файлы из ассета в проект Unity.

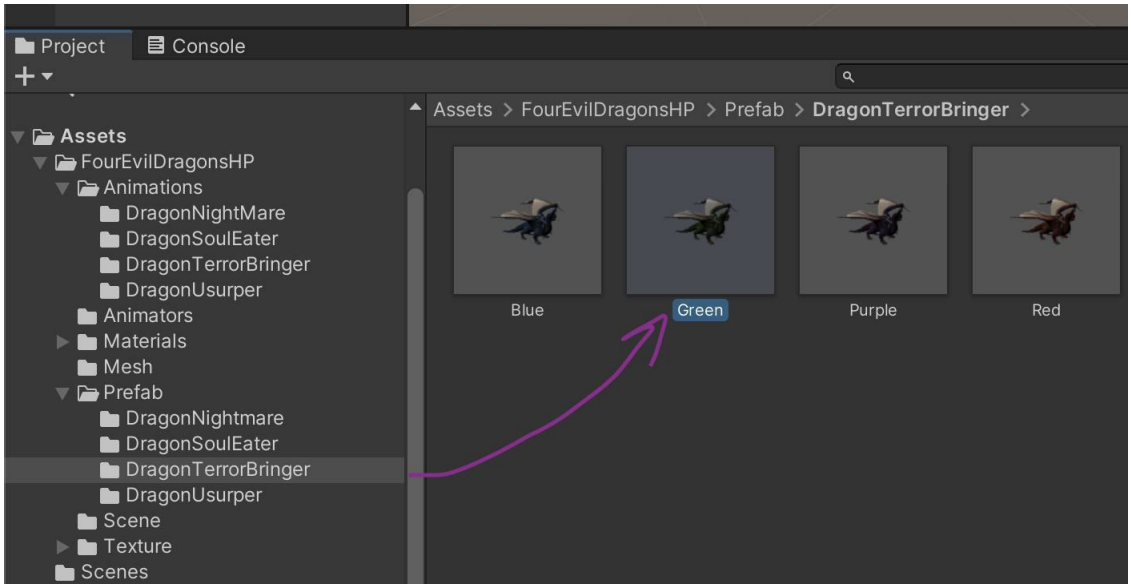
2.3 Добавление дракона с анимацией

Импортированный в проект ассет-пак содержит несколько видов драконов и большое количество анимации для каждого из них. Все они разбиты по папкам и структурированы так, как показано ниже:



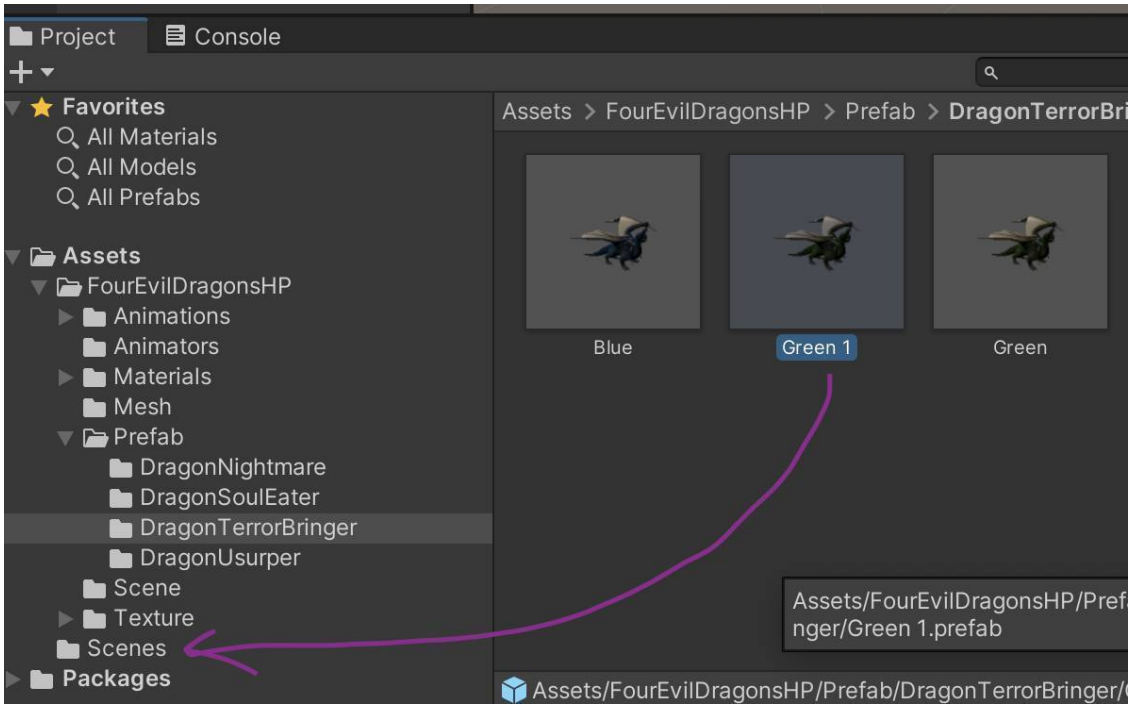
Наша задача заключается в том, чтобы выбрать некоторые 3D-модели и анимации, которые мы планируем использовать в проекте. После этого нужно перенести необходимые файлы в папку с проектом и добавить на сцену.

1. В окне Project откройте папку с префабом дракона Green, путь к папке: Assets – Prefab – DragonTerrorBringer.



2. Создайте дубликат дракона Green, для этого выберите его (кликнув левой кнопкой мыши) и нажмите комбинацию клавиш Ctrl+D (или Command+D для MacOS). Автоматически будет создана префаб-копия с именем Green 1.

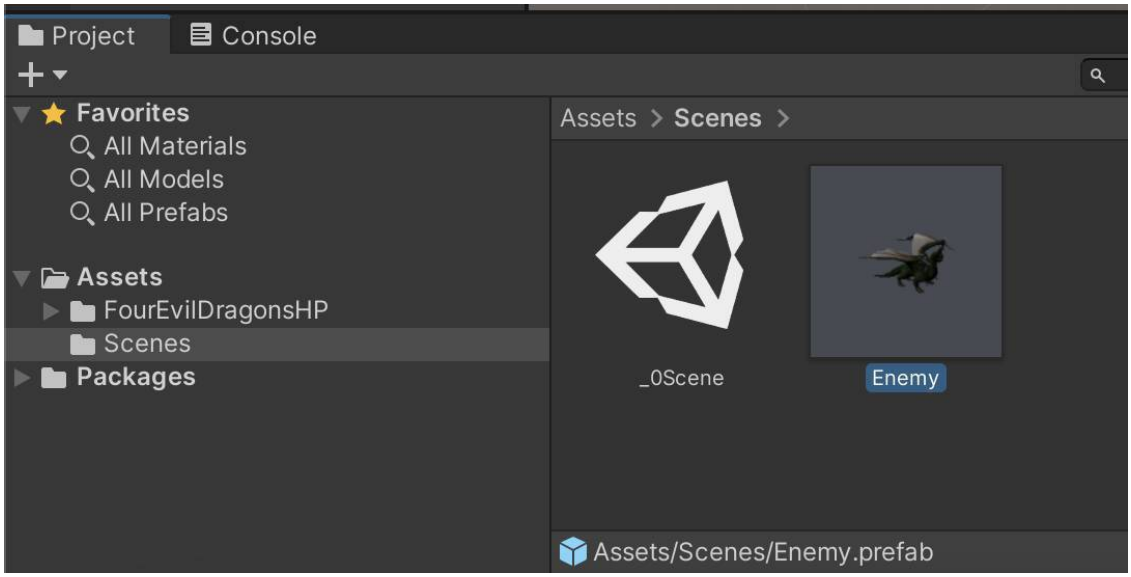
3. Перетащите префаб дракона с именем Green 1 в папку Scene. Для этого наведите курсор мыши на дракона Green 1 и зажав левую кнопку мыши перетащите в папку Scenes:



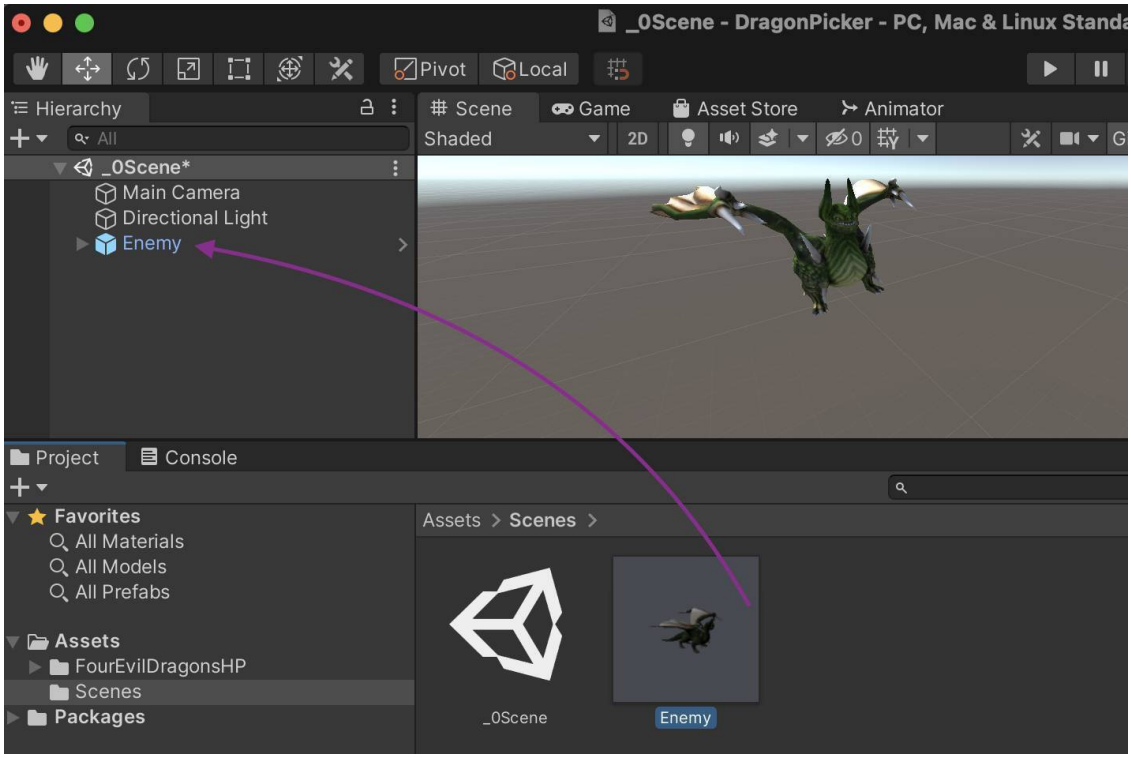
4. Таким образом, в папке Scenes вашего проекта теперь должно находиться два файла: сцена (_0Scene) и префаб с драконом Green 1. Переименуйте дракона Green 1 в Enemy (для этого кликните левой кнопкой мыши по объекту и нажмите Rename). Мы сделали это для того, чтобы:

– префаб с игровым персонажем находился в отдельной папке внутри нашего проекта;

– чтобы не работать с оригинальными файлами внутри импортированного ассет-пака, содержащего большое количество файлов, не используемых в проекте:



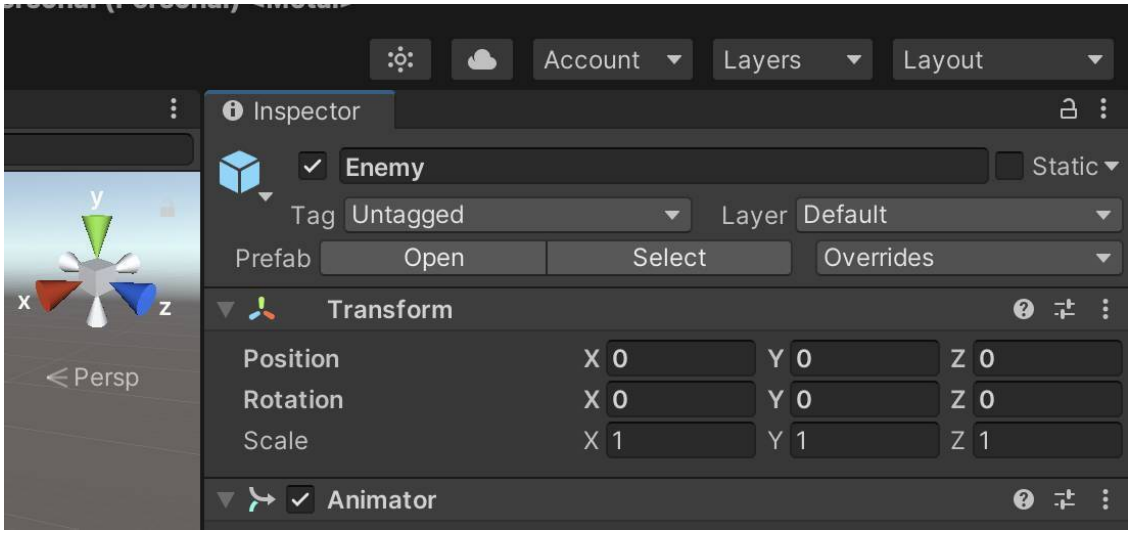
5. Теперь добавим персонажа Enemy на игровую сцену. Для этого перетащите префаб Enemy в окно Hierarchy:



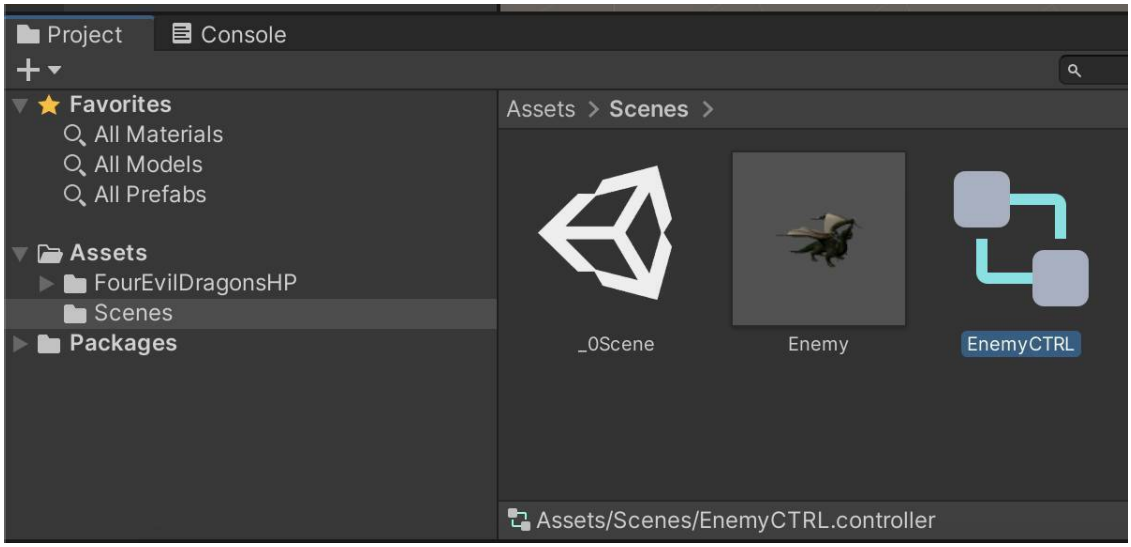
6. После этого игровой персонаж автоматически появится в окне Scene (в центральной части среды разработки Unity).

7. Как было указано ранее, координаты добавляемых на сцену объектов отображаются в окне Inspector. Чтобы узнать координаты добавленного персонажа Enemy, кликните по нему в окне Hierarchy, после этого в окне Inspector отобразятся его свойства. Нас интересуют параметры компонента Transform. Установите их значения:

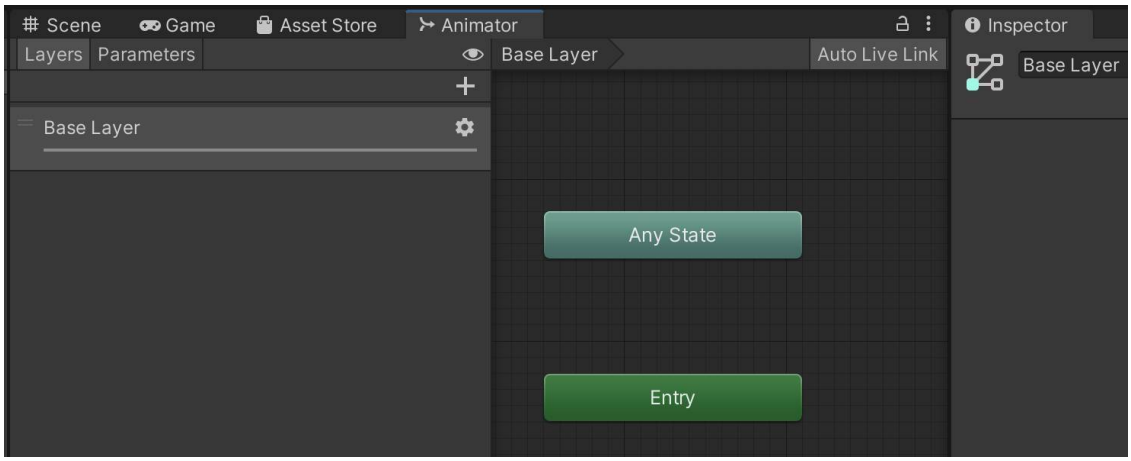
– Rotation: 0, 0, 0;
– Position: 0, 0, 0;
– Scale: 1, 1, 1.



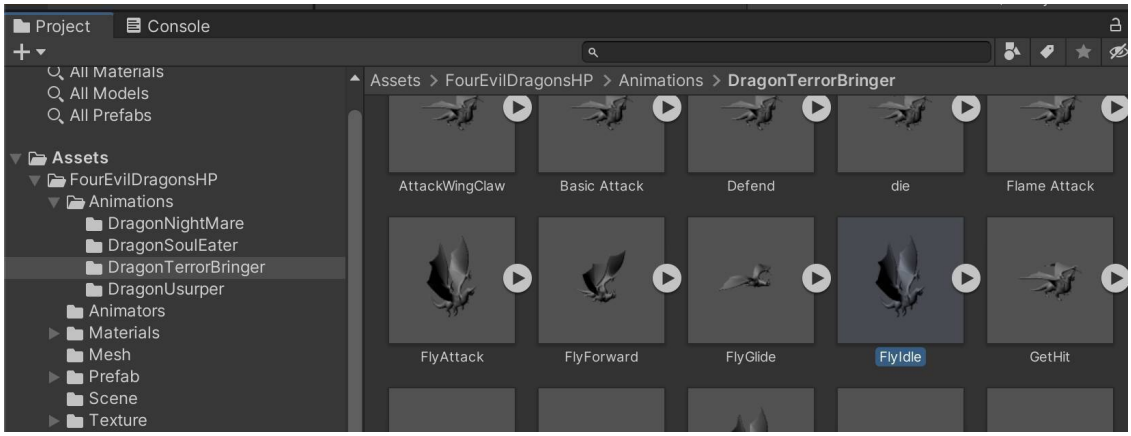
8. Теперь добавим анимацию игровому персонажу. В окне Project, внутри папки Scene (там где находится префаб Enemy) кликните правой кнопкой мыши и выберите из контекстного меню Create – Animator Controller. Дайте ему имя EnemyCTRL:



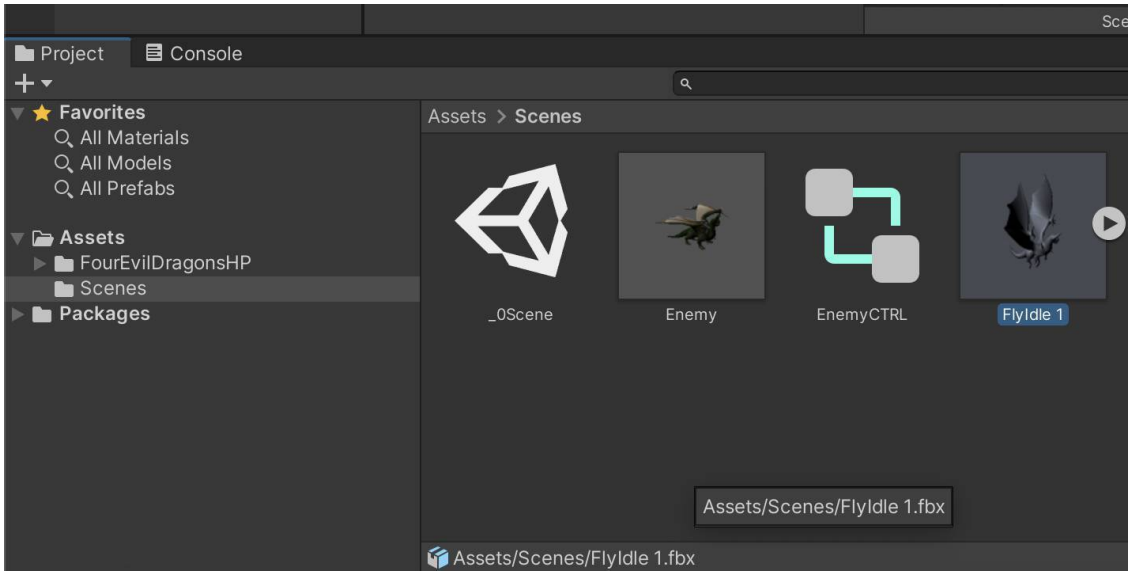
9. Контроллер нужен для того, чтобы строить “дерево анимации”, в котором описывается порядок и условия запуска анимации игрового объекта. Кликните дважды по контроллеру EnemyCTRL чтобы открыть его:



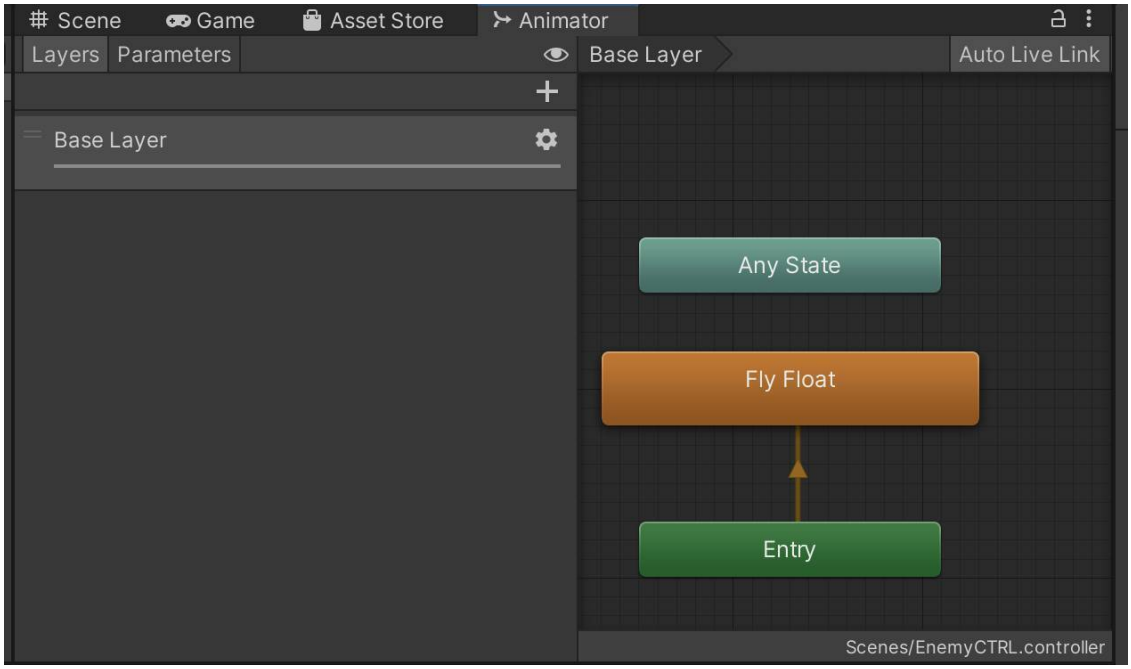
10. Окно Animator выглядит пустым, так как он был только что создан и в него не добавлено ни одной анимации. Давайте добавим подходящую анимацию парения дракона в воздухе, для этого найдите в скачанном Asset-паке анимацию FlyIdle. Она находится в папке Assets – FourEvilDragonHP – Animations – DragonTerrorBringer – FlyIdle:



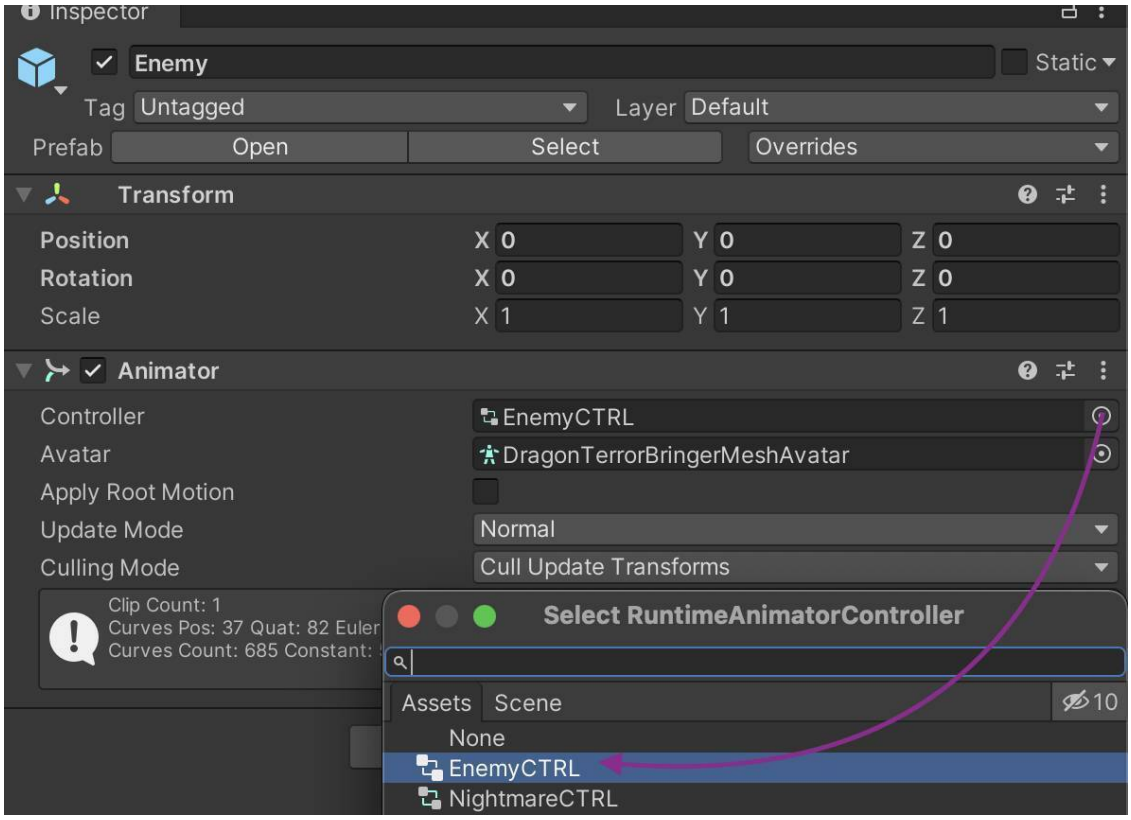
11. Создайте дубликат анимации (Ctrl+D или Command+D для MacOS), переместите созданную копию в папку Scenes:



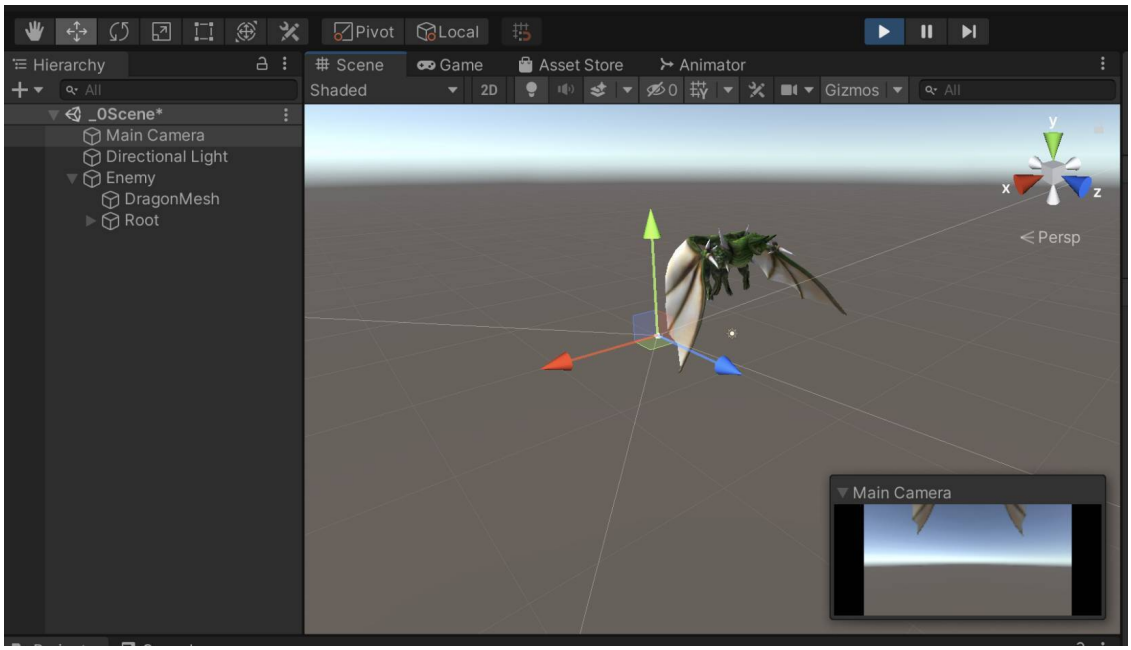
12. Теперь перетащите анимацию FlyIdle 1 из папки Scenes в окно Animator. Автоматически создается связь Entry -> Fly Float, которая говорит контроллеру о том, что после запуска игры должна запускаться стандартная анимация Fly Float:



13. Таким образом, был создал контроллер, и в него загружена стандартная анимация полета. Осталось лишь подключить контроллер к игровому персонажу. Для этого выберите персонажа Enemy и в окне Inspector найдите компонент Animator. В списке напротив поля Controller нажмите значок мишени и выберите созданный ранее контроллер с именем EnemyCTRL:



14. Теперь, если запустить сцену, нажав кнопку Play (стрелка 
в центральной верхней части среды разработки Unity), то можно заметить, что запускается анимация парящего в воздухе дракона. Однако, игровая камера еще не была настроена, поэтому ракурс может оказаться не очень удачным. Тем не менее, переключившись в окно Scene вы сможете увидеть все объекты, присутствующие на сцене:



15. Позднее мы настроем вид камеры и выберем более удачное расположение игрового персонажа Enemy. Нажмите кнопку Play еще раз, чтобы выйти из режима проигрывания сцены и вернуться к работе в среде разработки Unity.

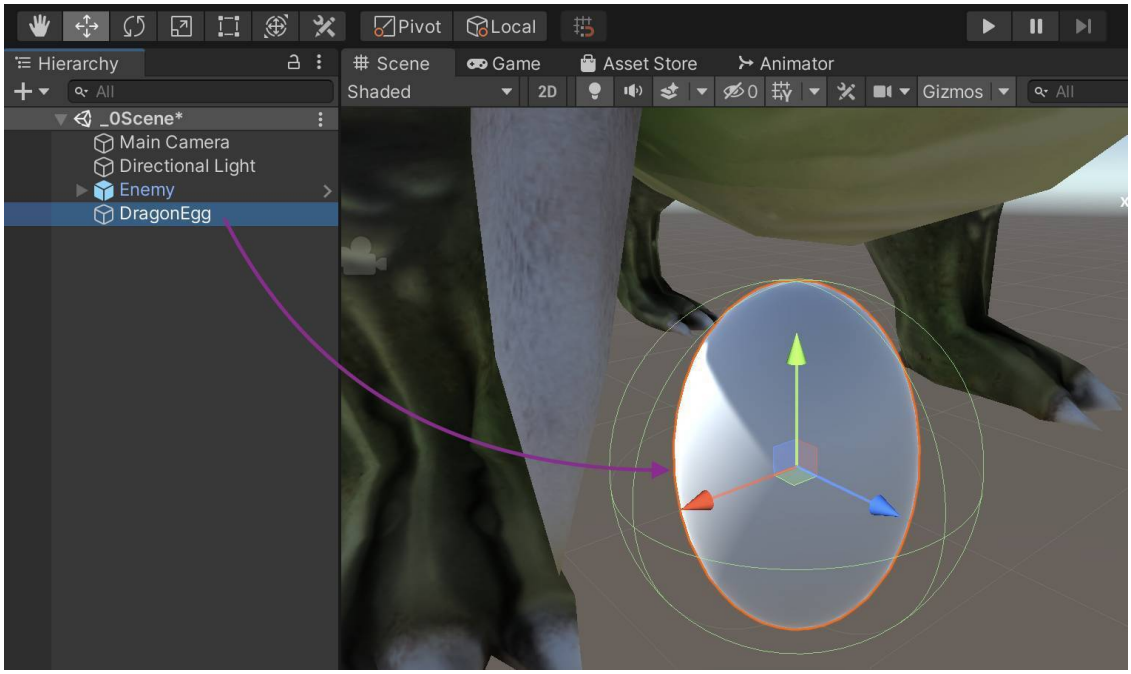
2.4 Создание игрового объекта – драконьего яйца

В Unity возможно создавать игровые объекты встроенными средствами. Конечно, без использования разработанных в сторонних пакетах игровых объектов не обойтись, но это не всегда необходимо. Особенно, если речь идет о простых примитивах. В этом пункте нашего руководства мы создадим игровой объект в виде драконьего яйца, при этом будем использовать встроенные возможности среды разработки Unity.

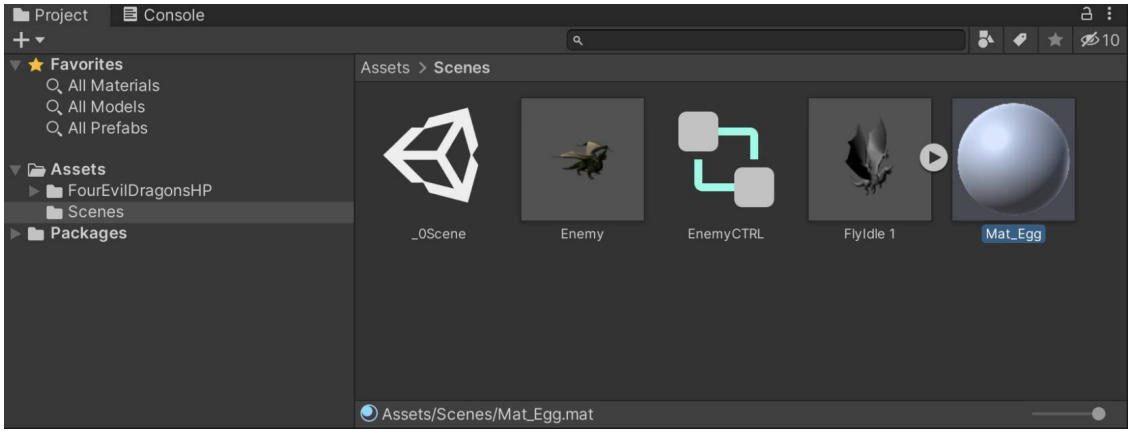
1. Создайте шаблон для драконьего яйца, выбрав в меню GameObject – 3D Object – Sphere. Переименуйте объект в DragonEgg. Проверьте, что его параметры Transform указаны в следующем виде:

– Position: 0, 0, 0;
– Rotation: 0, 0, 0;
– Scale: 1, 1.5, 1.

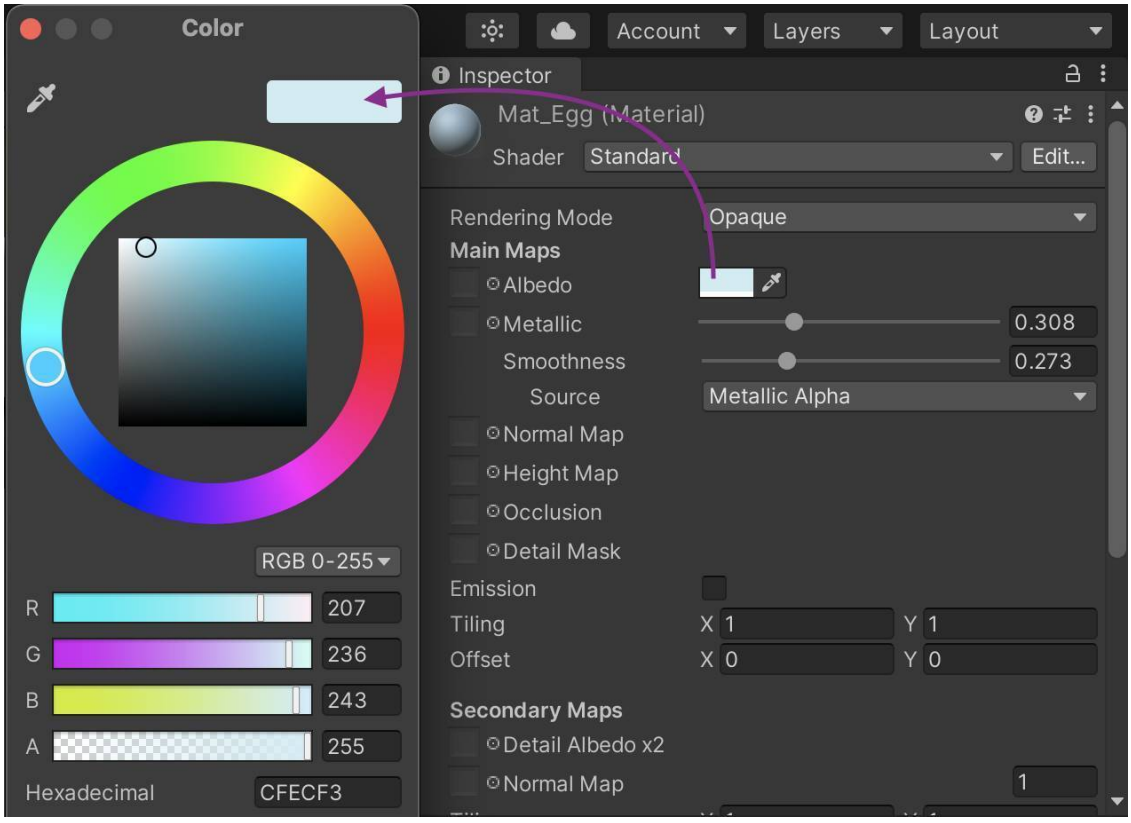
2. Созданное яйцо должно появиться под драконом, добавленным ранее:



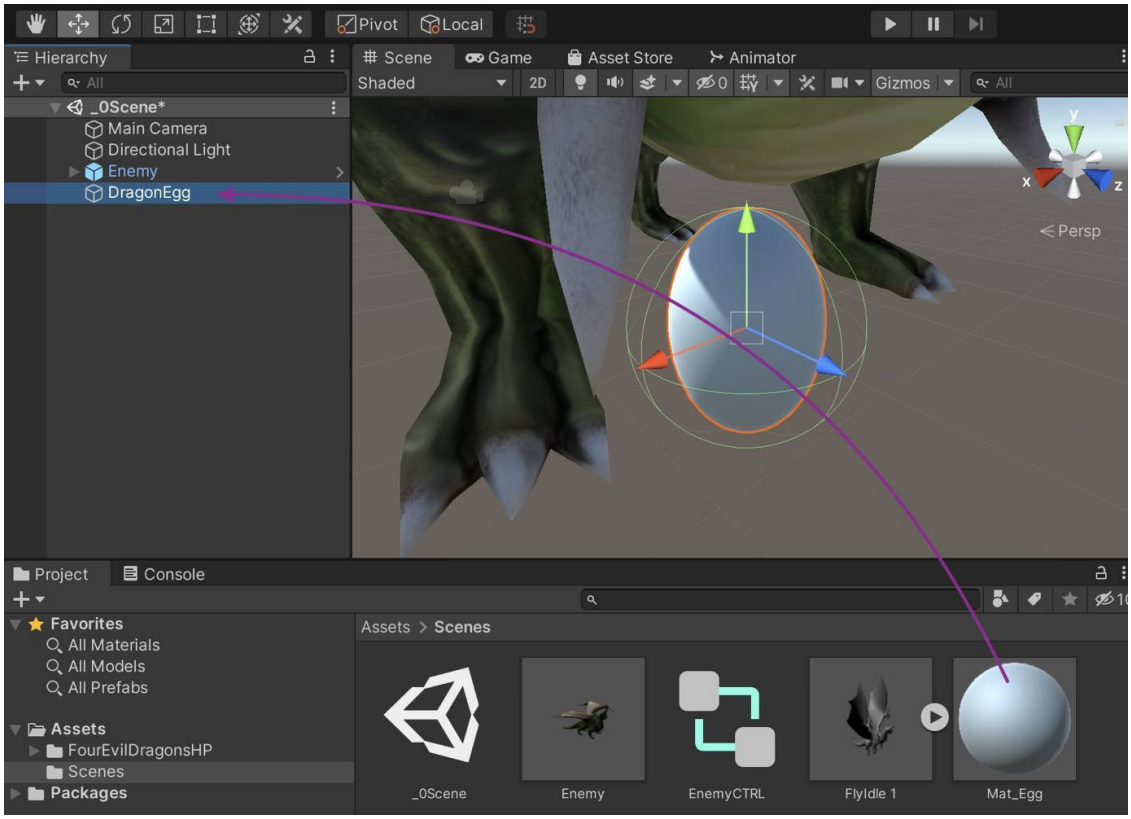
3. Внутри папки Assets – Scenes создайте материал, кликнув внутри Scenes правой кнопкой мыши и выбрав Create – Material. Назовите созданный материал Mat_Egg:



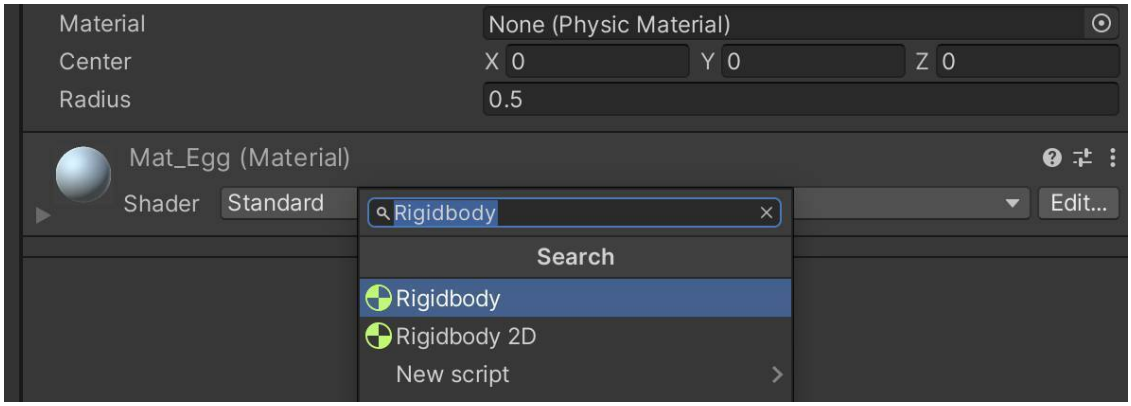
4. Кликните по материалу Mat_Egg, в окне Inspector (см. в правой части среды разработки) появятся свойства материала, доступные для редактирования. Например, кликнув по полю Albedo можно изменить основной цвет, также можете использовать бегунки Metallic и Smoothness. Можете поэкспериментировать и установить цвет, который вам покажется наиболее подходящим. В любом случае вы всегда сможете вернуться к этому пункту и отредактировать свойства материала. В нашем примере настройки цвета материала показаны ниже:



5. Чтобы назначить созданный материал объекту, просто перетяните материал Mat_Egg (из папки Assets – Scenes) на объект DragonEgg (в окне Hierarchy). Мы это уже неоднократно делали, так материал будет назначен созданному ранее 3D-объекту.

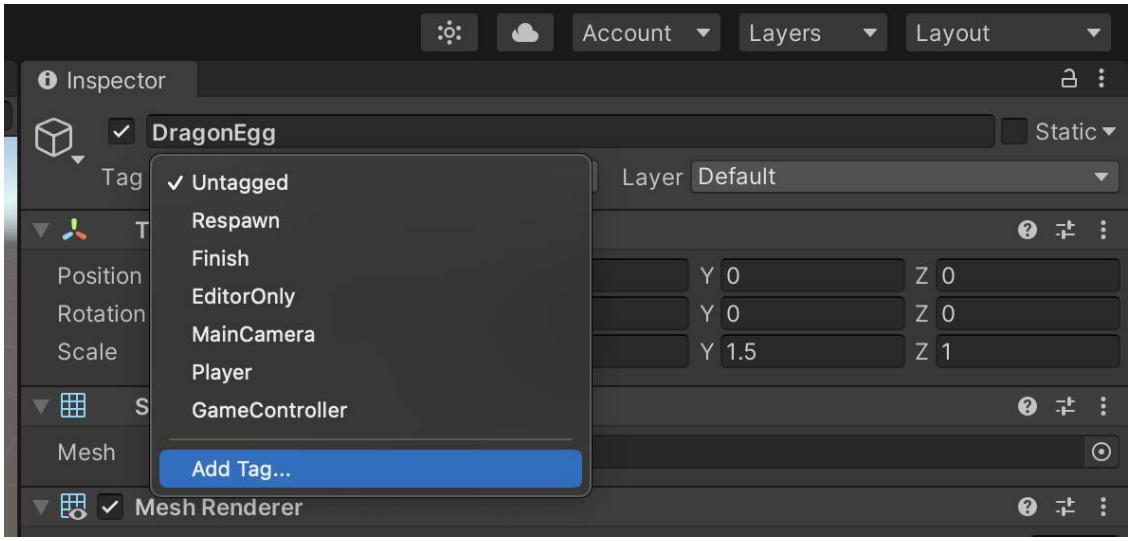


6. Выберите объект DragonEgg в панели Hierarchy, и в окне Inspector нажмите кнопку Add Component. Откроется окно добавления компонентов объекту. В появившейся строке поиска напишите Rigidbody и кликните по найденному компоненту Rigidbody в окне поиска Search.

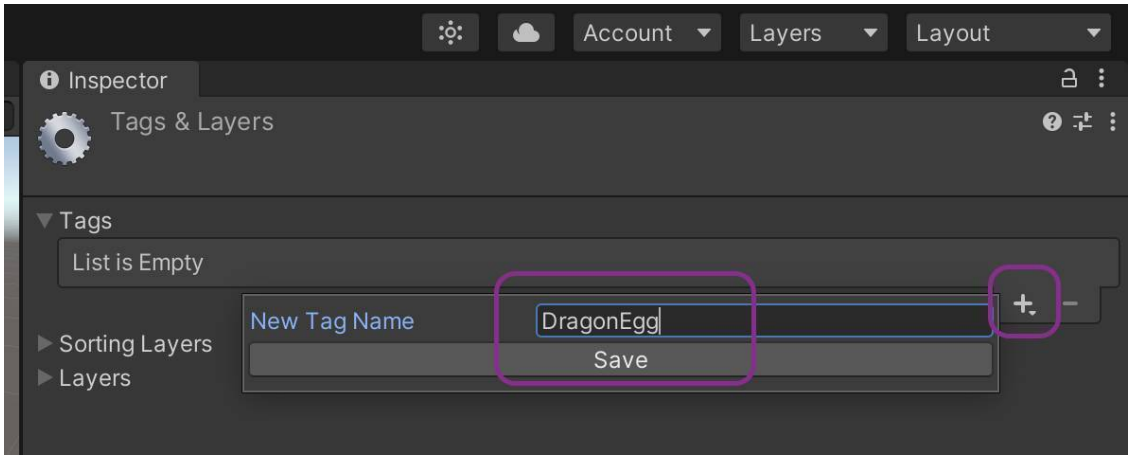


7. Так мы добавили яйцу свойство твердого тела. Теперь при воспроизведении сцены (нажатии кнопки Play) яйцо DragonEgg будет падать вниз, тогда как дракон Enemy останется парить в воздухе и махать крыльями.

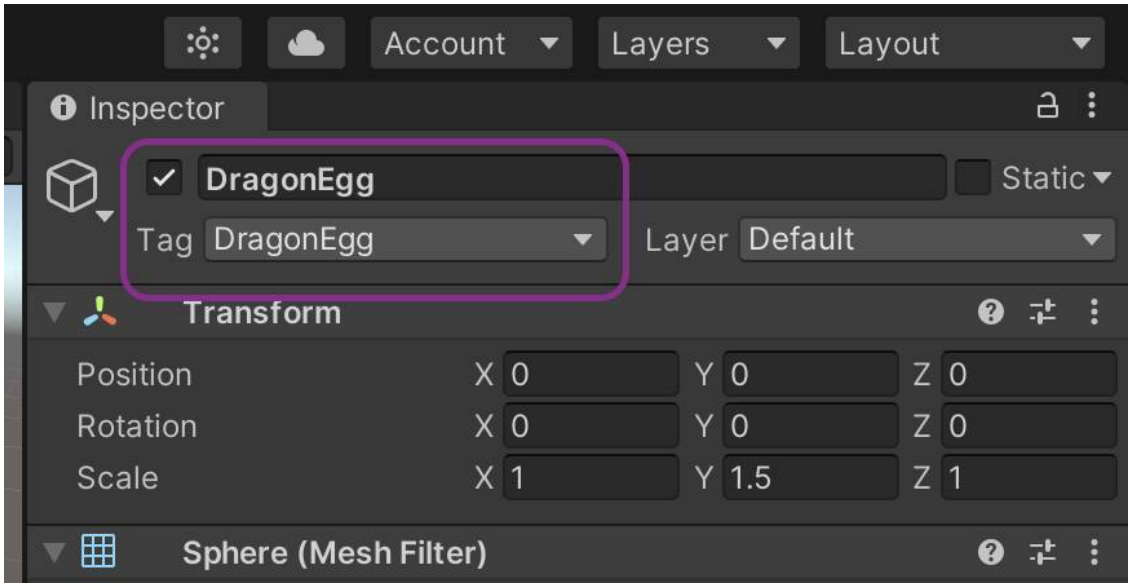
8. Так как на экране будет появляться множество объектов DragonEgg, позднее нам придется работать с массивами этих объектов. Для простоты дальнейшей работы привяжем к DragonEgg специальный тег. Для этого выберите DragonEgg в окне иерархии (Hierarchy), далее нажмите по кнопке Tag в окне Inspector и выберите Add Tag…:



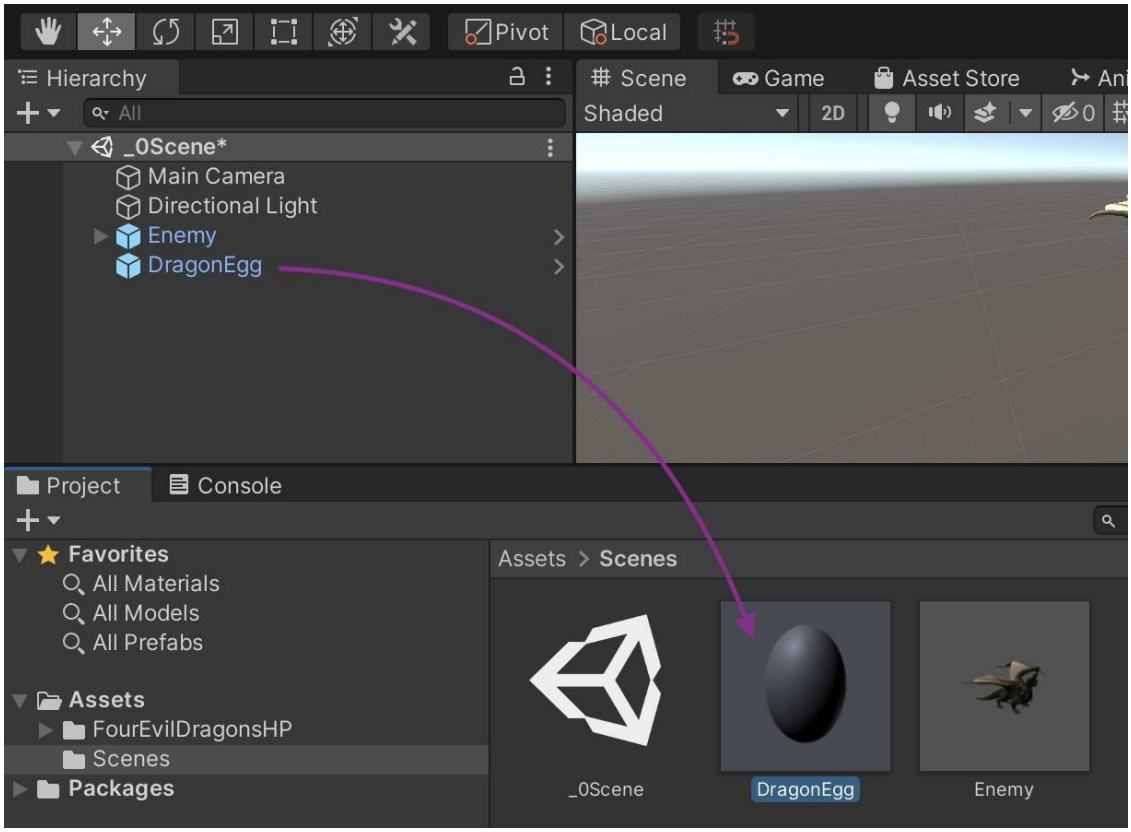
9. После этого откроется диспетчер тегов и слоев. Внутри окна Tags & Layers кликните по значку “+” и добавьте новый тег с именем DragonEgg. После того как напишете новое имя, нажмите Save:



10. Теперь в поле Tag (окно Inspector) для объекта DragonEgg можно задать новый тег с именем DragonEgg:



11. Так как яйцо дракона должно будет генерироваться автоматически, то его нахождение на сцене не требуется. В этом случае в Unity принято работать с так называемыми префабами. Unity prefab – это особый тип ассетов, позволяющий хранить весь GameObject со всеми компонентами и значениями свойств. Ранее мы уже работали с префабом – драконом Enemy, который был скачан из Asset Store. Теперь давайте создадим префаб для DragonEgg, чтобы это сделать просто перетащите 3D объект с именем DragonEgg из окна Hierarchy в окно Project:

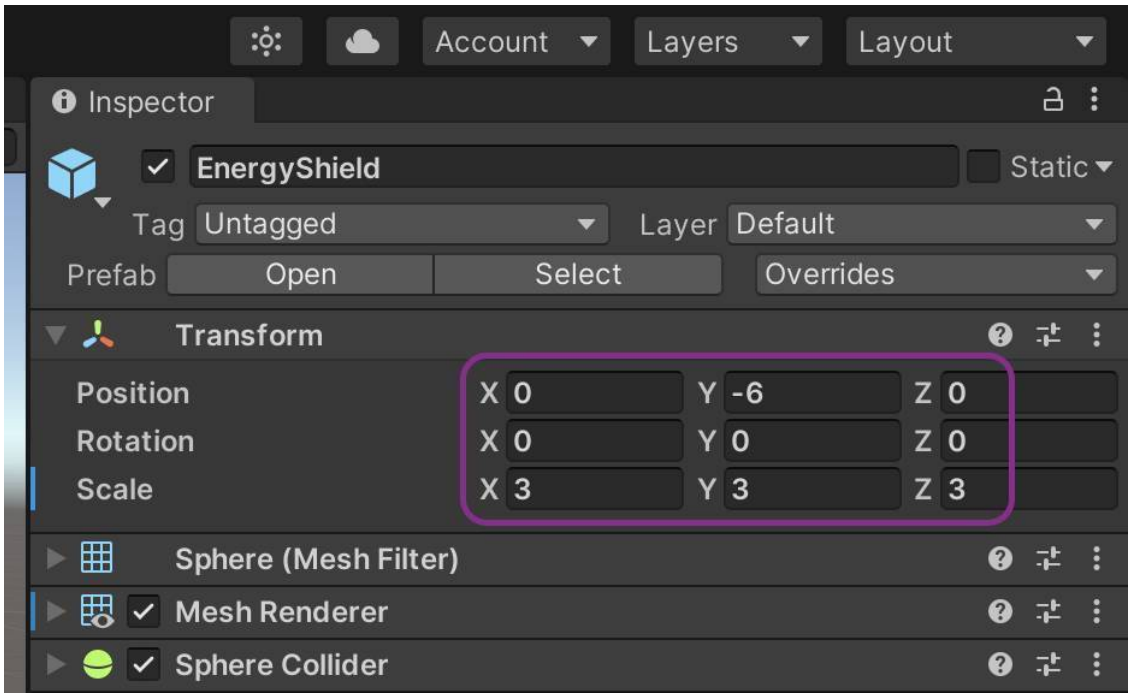


12. Если все сделано правильно, то DragonEgg в окне Hierarchy поменяет свой цвет. После того как шаблон будет создан, его можно удалить из окна Hierarchy (и соответственно со сцены). Удалите DragonEgg из Hierarchy (он останется в папке Assets – Scenes).

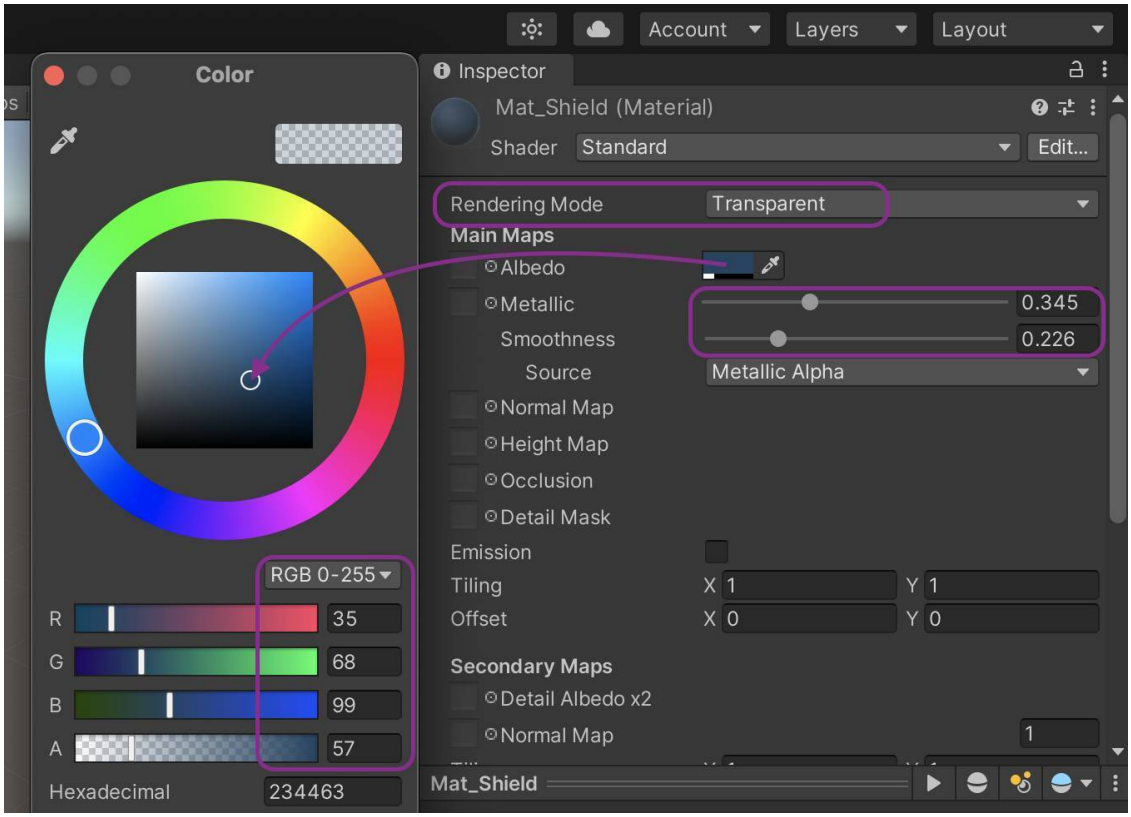
2.5 Создание игрового объекта – энергетического щита

1. Для создания игрового объекта – EnergyShield, выберите в главном меню GameObject – 3D Object – Sphere. Переименуйте объект в EnergyShield. Проверьте, что его параметры Transform указаны в следующем виде:

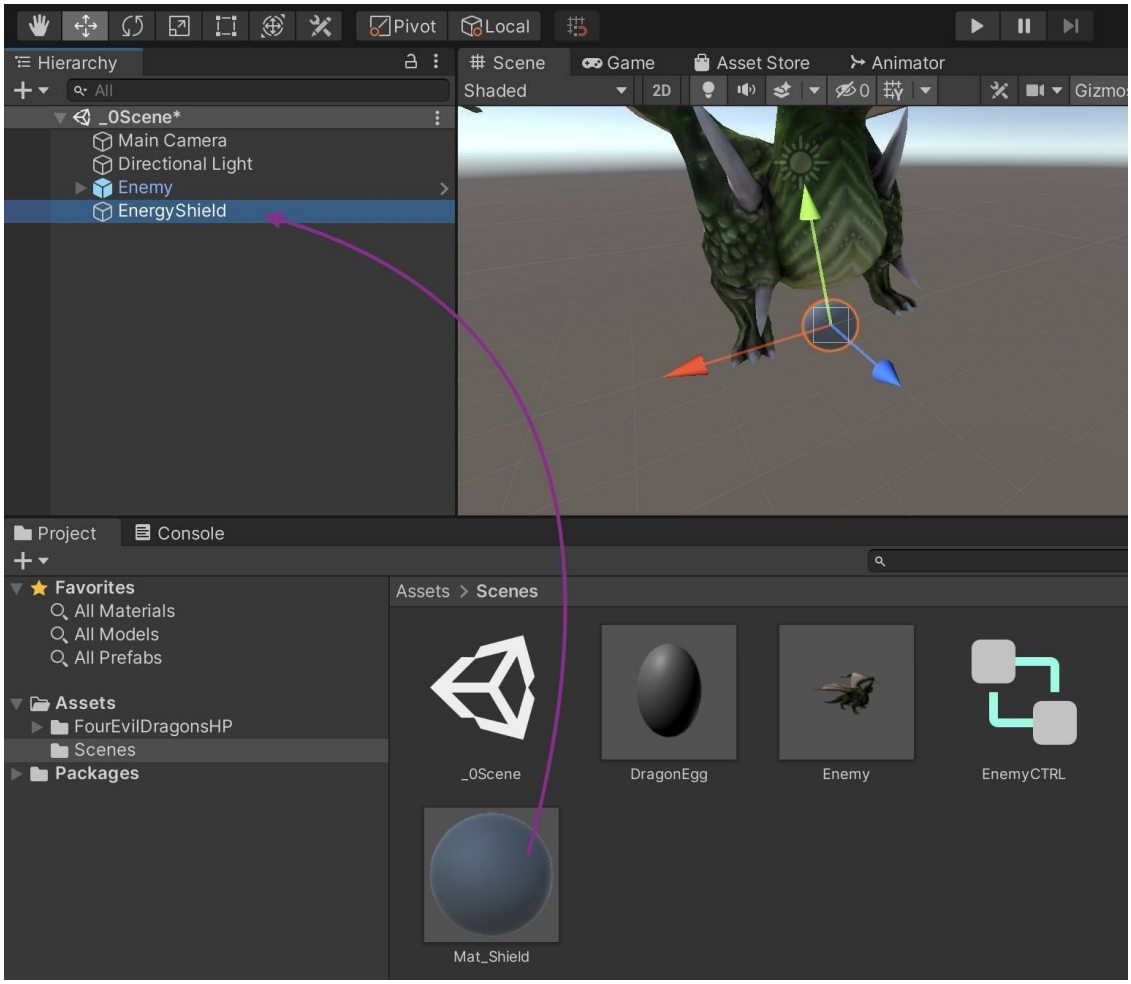
– Position: 0, -6, 0];
– Rotation: 0, 0, 0;
– Scale: 3, 3, 3;



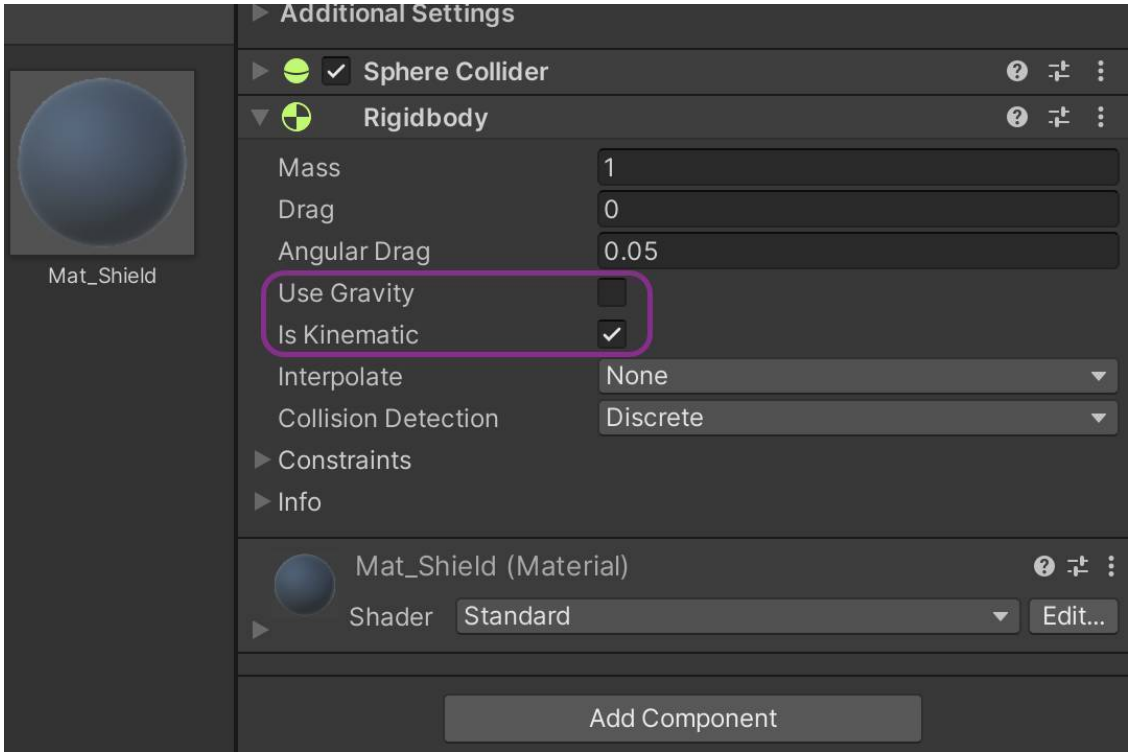
2. Для объекта EnergyShield создайте материал Mat_Shield. В свойствах материала установите для него Rendering mode – Transparent, а цвет, прозрачность как показано на рисунке ниже:



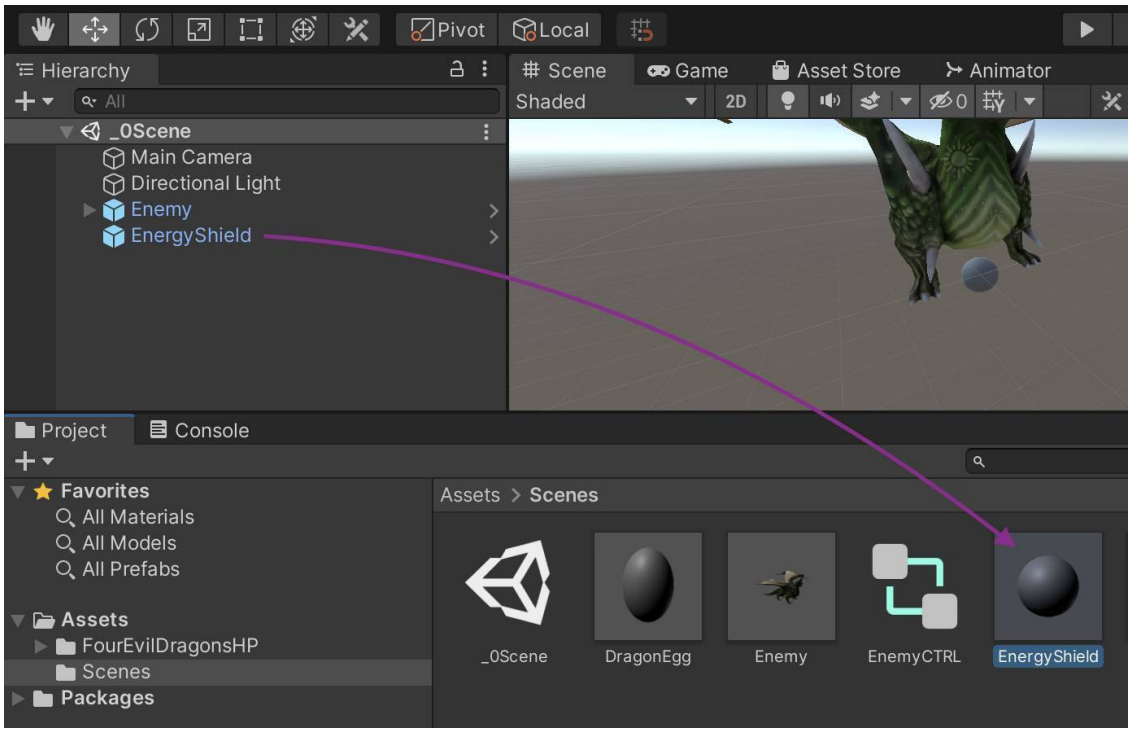
3. Перетащите материал Mat_Shield на объект EnergyShield, как это делали уже ранее:



4. Добавьте в EnegryShield компонент Rigidbody. Внутри компонента снимите флажок Use Gravity (при выключенной Use Gravity объект не будет падать). Там же установите флажок Is Kinematic (данный пункт отключает физику для объекта, отключаются силы, не работают столкновения объектов, эта модификация объекта нам будет полезна далее):



5. Создайте Prefab для объекта EnegryShield, перетащив его из окна Hierarchy в Project:

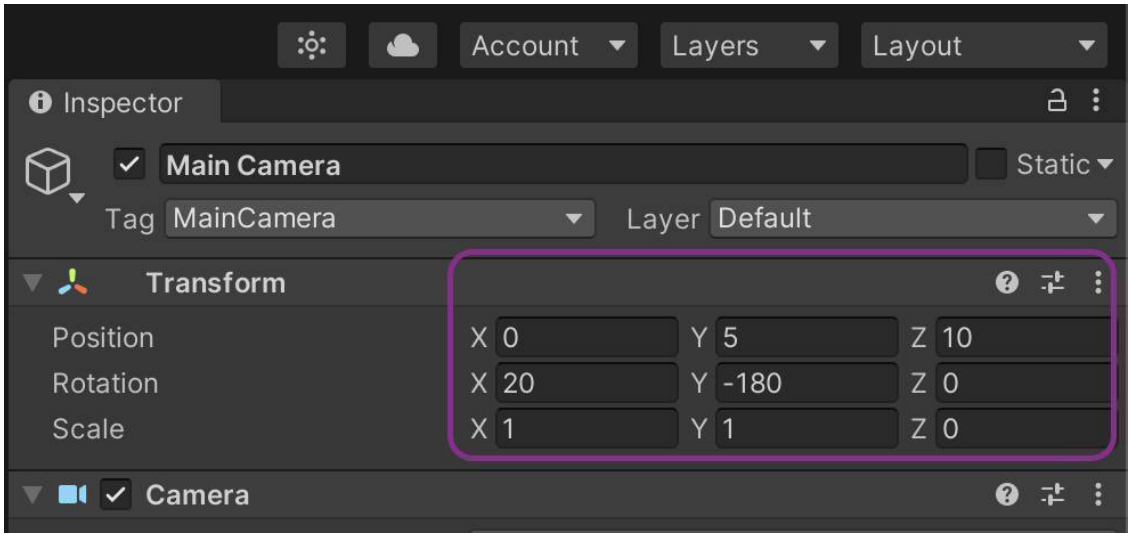


6. Сохраните сцену, кликнув правой кнопкой мыши по сцене и выбрав Save Scene.

2.6 Настройка камеры и игровой области

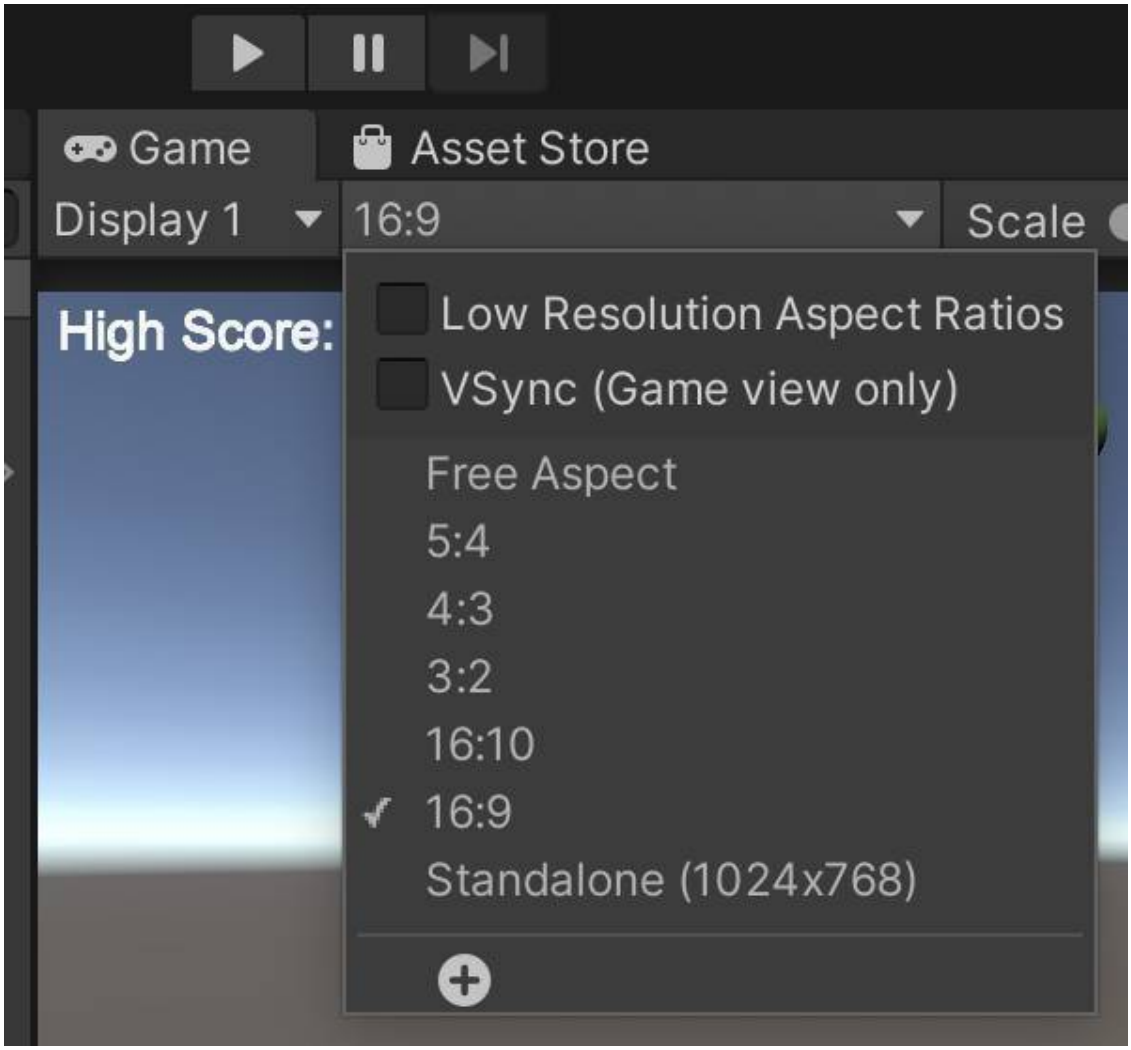
Как и многие вещи в разработке игр, поиск оптимальных настроек камеры – итеративный процесс. Не обязательно следовать всем указаниям в руководстве, вы всегда можете изменить настройки камеры таким образом, чтобы получить наиболее подходящие на ваш взгляд настройки вида игровой области.

1. Выберите объект Main Camera в иерархии объектов. Настройте компонент Transform:
– Position: 0, 5, 10;
– Rotation: 20, – 180, 0;
– Scale: 1, 1, 0;

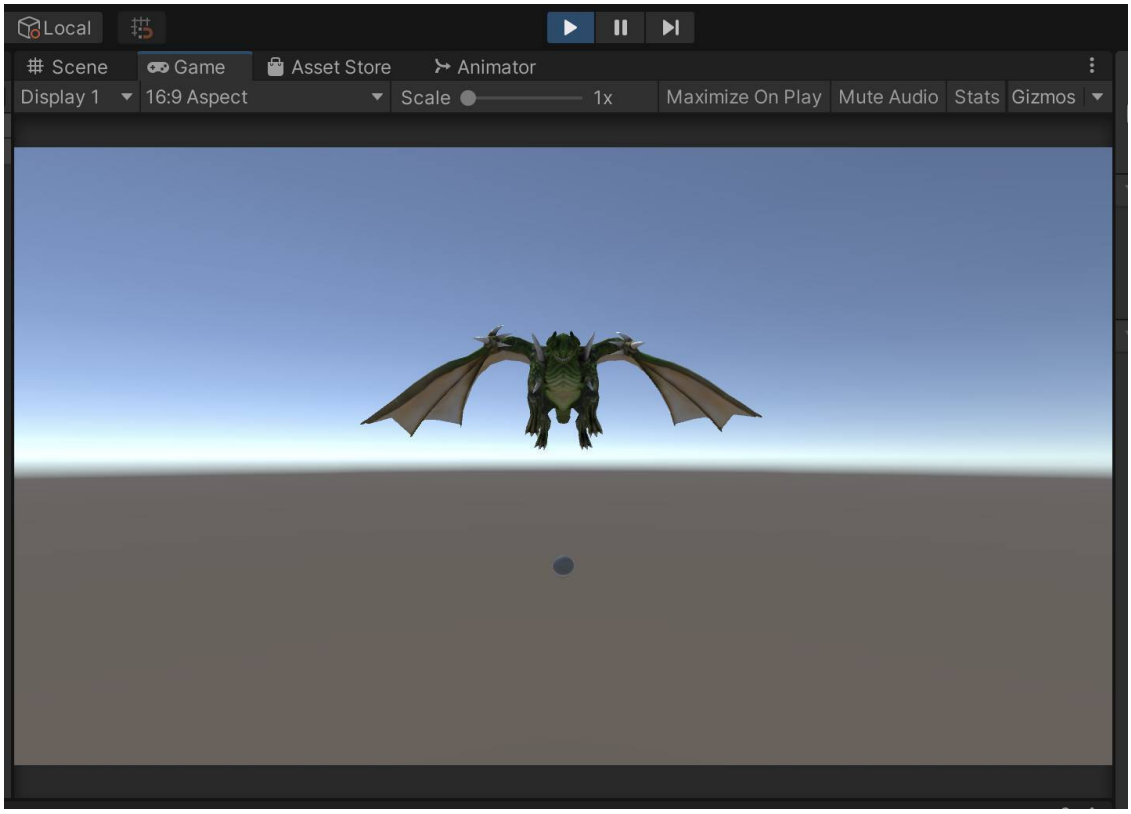


2. В инспекторе для компонента Camera установите в поле Projection – Orthographic, в поле Size значение 14:

3. Еще один важный фактор – соотношение сторон панели Game. Вверху панели отображается значение Free Aspect. Выберите из списка 16:9 – стандартное соотношение сторон для большинства современных экранов.



4. При запуске сцены (кнопка Play) будет показан вид окна игры в соответствии с выбранными настройками камеры:



На данный момент в окне должен быть виден дракон, расположенный по центру экрана таким образом, чтобы размах крыла не выходил за верхние/нижние и боковые края экрана. Также на сцене видна расположенная по центру системы координат сфера EnergyShield. В следующем разделе мы приступим к программированию игровых объектов и сделает так, чтобы сцена ожила и с ней можно было взаимодействовать.

Выводы

Таким образом, во второй части мы создали заготовку для нашего будущего проекта и даже добавили парящего дракона с анимацией. Если у вас будет желание разобраться в проекте чуть более глубоко и получить к концу практикума игру, которая будет отличаться от этой, рекомендуем вам попробовать самостоятельно внести следующие изменения в проект:

– Замените префаб с драконом на любой другой из скачанного ассет-пака.

– Замените анимацию полета.

– Можете поэкспериментировать со значениями scale в префабе, это позволит изменить размер игрового объекта.

Часть 3. Программирование игровых объектов

Введение

При написании сценариев мы будем использовать пошаговое добавление строк кода, давая описание команд по ходу добавления нового функционала в скрипт-файлы. Как уже было отмечено ранее, действия игровых объектов описываются в так называемых Script-файлах. Эти файлы содержат код на языке C# и имеют расширение .cs. При этом следует отметить, что предварительная компиляция файлов не требуется, Unity берет эту работу на себя в режиме реального времени в процессе работы в среде разработки.

3.1 Скрипт-файл EnemyDragon

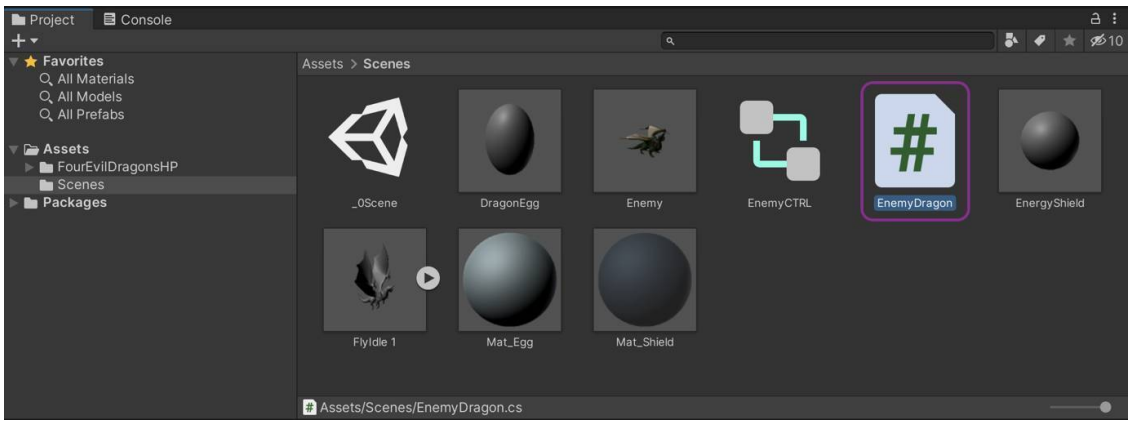
В пункте 3.1 мы создадим скрипт-файл, который будет выполнять следующие действия:

– перемещать игрового персонажа Enemy влево и вправо;

– через случайные интервалы времени вероятность изменения направления движения будет изменяться;

– сбрасывать яйцо (DragonEgg) через случайные интервалы времени.

1. Создайте сценарий с именем EnemyDragon.cs в папке Assets – Scenes. Для этого кликните правой кнопкой мыши внутри содержимого окна Scene и из выпадающего меню выберите Create – C# Script, дайте имя файлу EnemyDragon:



2. Откройте сценарий EnemyDragon.cs, кликнув по нему два раза (файл автоматически откроется в среде разработки Microsoft Visual Studio) и введите следующий код:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;
    void Start()
    {
    }
    void Update()
    {
    }
}
```

// End Code

Листинг дублируется ниже в виде скриншота из MS Visual Studio.

EnemyDragon.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyDragon : MonoBehaviour
6   {
7       [Header("Set in Inspector")]
8       public GameObject dragonEggPrefab;
9       public float speed = 1f;
10      public float timeBetweenEggDrops = 1f;
11      public float leftRightDistance = 10f;
12      public float chanceDirections = 0.1f;
13
14      void Start()
15      {
16      }
17
18      void Update()
19      {
20      }
21  }
22
```

В листинге мы добавили следующие строки кода:

– с помощью команды [Header("Set in Inspector")] создализаголовок в окне Inspector;

– public GameObject dragonEggPrefab создает префаб для драконьего яйца DragonEgg;

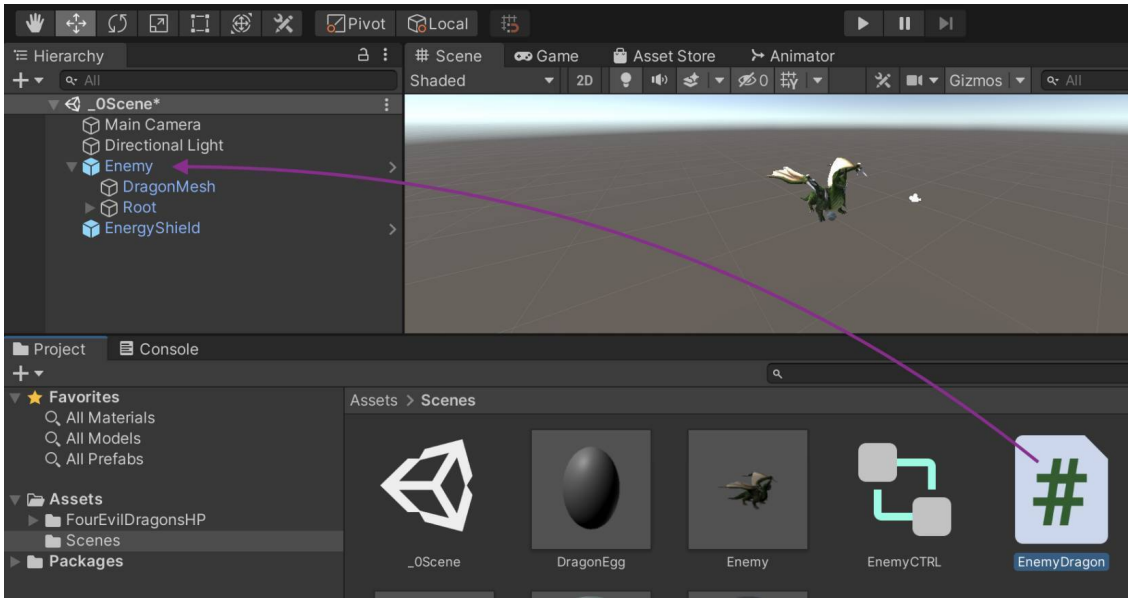
– созданная переменная speed определяет скорость движения объекта DragonEgg;

– переменная timeBetweenEggDrops определяет скорость генерации объектов DragonEgg;

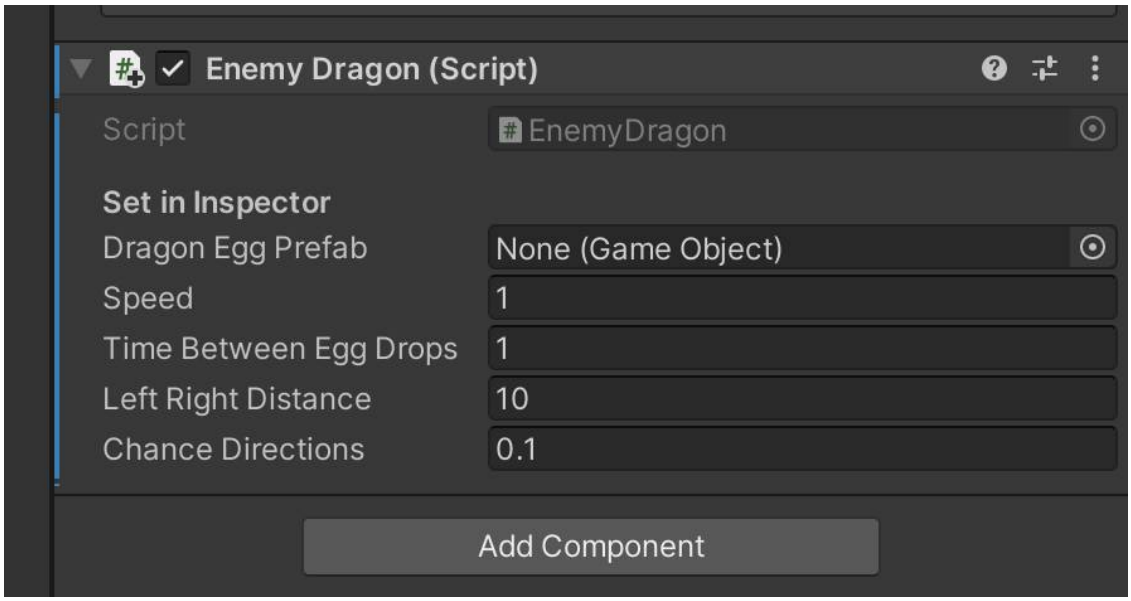
– переменная leftRightDistance определяет расстояние от края экрана, на котором меняется направление движения дракона;

– chanceDirections определяет вероятность изменения направления движения.

3. Подключите сценарий к игровому объекту Enemy. Сделать это можно так же, как и ранее – перетаскиванием скрипта EnemyDragon.cs, на объект Enemy:



4. Теперь, если выбрать объект Enemy в окне Hierarchy, то можно увидеть, что в инспекторе объекта (в правой части среды разработки) добавился компонент с соответствующим именем Enemy Dragon (Script). Обратите внимание, что компонент содержит поля, для которых мы создали переменные на языке C#. Это значит, что в дальнейшем мы сможем изменять значения переменных прямо в этих полях, находясь в среде разработки Unity без лишнего обращения к коду:



5. Внутри скрипт-файла EnemyDragon.cs добавьте в метод Update следующие строки кода, выделенные жирным шрифтом:

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyDragon : MonoBehaviour
```

```
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;
    void Start()
    {
    }
    void Update()
    {
        Vector3 pos = transform.position; //1
        pos.x += speed * Time.deltaTime; //2
        transform.position = pos; //3
    }
}
```

// End Code

Листинг дублируется ниже в виде скриншота из MS Visual Studio.

EnemyDragon.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyDragon : MonoBehaviour
6   {
7       [Header("Set in Inspector")]
8       public GameObject dragonEggPrefab;
9       public float speed = 1f;
10      public float timeBetweenEggDrops = 1f;
11      public float leftRightDistance = 10f;
12      public float chanceDirections = 0.1f;
13
14      void Start()
15      {
16      }
17
18      void Update()
19      {
20          Vector3 pos = transform.position; //1
21          pos.x += speed * Time.deltaTime; //2
22          transform.position = pos; //3
23      }
24  }
25
```

Каждому закомментированному номеру соответствует следующее:
– Vector3 pos – это локальная переменная для хранения текущий позиции (//1);

– x увеличивается на произведение скорости и временного интервала Time.deltaTime (//2). Это встроенная переменная, которая будет изменять свое значение при разной частоте кадров. Если говорить простыми словами, то она отвечает за плавность игры при разном FPS;

– изменение значения pos записывается обратно в transform.position (//3).

6. Сохраните сценарий и нажмите Play в Unity. Теперь дракон (игровой объект Enemy) должен медленно начать двигаться по экрану. Можете самостоятельно подобрать такое значение скорости Speed в окне Inspector, которое покажется вам наиболее подходящим.

7. Чтобы дракон не улетал за край экрана следует добавить в метод Update строки кода:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;
    void Start()
    {
    }
    void Update()
    {
        Vector3 pos = transform.position;
        pos.x += speed * Time.deltaTime;
        transform.position = pos;
        if (pos.x < -leftRightDistance) //1
        {
            speed = Mathf.Abs(speed);
        }
        else if (pos.x > leftRightDistance) //2
        {
            speed = -Mathf.Abs(speed);
        }
    }
}
```

// End Code

Листинг дублируется ниже в виде скриншота из MS Visual Studio.

EnemyDragon.cs

EnemyDragon ▸ No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnemyDragon : MonoBehaviour
6   {
7       [Header("Set in Inspector")]
8       public GameObject dragonEggPrefab;
9       public float speed = 1f;
10      public float timeBetweenEggDrops = 1f;
11      public float leftRightDistance = 10f;
12      public float chanceDirections = 0.1f;
13
14      void Start()
15      {
16      }
17
18      void Update()
19      {
20          Vector3 pos = transform.position;
21          pos.x += speed * Time.deltaTime;
22          transform.position = pos;
23
24          if (pos.x < -leftRightDistance) //1
25          {
26              speed = Mathf.Abs(speed);
27          }
28          else if (pos.x > leftRightDistance) //2
29          {
30              speed = -Mathf.Abs(speed);
31          }
32      }
33  }
```

Каждому закомментированному номеру соответствует следующее:

– если значение pos.x оказалось меньше значения leftRightDistance – переменная speed имеет положительное значение (движение вправо) (//1);

– иначе если значение pos.x оказалось больше значения leftRightDistance – переменной speed присваивается отрицательное значение (движение влево) (//2).

8. Сохраните сценарий EnemyDragon.cs в Unity и нажмите Play. Проверьте корректность движения игрового объекта Enemy, согласно написанному сценарию. Теперь он должен менять направление движения при приближении к краю экрана.

9. Для добавления случайного изменения направления перемещения дракона, создайте под методом Update() новый метод с именем FixedUpdate(). FixedUpdate вызывается точно 50 раз в секунду и в отличие от Update() не зависит от быстродействия компьютера.

// Start Code

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

```
public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;
    void Start()
    {
    }
    void Update()
    {
        Vector3 pos = transform.position;
        pos.x += speed * Time.deltaTime;
        transform.position = pos;
        if (pos.x < -leftRightDistance)
        {
            speed = Mathf.Abs(speed);
        }
        else if (pos.x > leftRightDistance)
        {
            speed = -Mathf.Abs(speed);
        }
    }
    private void FixedUpdate()
    {
        if (Random.value < chanceDirections)
        {
                speed *= -1;
        }
    }
}
// End Code
```

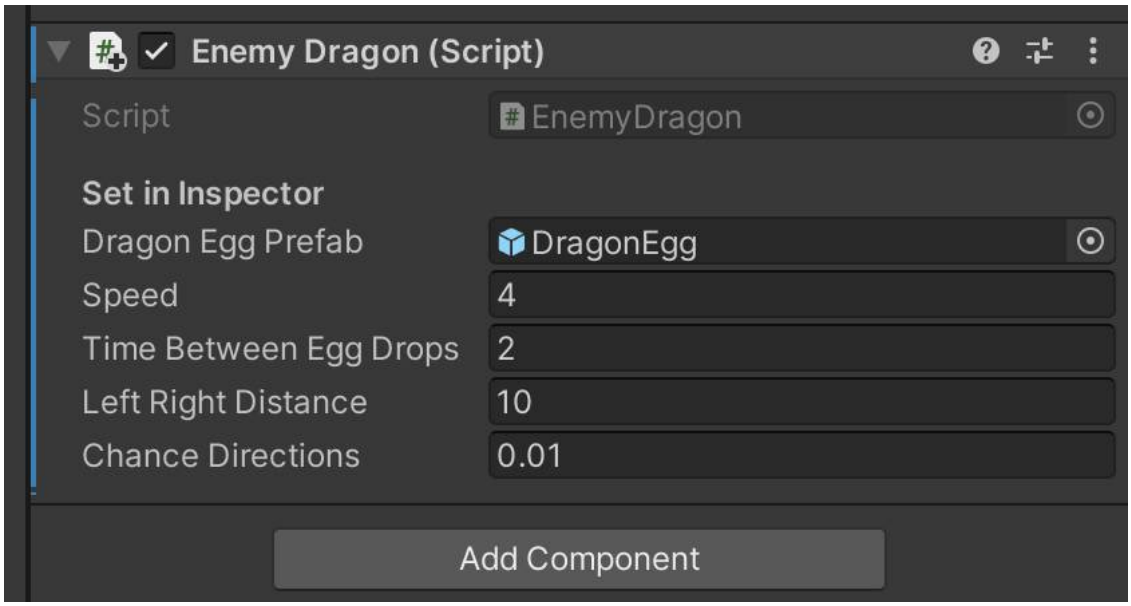
Так как листинг достаточно длинный, то для удобства на скриншоте приводится лишь его нижняя часть, включая уже созданный ранее метод Update и новый метод FixedUpdate.

EnemyDragon.cs

EnemyDragon ▸ No selection

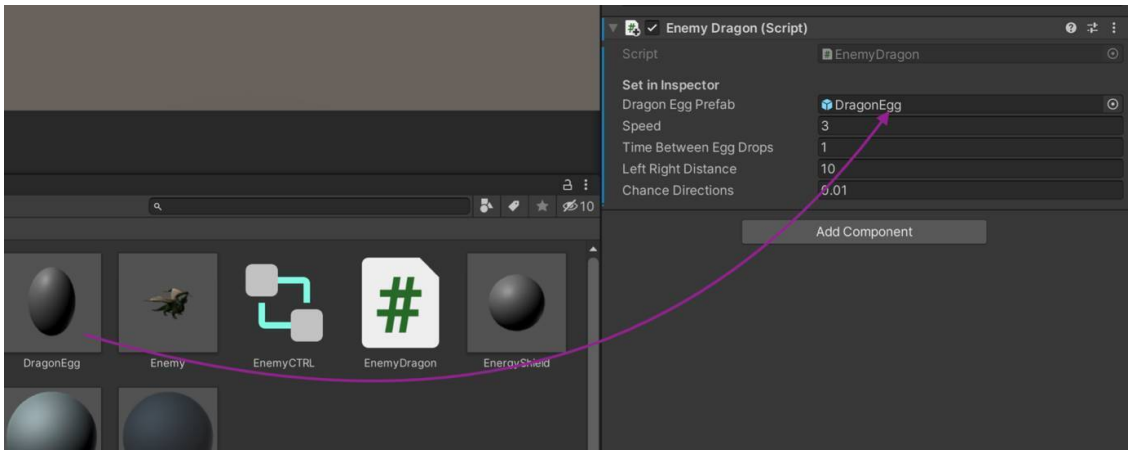
```
17
18          void Update()
19          {
20              Vector3 pos = transform.position;
21              pos.x += speed * Time.deltaTime;
22              transform.position = pos;
23
24              if (pos.x < -leftRightDistance)
25              {
26                  speed = Mathf.Abs(speed);
27              }
28              else if (pos.x > leftRightDistance)
29              {
30                  speed = -Mathf.Abs(speed);
31              }
32          }
33
34          private void FixedUpdate()
35          {
36              if (Random.value < chanceDirections)
37              {
38                  speed *= -1;
39              }
40          }
41      }
```

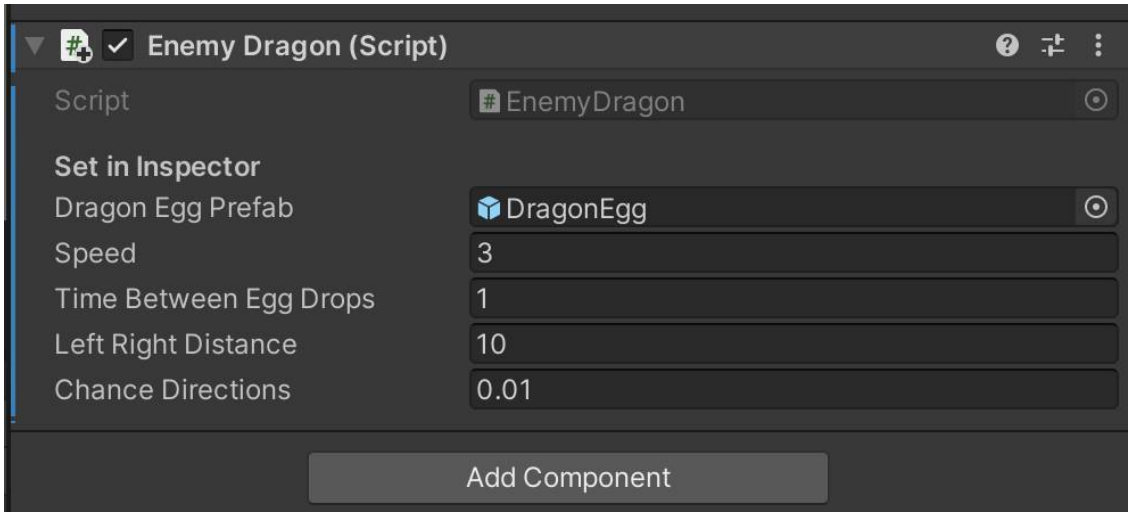
10. Подберите значение переменной chanceDirections таким образом, чтобы объект Enemy случайно менял направление в среднем 1 раз в секунду. Подобрать подходящее значение поможет формула: 50 вызовов FixedUpdate в секунду * вероятность = среднее число срабатываний в секунду. Напомним, что подбирать подходящие значения можно в окне Inspector (сразу после сохранения скрипта). Например, у нас получились следующие значения:



11. Запустите игру, нажав Run. Проверьте корректность срабатывания вероятности изменения направления движения Enemy.

12. Далее мы реализуем сбрасывание объектов DragonEgg из дракона Enemy. Для этого выберите игровой объект Enemy и в окне Inspector найдите поле для добавления префаба DragonEgg.prefab, и добавьте в него префаб игрового объекта DragonEgg. Это можно сделать также, как мы это делали ранее – перетаскиванием, либо нажав мишень и выбрать нужный из доступных префабов:





13. После того как объект подключен, добавим необходимые строки кода для генерации яблок. Для этого нам понадобится создать метод DropEgg() и команду вызова этого метода внутри Start(). Метод DropEgg будет создавать экземпляр драконьего яйца DragonEgg в точке нахождения дракона Enemy. Из метода Start будет запускаться генерация объекта-яйцо через заданное количество секунд:

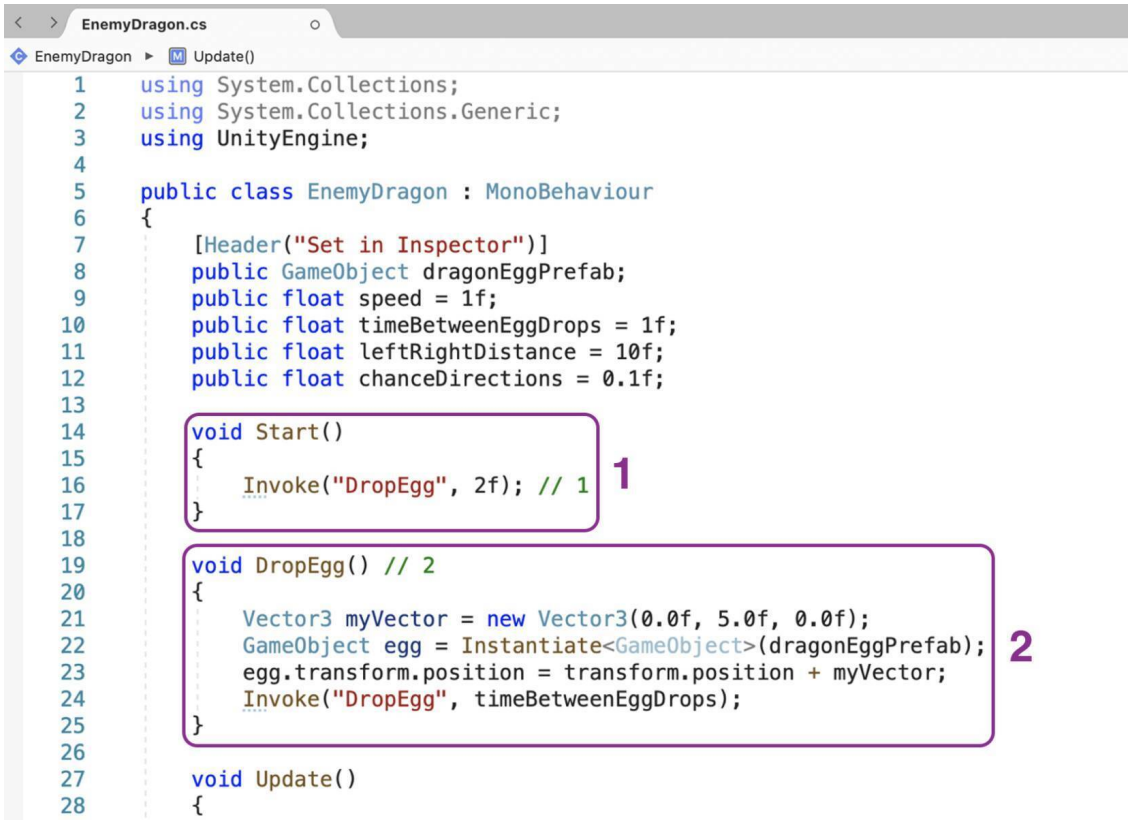
// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
```

```
        public GameObject dragonEggPrefab;
        public float speed = 1f;
        public float timeBetweenEggDrops = 1f;
        public float leftRightDistance = 10f;
        public float chanceDirections = 0.1f;
        void Start()
        {
            Invoke("DropEgg", 2f); // 1
        }
        void DropEgg() // 2
        {
            Vector3 myVector = new Vector3(0.0f, 5.0f, 0.0f);
                                          GameObject egg =
Instantiate<GameObject>(dragonEggPrefab);
           egg.transform.position = transform.position + myVector;
            Invoke("DropEgg", timeBetweenEggDrops);
        }
        void Update()
        {
            Vector3 pos = transform.position;
            pos.x += speed * Time.deltaTime;
            transform.position = pos;
            if (pos.x < -leftRightDistance)
            {
                speed = Mathf.Abs(speed);
            }
            else if (pos.x > leftRightDistance)
            {
                speed = -Mathf.Abs(speed);
            }
        }
        private void FixedUpdate()
        {
            if (Random.value < chanceDirections)
            {
                    speed *= -1;
            }
        }
    }
```

// End Code

На скриншоте листинга показаны первые строки, включающие новые команды внутри методов Start() и DropEgg().




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

public class EnemyDragon : MonoBehaviour
{
    [Header("Set in Inspector")]
    public GameObject dragonEggPrefab;
    public float speed = 1f;
    public float timeBetweenEggDrops = 1f;
    public float leftRightDistance = 10f;
    public float chanceDirections = 0.1f;

    void Start()
    {
        Invoke("DropEgg", 2f); // 1
    }

    void DropEgg() // 2
    {
        Vector3 myVector = new Vector3(0.0f, 5.0f, 0.0f);
        GameObject egg = Instantiate<GameObject>(dragonEggPrefab);
        egg.transform.position = transform.position + myVector;
        Invoke("DropEgg", timeBetweenEggDrops);
    }

    void Update()
    {
```

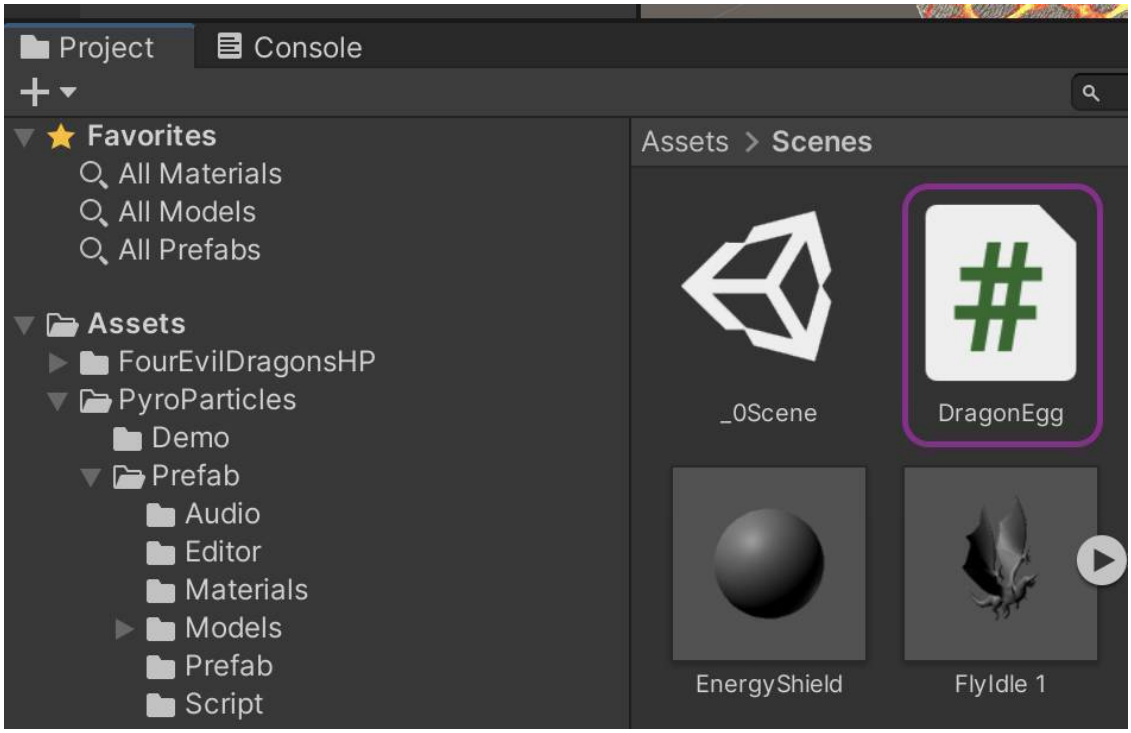
– функция Invoke в методе Start() выше вызывает функцию, заданную именем через указанное число секунд. В данном случае – вызывается функция DropEgg(), с параметром 2f, т.е. с ожиданием 2 секунды перед каждым новым вызовом. Таким образом, мы можем установить наиболее подходящие значения для частоты генерации объектов. Можете самостоятельно подобрать наиболее подходящее значение.

– функция DropEgg() создает экземпляр GameObject с именем dragonEggPrefab и присваивает его переменной egg. Далее изменяется положение egg, и функция вызывает саму себя снова через интервалы времени, равные timeBetweenEggDrops.

14. Теперь если нажать кнопку Play, то движущийся по экрану дракон будет сбрасывать драконье яйцо через равные интервалы времени.

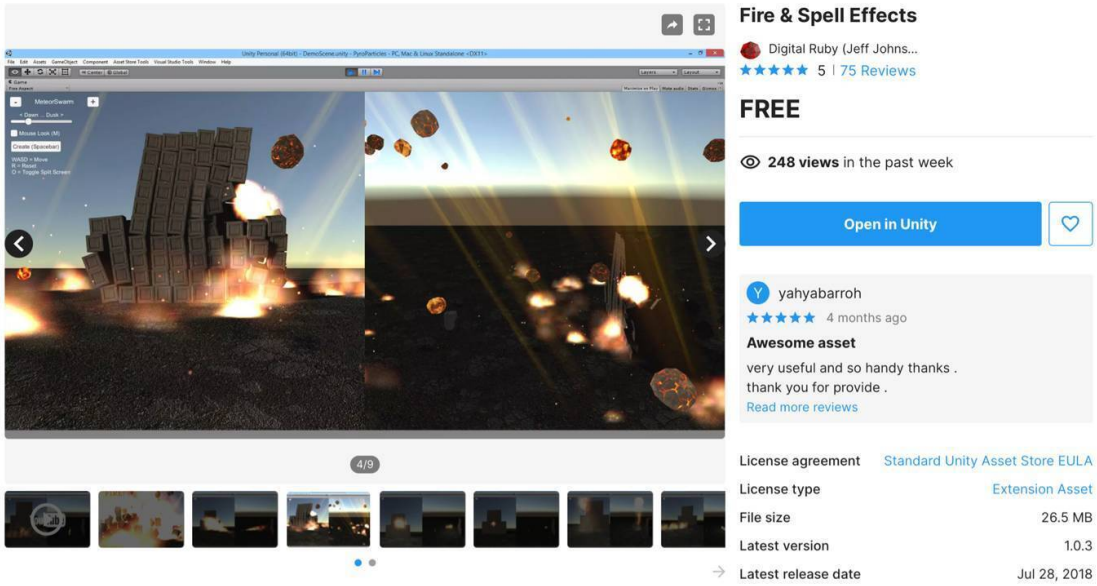
3.2 Скрипт-файл DragonEgg

В этом разделе мы опишем работу драконьего яйца. Создайте сценарий DragonEgg.cs:

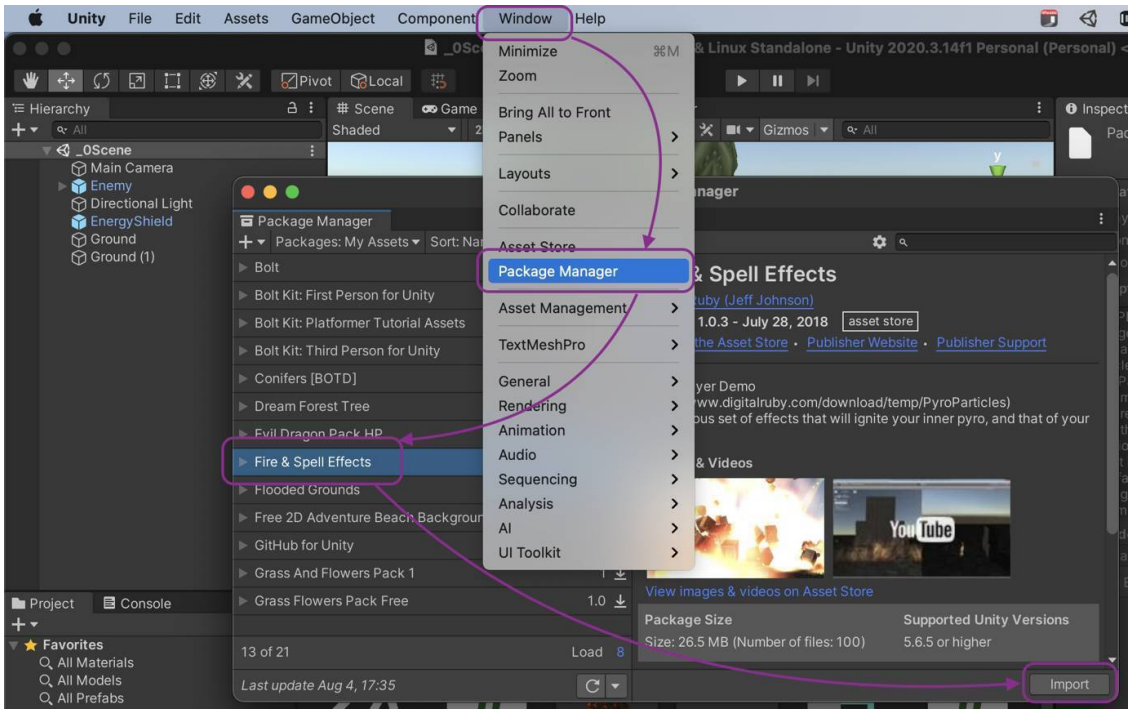


Перед тем как открыть файл и перейти к программированию, давайте несколько изменим оформление нашей сцены. В частности, добавим несколько игровых объектов и элементов визуального оформления. Мы сделаем это для того, чтобы нам стало более наглядно работать, к тому же в этом разделе мы добавим спецэффекты, а работать с ними конечно же эффективней, когда можно визуально оценить их работу.

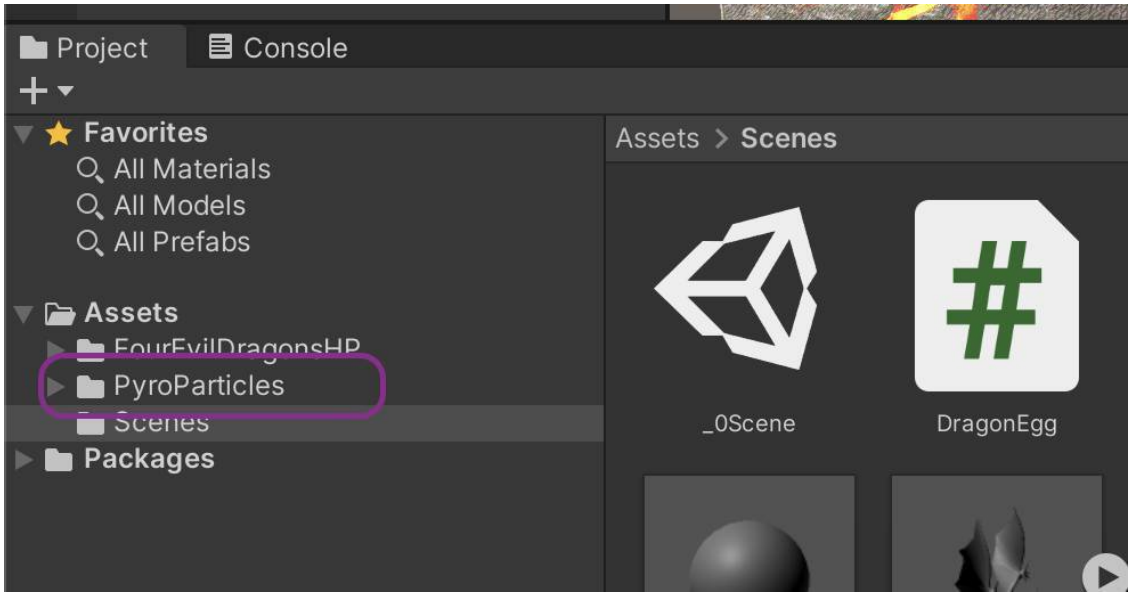
1. Дополнительные визуальные эффекты мы возьмем из уже знакомого Asset Store. Зайдите в assetstore.unity.com и используя строку поиска найдите Asset с именем Fire & Spell Effects. Добавьте его в свой профиль и вернитесь в среду разработки Unity.



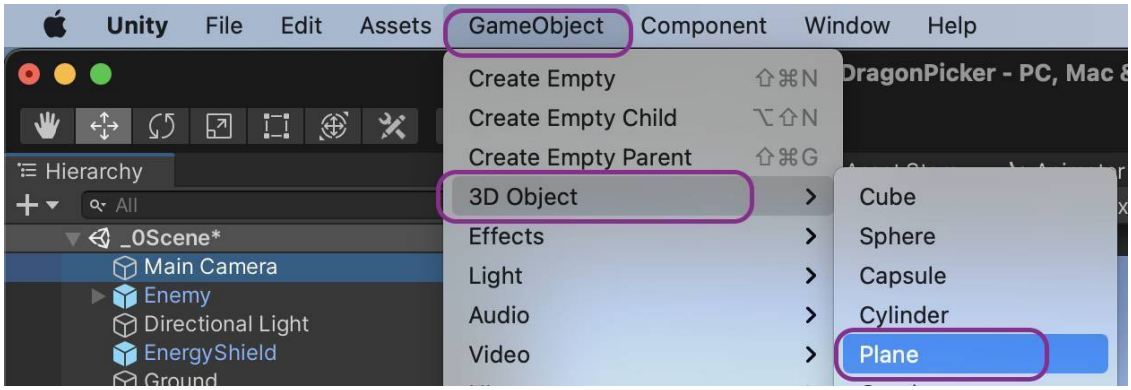
2. В среде разработки Unity откройте Window – Package Manager, в списке Package: My Assets найдите добавленный ассет с именем Fire & Spell Effects (если вы его не видите возможно потребуется обновить список, нажмите Refresh в нижней части окна Package Manager) и нажмите Download и далее Import. Описанная последовательность действий изображена на скриншоте ниже:



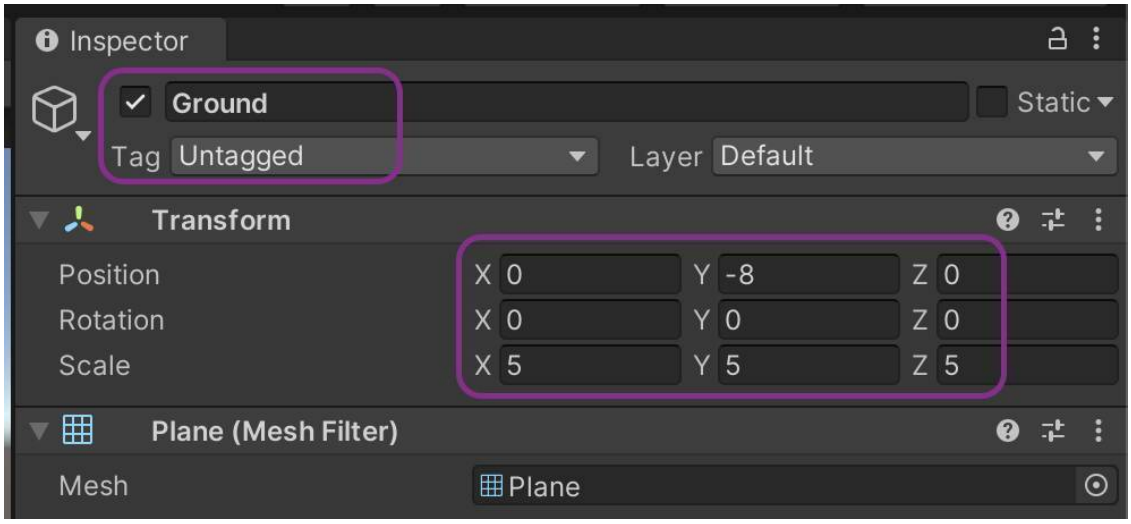
3. После того как импорт будет завершен, в папке Assets появится папка с набором текстур и эффектов:



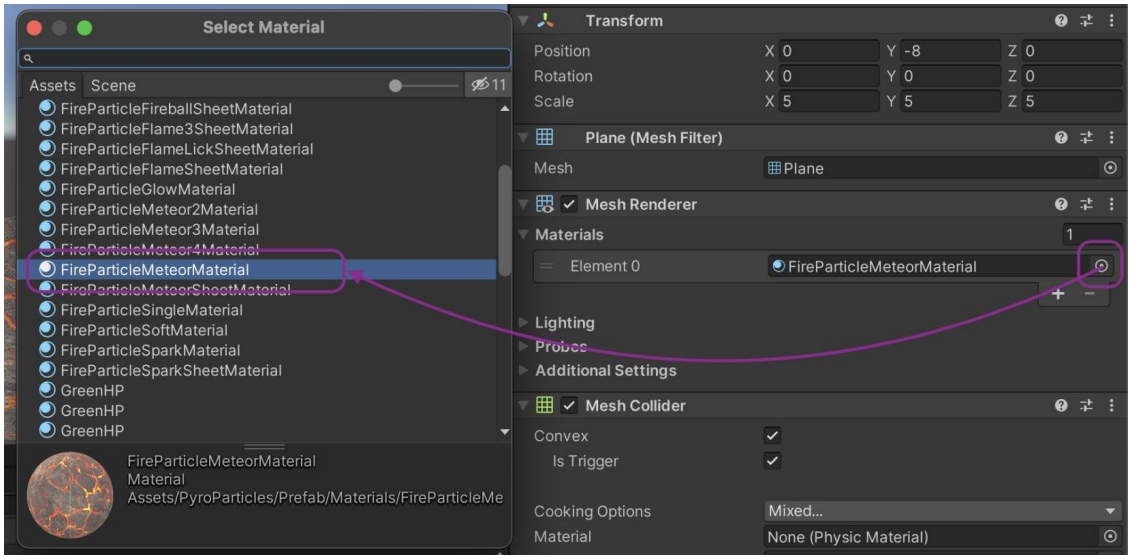
4. Далее мы добавим плоскость Plane, которая будет играть роль поверхности земли. Также мы оформим ее в виде раскаленной лавы, падая на которую драконьи яйца будут взрываться. Чтобы добавить плоскость на сцену, в верхнем меню выберите Windows – GameObject – Plane:



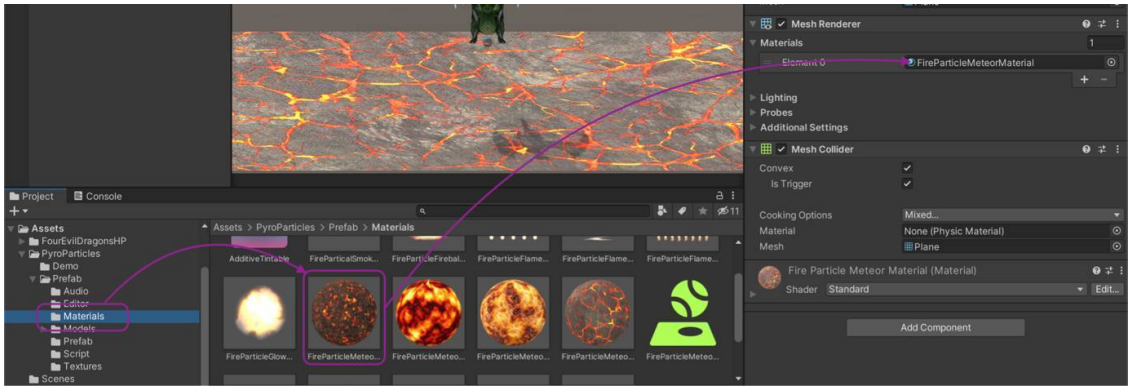
5. Переименуйте объект Plane в Ground. Установите размеры объекта Ground:
– Position: 0, -8, 0;
– Rotation: 0, 0, 0;
– Scale: 5, 5, 5;



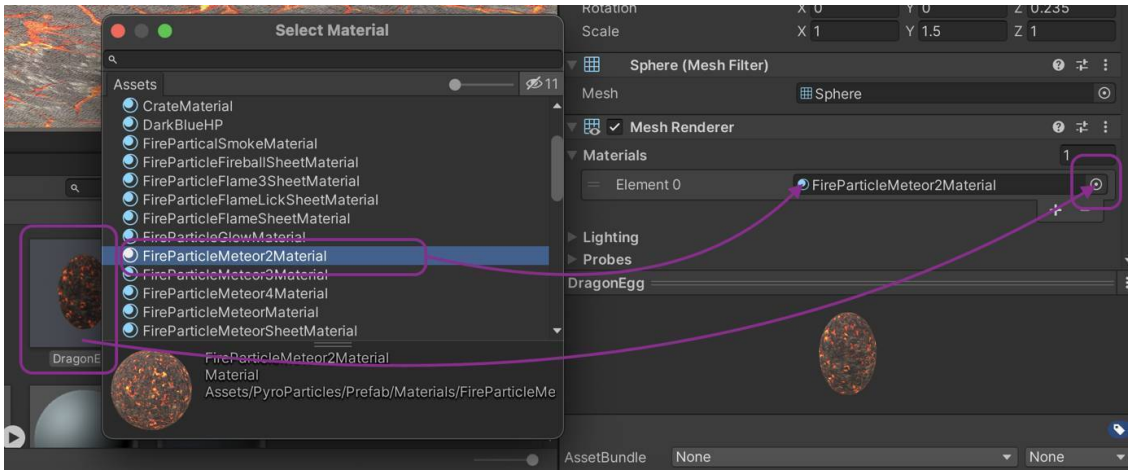
6. Изменим визуальное оформление элемента Ground. В окне Inspector (в правой части) разверните компонент Mesh Renderer (), нажмите на “мишень” и из появившегося меню Select Material выберите материал FireParticleMeteorMaterial.



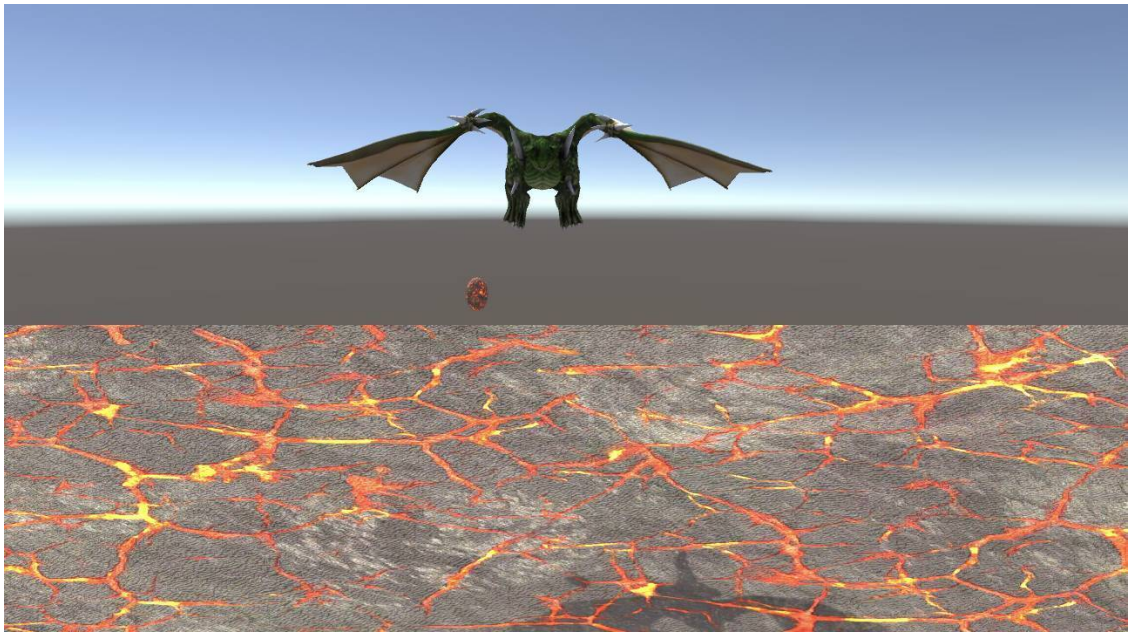
7. В качестве альтернативного варианта наложения текстуры на объект, можете найти нужную текстуру в папке скачанного ассет-пака и перетащить FireParticleMeteorMaterial в поле выбора материала, напротив надписи Element 0, как показано на рисунке ниже:



8. Текстуры из скачанного ассет-пака можно использовать и для других объектов. Например, вместо созданного вручную материала драконьего яйца Mat_Egg, можно также использовать готовую текстуру из Asset Store. Для этого выберите префаб DragonEgg из папки Scene и внутри компонента MeshRenderer выберите подходящую текстуру, например FireParticleMeteor2Material:



9. Теперь можете запустить сцену и проверить ее работу. Если вы выполнили все пункты выше, то сцена должно выглядеть так, как изображено на рисунке ниже:



10. Откройте сценарий DragonEgg.cs, подключенный к игровому объекту DragonEgg.prefab. Этот игровой объект дракон сбрасывает через равные временные интервалы. В скрипт-файл мы добавим следующий функционал:

– При столкновении с плоскостью Ground мы создадим эффект взрыва яйца. А именно, будет запускаться Particle System – это компонент, который может имитировать генерацию частиц разного вида (используется при добавлении взрывов, эффектов дыма, молний, тумана и т. д.). В нашем случае мы будем использовать Particle System для имитации взрыва.

– Генерируемые объекты DragonEgg все время существуют на сцене после создания. Со временем их количество будет увеличиваться, поэтому будет правильнее удалять объекты DragonEgg после того, как они будут уходить за пределы сцены.

11. Настройку визуальных эффектов мы можем разделить на два этапа:

– написание кода в скрипт-файл в DragonEgg.cs;

– настройка свойств Particle Syste, в окне Inspector элемента DragonEgg.prefab.

12. Приступим к написанию кода. Для этого добавьте в скрипт-файл DragonEgg.cs следующие строки кода:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DragonEgg : MonoBehaviour
{
    public static float bottomY = -30f;
    void Start()
    {
    }
    private void OnTriggerEnter(Collider other)
    {
        ParticleSystem ps = GetComponent<ParticleSystem>();
        var em = ps.emission;
        em.enabled = true;
```

```
            Renderer rend;
            rend = GetComponent<Renderer>();
            rend.enabled = false;
        }
        void Update()
        {
            if (transform.position.y < bottomY)
            {
                Destroy(this.gameObject);
            }
        }
}
```

// End Code

Как и ранее, скриншот листинга покажем ниже.

EnemyDragon.cs | DragonEgg.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class DragonEgg : MonoBehaviour
6   {
7       public static float bottomY = -30f;
8
9       void Start()
10      {
11      }
12
13      private void OnTriggerEnter(Collider other)
14      {
15          ParticleSystem ps = GetComponent<ParticleSystem>();
16          var em = ps.emission;
17          em.enabled = true;
18
19          Renderer rend;
20          rend = GetComponent<Renderer>();
21          rend.enabled = false;
22      }
23
24      void Update()
25      {
26          if (transform.position.y < bottomY)
27          {
28              Destroy(this.gameObject);
29          }
30      }
31  }
```

В скрипт файле основное функциональные блоки выполняют следующие действия:

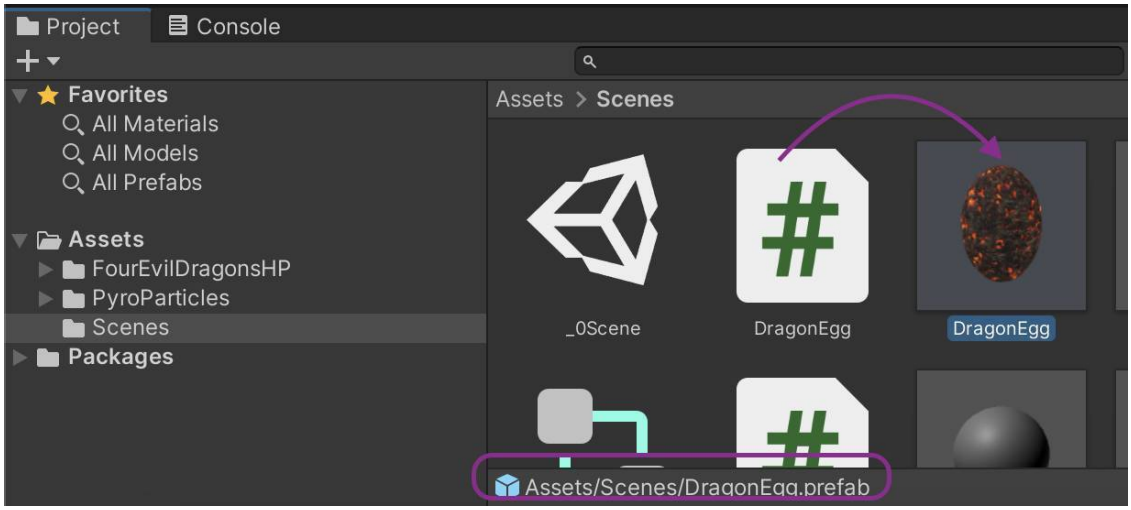
– создается переменная bottomY, которая будет указывать, на каком расстоянии Y нужно будет удалять объекты DragonEgg.

– По названию метода OnTriggerEnter (Collision Other) можно понять, что он начинает работу, когда происходит пересечение с объектом, который работает как Trigger. В нашем случае таким объектом-триггером будет выступать плоскость Plane. При срабатывании триггера

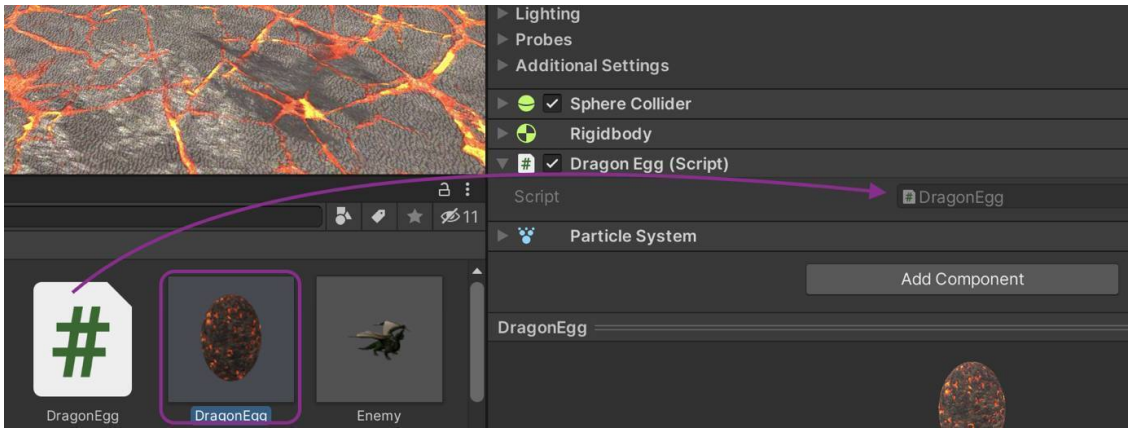
запускается проигрывание Particle System (em.enabled = true), и яйцо становится невидимым (rend.enabled = false).

– внутри метода Update происходит уничтожение объекта DragonEgg (Destroy(this.gameObject)), если яйцо падает ниже уровня -10f. Это позволяет избежать накопления ненужных объектов за пределами игровой сцены.

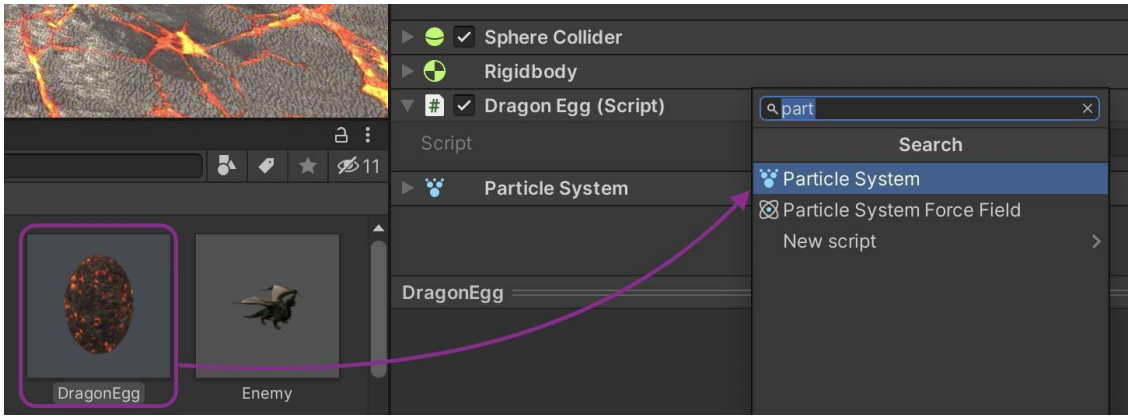
13. Сохраните сценарий DragonEgg.cs и подключите его к префабу (шаблону) DragonEgg.prefab в панели Project. Для этого вы можете просто перетащить файл DragonEgg.cs на префаб DragonEgg.prefab:



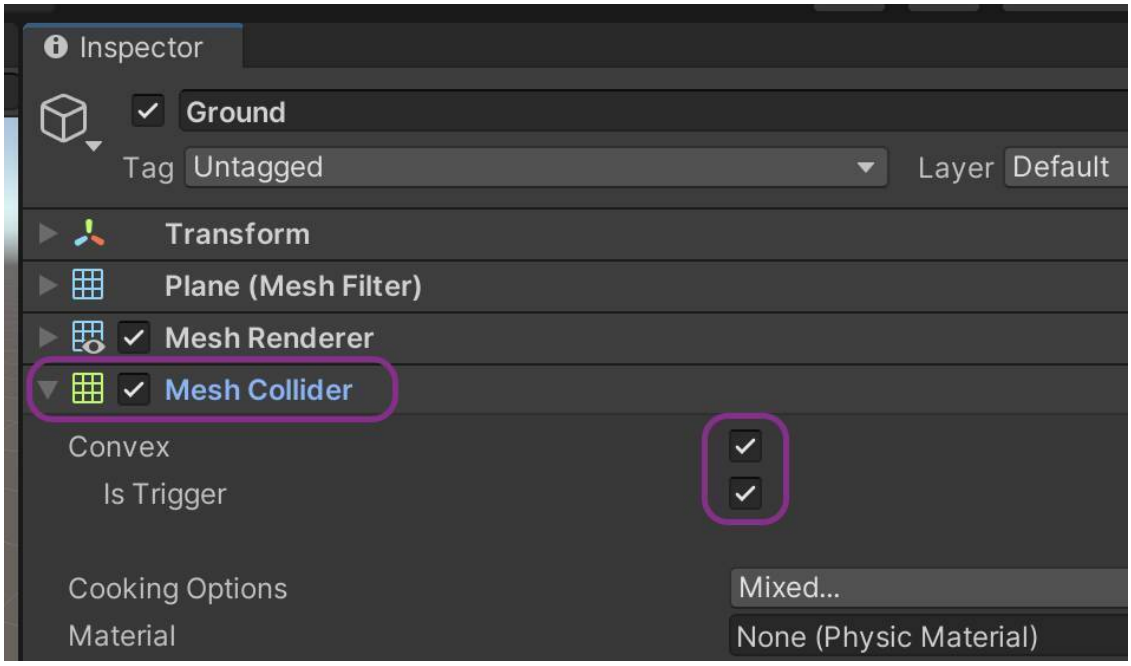
14. После этого можно убедиться, что в окне инспектора Inspector появится компонент с подключенным Script-файлом.



15. Выберите DragonEgg.prefab и добавьте компонент Particle System:

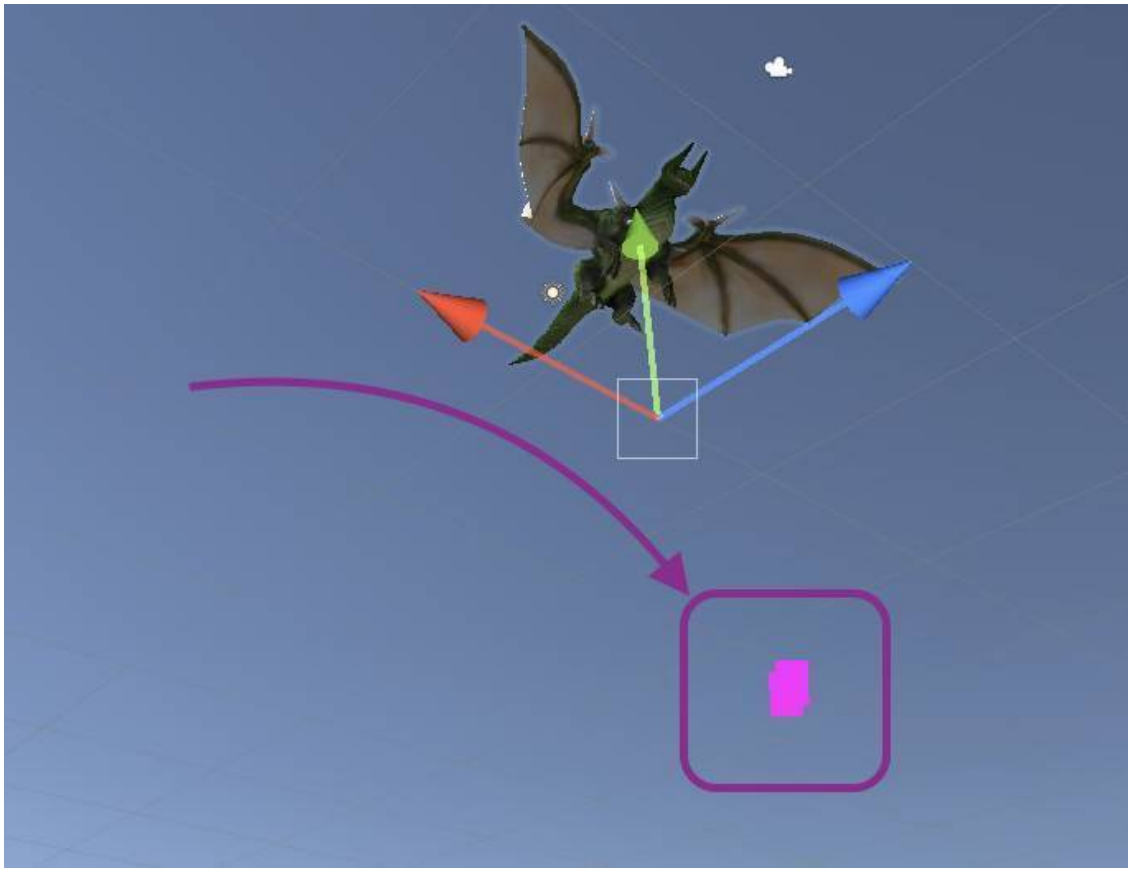


16. Теперь настроим объект Ground, чтобы он работал в качестве триггера. Сделать это можно достаточно просто, в окне Hierarchy выберите объект Ground и после этого в окне Inspector поставьте две галочки в компоненте Mesh Collider – Convex, Is Trigger:

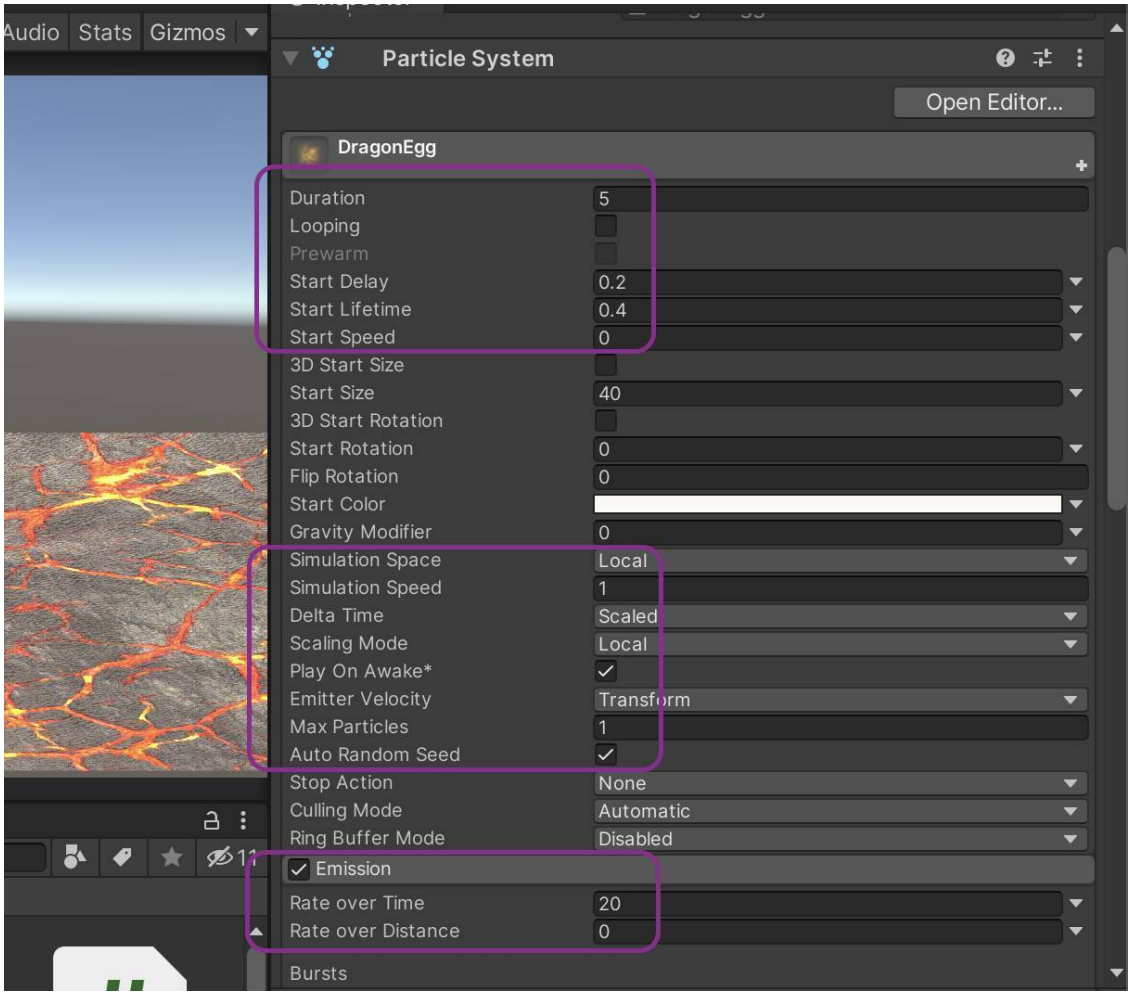


17. Убедитесь, что при нажатии кнопки Play игровые объекты DragonEgg летят вниз, проходят сквозь Ground с генерацией элементов Particle System и исчезают из иерархии объектов после того как достигнут нижнего края (заданного уровня -10).

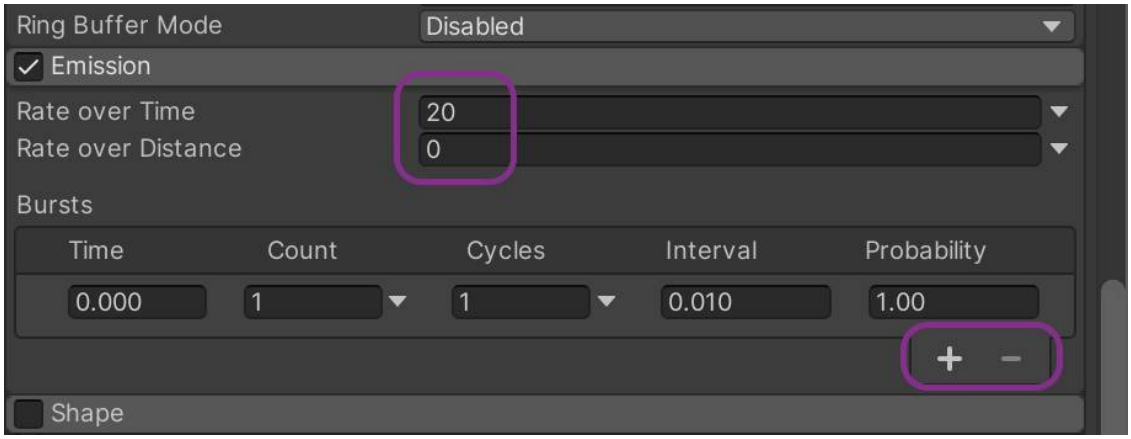
18. В данный момент вид элементов Particle System не настроен, поэтому они отображаются в стандартном виде:



19. Изменим настройки Particle System, чтобы они имели внешний вид, похожий на взрыв. Для этого выберите элемент DragonEgg, в окне Inspector разверните компонент Particle System. Он содержит большое количество дополнительных настроек, ниже выделены основные, на которые следует обратить внимание. Настройте компонент так, как показано на рисунке ниже:



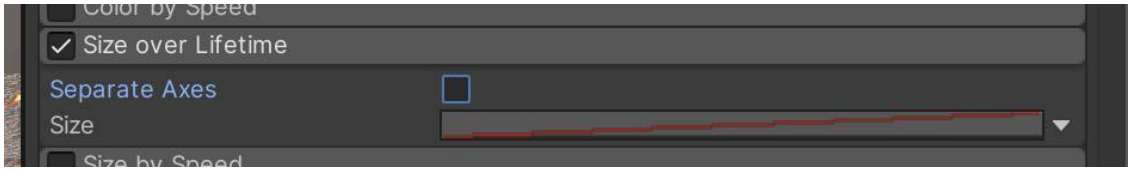
20. Поставьте галочку возле Emission и введите параметры, показанные на рисунке ниже (чтобы добавить новую линию с параметрами “всплесков” нажмите “+”).



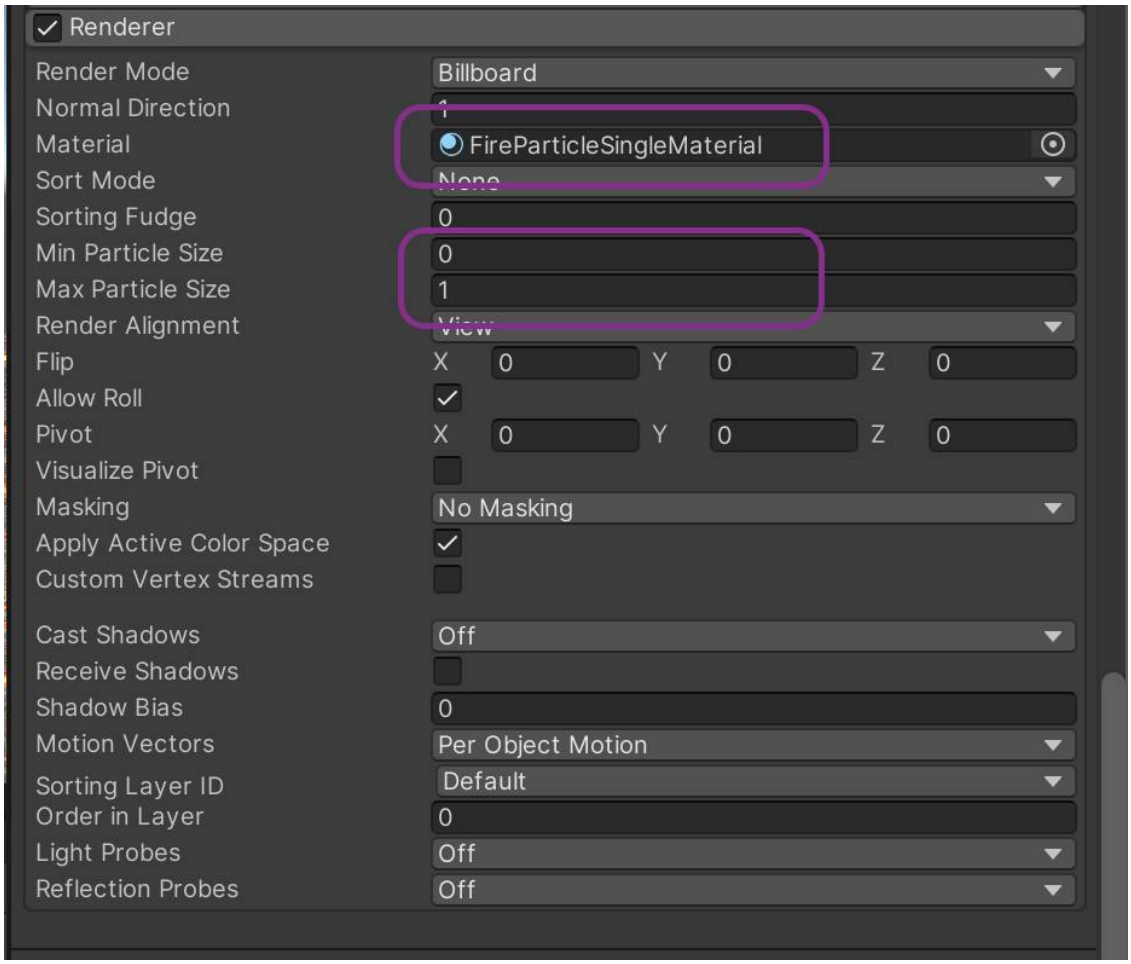
21. Теперь снимите галочку с Emission. Это делается для того, чтобы при запуске сцены этот параметр был выключен и объект не излучал Particles с эффектами взрыва.

22. Поставьте галочку напротив “Size over Lifetime”, и выберите вид кривой, которая растет от минимума слева до максимума справа. Эта кривая означает, что элемент Particle будет распространяться от центра в разные стороны, как это происходит во время взрыва. Если обра-

тите внимание, что существует несколько типовых видов кривых, которые позволяют создавать совершенно разные эффекты.



23. Последний элемент, с которым мы поработаем внутри Particle System – это Renderer, в нем назначается текстура частицы, которая генерируется при запуске. Выберем в качестве текстуры материал с названием FireParticleSingleMaterial, эта текстура также находится в скачанном ассет-паке в папке PyroParticles.

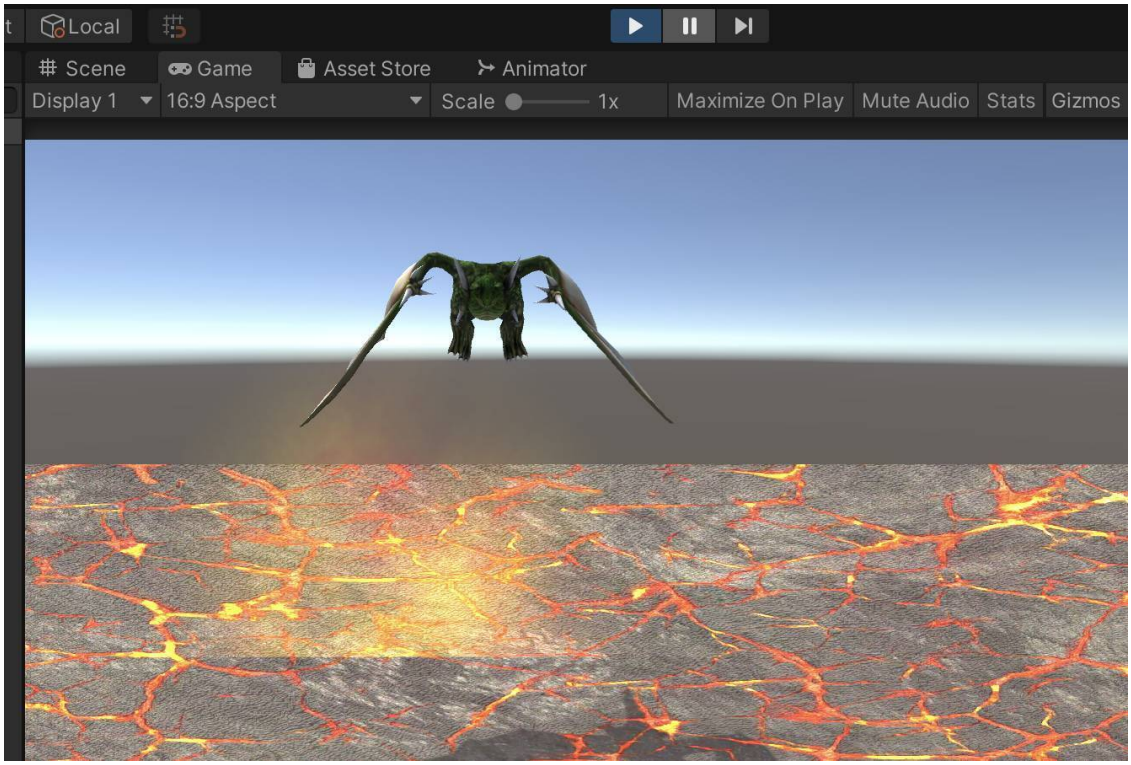


24. Теперь вы можете запустить сцену и проверить, что при столкновении драконьего яйца с плоскостью земли, создается эффект взрыва. Но если быть точным, то на самом деле происходит следующее:

– при столкновении объекта DragonEgg с Ground срабатывает триггер, после этого яйцо перестает отображаться (rend.enabled = false);

– запускается работа Particle System, имеющей визуальное оформление взрыва (em.enabled = true);

– объект DragonEgg уничтожается при падении ниже уровня y = -10f.

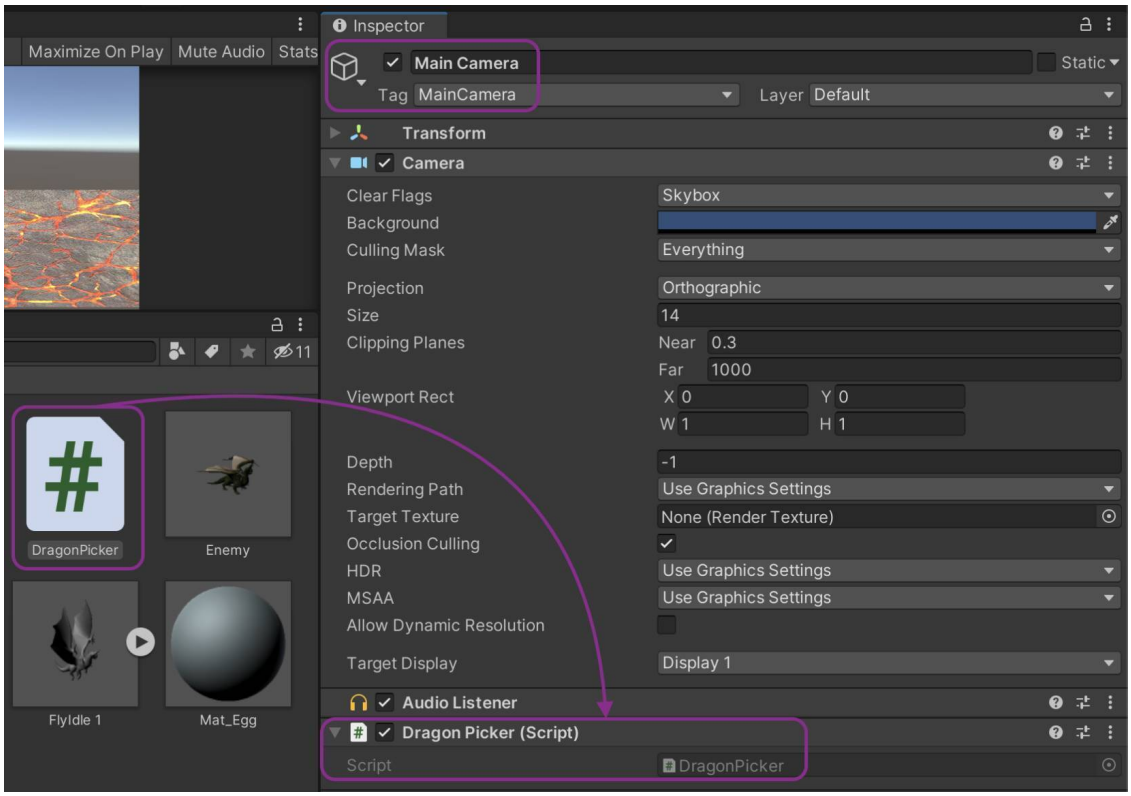


Таким образом, в пункте 3.2 мы проделали достаточно большую работу, улучшив визуальное оформление нашей игровой сцены. Также мы написали функционал объекта DragonEgg, научились создавать эффект взрыва, скрывать и удалять объекты при наступлении различных событий.

3.3 Скрипт-файл DragonPicker

Практически во всех играх существует сценарий, который управляет игрой в целом: запуском и перезапуском сцены, сохранением, загрузкой и т. д. В нашем примере мы создадим такой сценарий, который будет называться DragonPicker.cs. Подключать сценарии, управляющие игрой, следует к тем объектам, которые гарантированно присутствуют в любой сцене, например к Main Camera. Давайте это сделаем.

1. Создайте сценарий DragonPicker.cs и подключите его (можно так же, как и ранее – перетаскиванием) к элементу Main Camera в окне Hierarchy:



2. Помимо управления игрой в сценарии будет создаваться три экземпляра EnergyShield – главного персонажа, который будет ловить летящие вниз яйца дракона. Условно игрок будет окружен энергетическими шарами EnergyShield, которые будут символизировать количество жизней.

3. Напишите код, который будет создавать три экземпляра шаблона EnergyShield, располагая их на экране внутри друг друга.

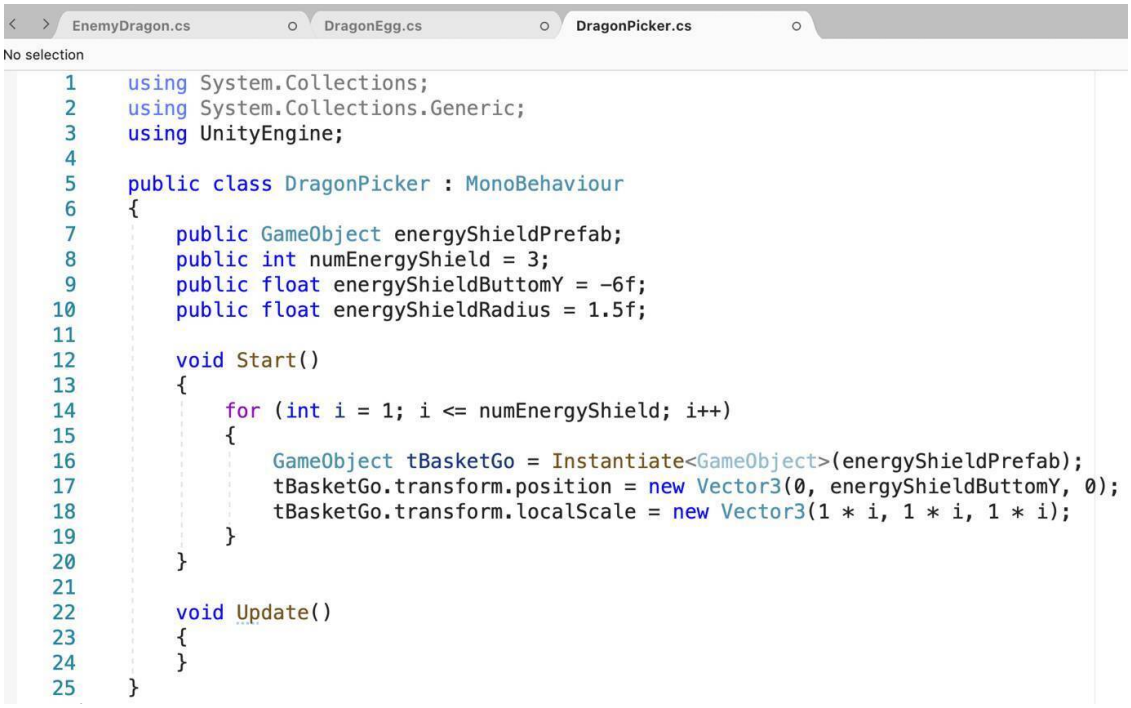
// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public classDragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
```

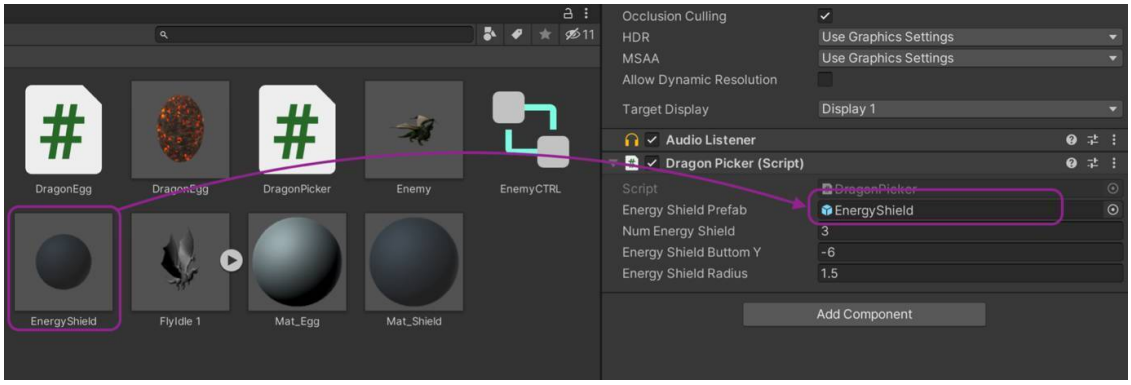
```
        public float energyShieldRadius = 1.5f;
        void Start()
        {
            for (int i = 1; i <= numEnergyShield; i++)
            {
                                        GameObject tBasketGo =
Instantiate<GameObject>(energyShieldPrefab);
                 tBasketGo.transform.position = new Vector3(0,
energyShieldButtomY, 0);
                tBasketGo.transform.localScale = new Vector3(1
* i, 1 * i, 1 * i);
            }
        }
        void Update()
        {
        }
    }
```

// End Code

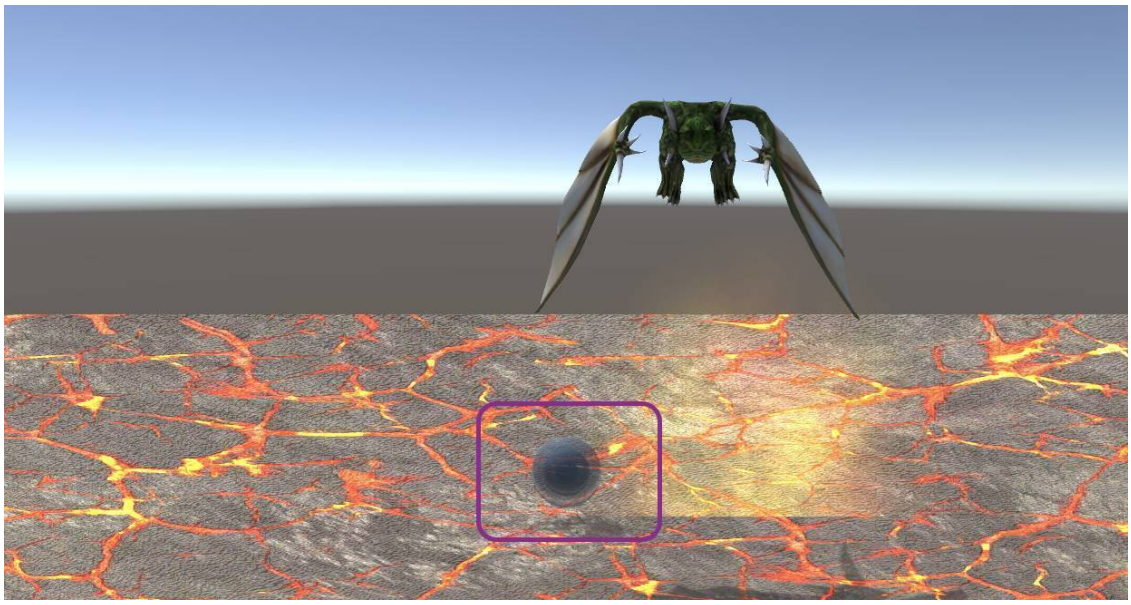
Скриншот листинга приведем ниже.



4. Чтобы подключить EnergyShield к сценарию, выберите Main Camera, в Inspector для поля Energy Shield Prefab установите шаблон корзины EnergyShield:



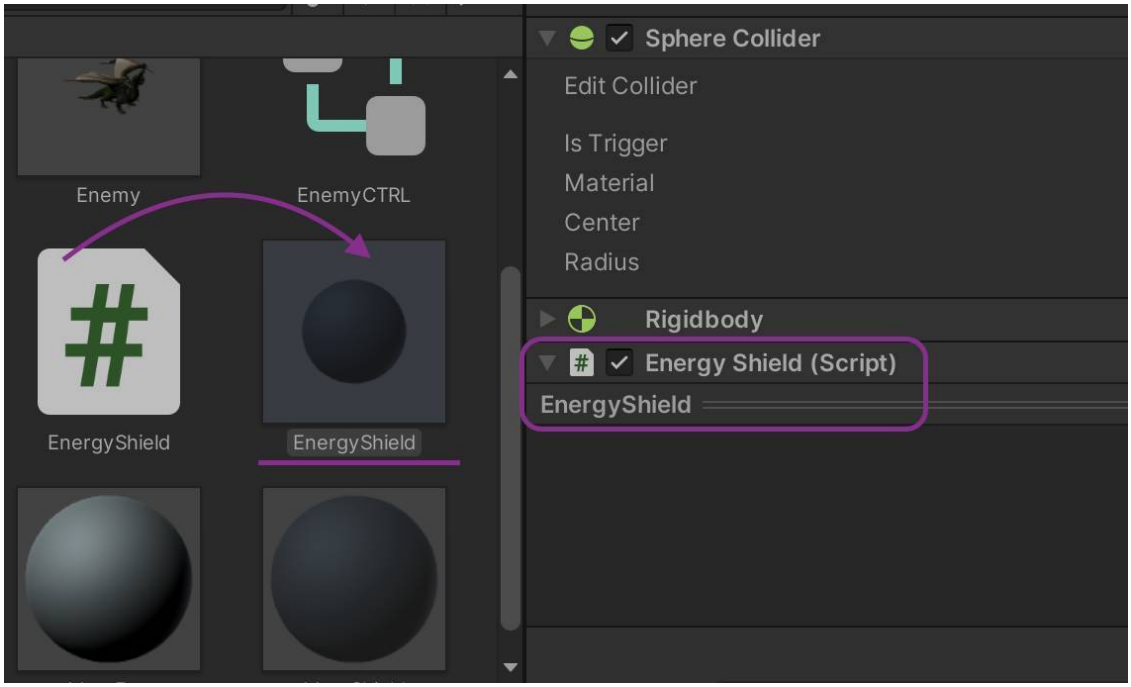
5. Нажмите кнопку Play и убедитесь, что что написанный код создает три экземпляра EnergyShield в нижней части экрана, как показано на рисунке ниже:



3.4 Скрипт-файл EnergyShield

1. В этом пункте будет написан код для перемещения игрового объекта EnergySheild вслед за указателем мыши. При желании вы можете изменить “интерфейс ввода”, адаптированный для клавиатуры, Touch Screen мобильных устройств (смартфонов, планшетов) и изменить тем самым механику игры.

2. Подключите сценарий EnergyShield.cs к шаблону Energy Shield Prefab:



3. Код, приведенный ниже, будет отвечать за перемещение энергетических щитов Energy Shield вслед за курсором мыши:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public classEnergyShield : MonoBehaviour
{
    void Update()
    {
        Vector3 mousePos2D = Input.mousePosition;
        mousePos2D.z = -Camera.main.transform.position.z;
                                    Vector3    mousePos3D    =
Camera.main.ScreenToWorldPoint(mousePos2D);
        Vector3 pos = this.transform.position;
        pos.x = mousePos3D.x;
        this.transform.position = pos;
    }
}
```

// End Code

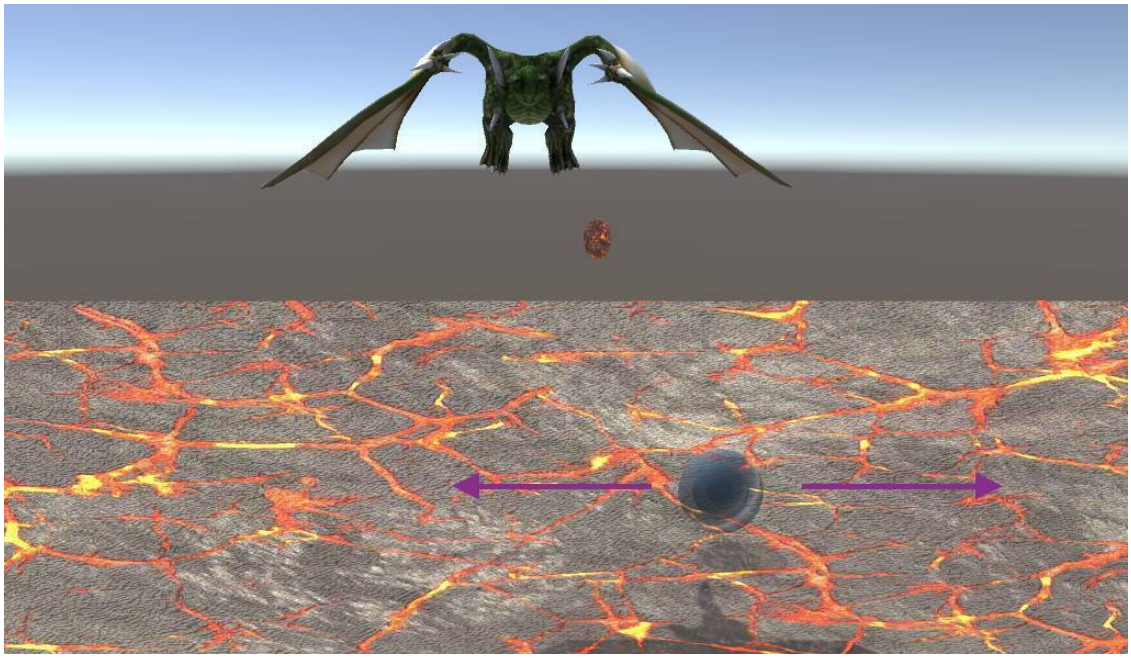
Скриншот листинга показан ниже.

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnergyShield : MonoBehaviour
6   {
7
8       void Update()
9       {
10          Vector3 mousePos2D = Input.mousePosition;
11          mousePos2D.z = -Camera.main.transform.position.z;
12          Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
13          Vector3 pos = this.transform.position;
14          pos.x = mousePos3D.x;
15          this.transform.position = pos;
16      }
17  }
18
```

4. Проверьте, что при нажатии на кнопку Play, энергетические шары перемещаются вслед за указателем мыши:



5. Реализовать ловлю объектов DragonEgg можно с помощью метода OnCollisionEnter. Добавьте метод ниже метода Update():

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public classEnergyShield : MonoBehaviour
{
    void Update()
```

```
        {
            Vector3 mousePos2D = Input.mousePosition;
            mousePos2D.z = -Camera.main.transform.position.z;
                                        Vector3    mousePos3D    =
Camera.main.ScreenToWorldPoint(mousePos2D);
            Vector3 pos = this.transform.position;
            pos.x = mousePos3D.x;
            this.transform.position = pos;
        }
        private void OnCollisionEnter(Collision coll)
        {
            GameObject Collided = coll.gameObject;
            if (Collided.tag == "DragonEgg")
            {
                Destroy(Collided);
            }
        }
    }
```

// End Code

Скриншот листинга приведен ниже.

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

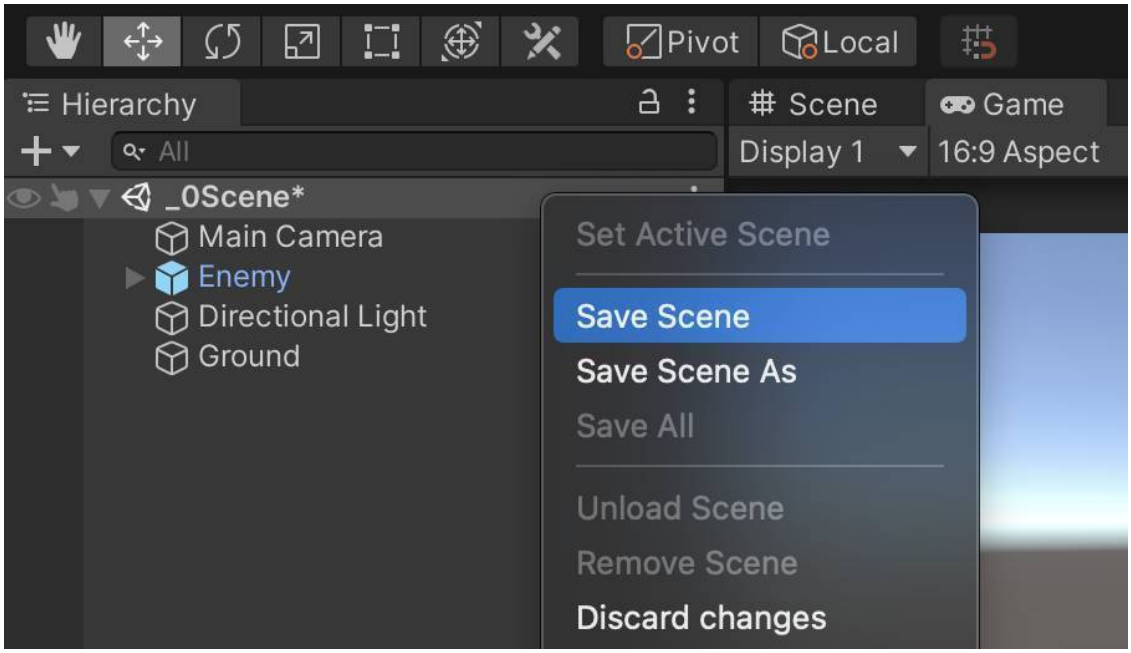
No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class EnergyShield : MonoBehaviour
6   {
7       void Update()
8       {
9           Vector3 mousePos2D = Input.mousePosition;
10          mousePos2D.z = -Camera.main.transform.position.z;
11          Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
12          Vector3 pos = this.transform.position;
13          pos.x = mousePos3D.x;
14          this.transform.position = pos;
15      }
16
17      private void OnCollisionEnter(Collision coll)
18      {
19          GameObject Collided = coll.gameObject;
20          if (Collided.tag == "DragonEgg")
21          {
22              Destroy(Collided);
23          }
24      }
25  }
```

– метод OnCollisionEnter срабатывает (вызывается) каждый раз, когда с коллайдером EnergyShield что-то сталкивается. В локальную переменную Collided записывается ссылка на игровой объект (coll.gameObject), столкнувшийся с энергетическим шаром. Далее следует проверка, если этот объект имеет тег DragonEgg (см. раздел 2.4, п.8), – то он уничтожается – Destroy(Collided);

6. Сохраните сценарий, запустите игру (Play) и проверьте что драконьи яйца DragonEgg ловятся (точнее уничтожаются) при взаимодействии с энергетическими шарами EnergyShield.

7. Сохраните сцену _Scene_0, см. рисунок ниже:


Pivot
Local
Hierarchy
All
_0Scene*
Main Camera
Enemy
Directional Light
Ground
Scene
Game
Display 1
16:9 Aspect
Set Active Scene
Save Scene
Save Scene As
Save All
Unload Scene
Remove Scene
Discard changes

Выводы

После выполнения пунктов в разделе 3, игра стала похожа на классическую и вполне работоспособную игру с ловлей объектов. На данном этапе это вполне реализованный функционал, который позволяет при желании модифицировать некоторые правила и аспекты игры, подобрать оптимальный уровень сложности и т. д.

Существует большое количество игр с похожей на Dragon Picker механикой, возможно вы уже неоднократно увидели схожесть с тысячами подобных игр. Даже традиционный Flappy Bird можно получить из нашего Dragon Picker'a, перевернув сцену на 90 градусов, создав эффект полета движением заднего фона сцены, и изменив немного условие, при котором уничтожаться при пересечении с объектом будет главный персонаж, а не объект.

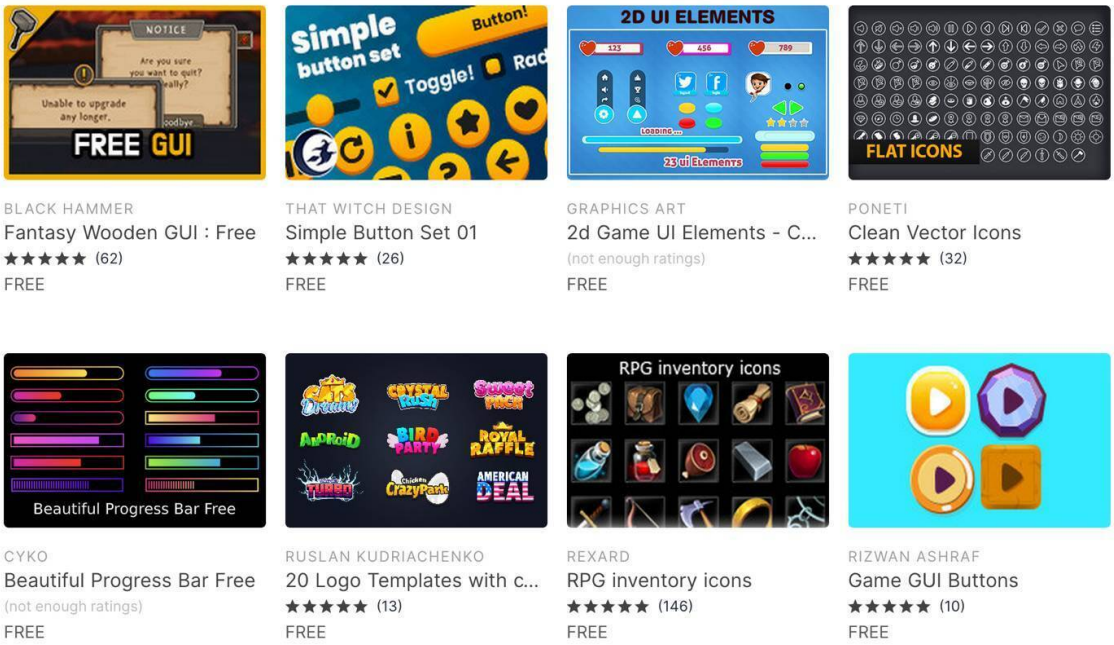
В качестве дополнительного задания можете самостоятельно подумать над тем, чтобы расширить функционал и правила игры Dragon Picker. Придуманные правила и функционал должны делать игру более интересной. Добавляйте только те элементы, которые вы бы смогли реализовать с незначительными временными затратами. Помните, что внесение даже самых маленьких дополнений в игру может кардинально изменить ее восприятие, но и в тоже время привести к неработоспособности или некорректной работе игры в самых непредвиденных местах.

Часть 4. Создание графического интерфейса пользователя

Введение

Graphic User Interface (GUI) или графический пользовательский интерфейс – это такой тип пользовательского интерфейса, который позволяет перемещаться по игре и выполнять действия с помощью визуальных индикаторов и графических значков.

В рамках Unity элементы GUI могут быть созданы самостоятельно с помощью любого графического редактора, либо скачаны с уже знакомого нам Asset Store. В последнем содержится довольно большое количество самых разнообразных скинов интерфейса под самую разнообразную тематику (для удобства поиска используйте вкладку Assets – GUI).

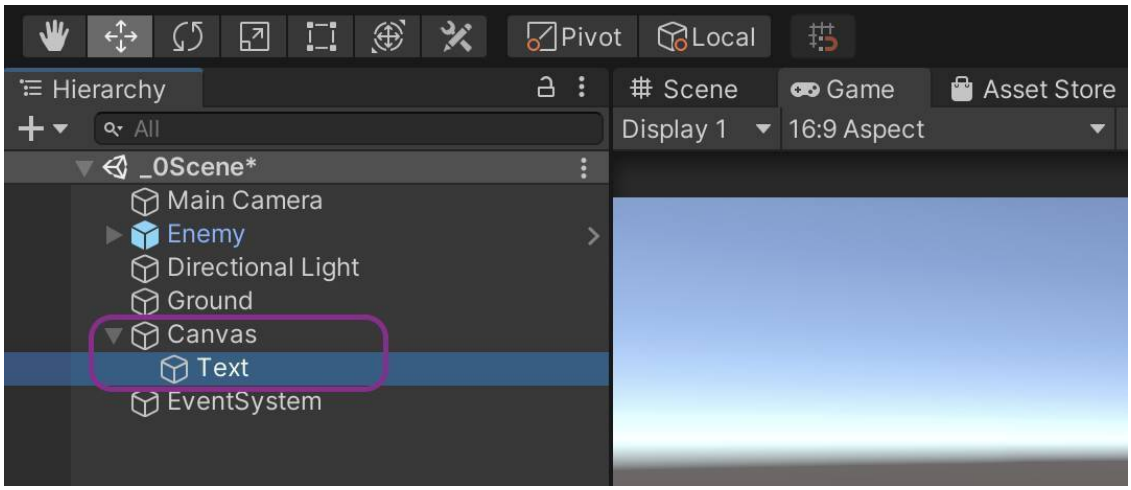


Графический интерфейс берет на себя достаточно большой функционал. Мало того, что без него невозможно будет удобно запустить, сохранить игру и выйти из нее, без графического интерфейса просто будет неинтересно играть. Ведь элементы GUI – это показатель здоровья, счетчики очков и достижений, инвентарь, умения и многое другое. Поэтому работа над графическим интерфейсом не менее важна чем и сама игра, ведь в итоге пользователь будет взаимодействовать с игрой через интерфейс, который вы для него создадите.

4.1 Создание счетчика очков

1. Задача счетчика очков заключается в том, чтобы дать пользователю обратную связь о результатах его действий или достижениях в игре.

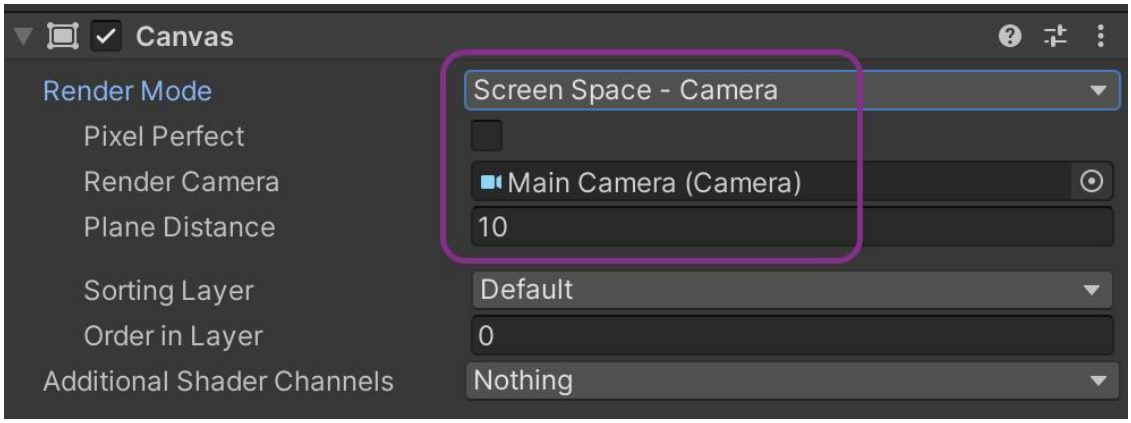
2. Дважды кликните по _0Scene в панели Project, и в меню Unity выберите пункт GameObject – UI – Text. После этого на сцену добавится элемент графического интерфейса пользователя. Обратите внимание, что в панели Hierarchy появится несколько объектов, один из них – Canvas:



3. Дважды кликните по Canvas, чтобы уменьшить его масштаб и увидеть весь объект целиком. Объект Canvas – это двумерный холст, на котором размещаются элементы графического интерфейса.

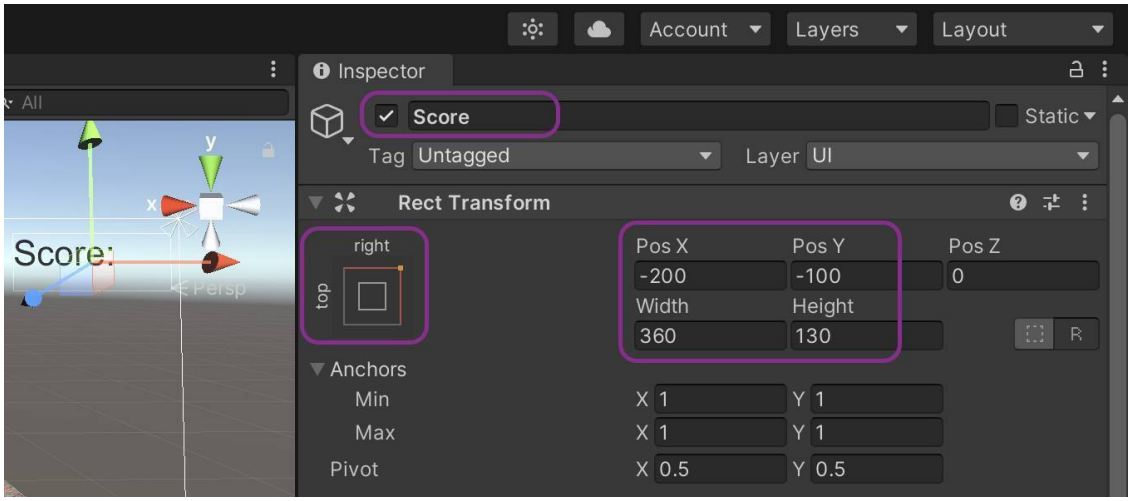
4. В списке дочерних объектов Canvas можно увидеть объект Text. Измените имя объекта на Score.

5. В окне Hierarchy выделите Canvas и в окне Inspector разверните компонент Canvas. Установите в поле Render Mode значение Screen Space – Camera, а в качестве Render Camera выбрана Main Camera (Camera). В поле Plane Distance установите значение равное 10.

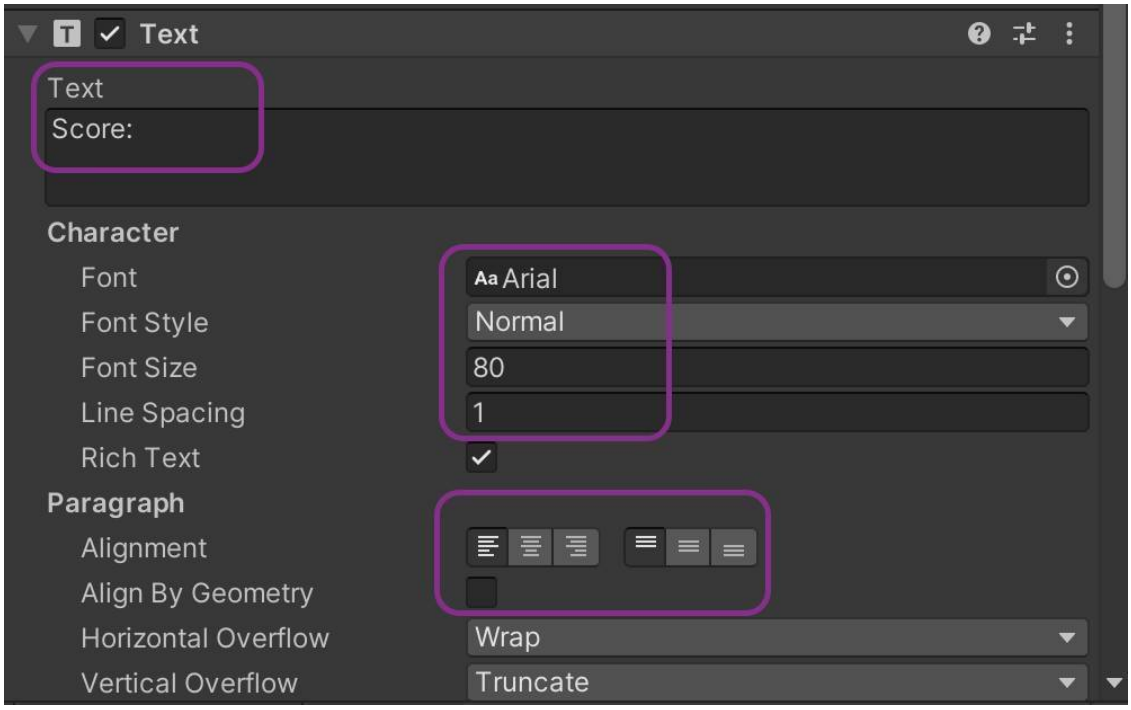


6. Настройте компонент RectTransform объекта Score так, как показано на рисунке ниже. Якоря Anchors нужно установить в положение top – right и задать значения отступа сверху

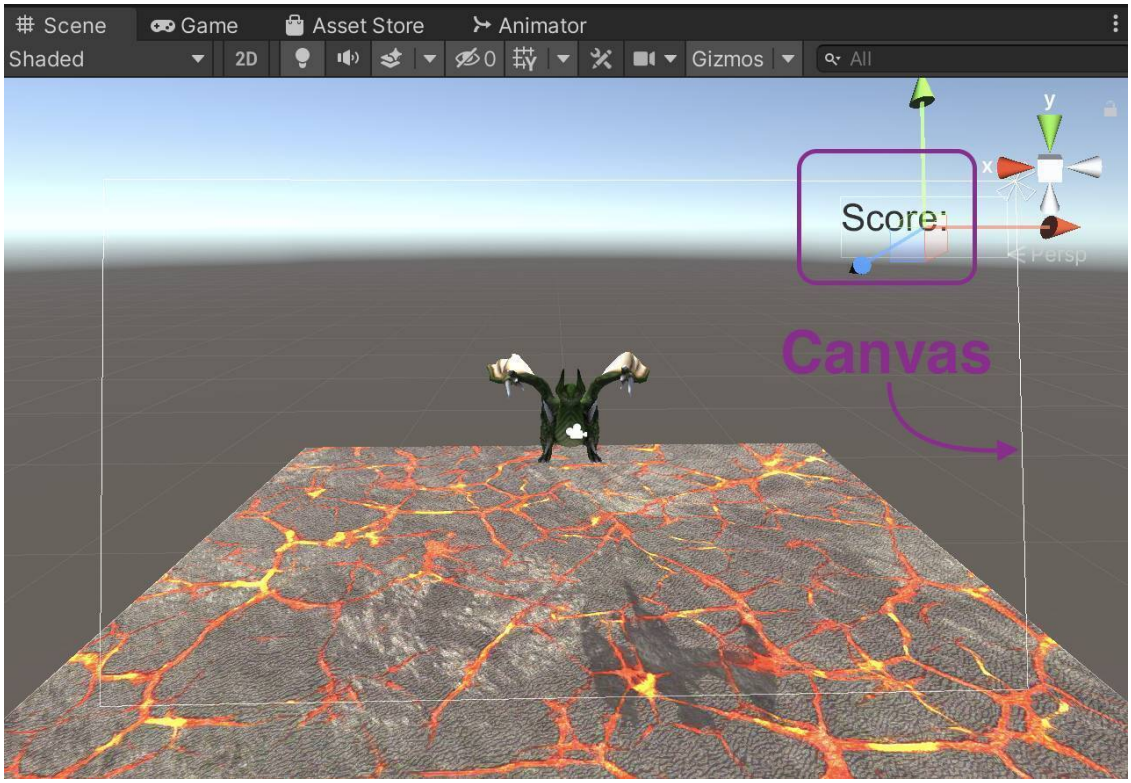
и справа. Таким образом, якорь как-бы указывается начальную точку, относительно которой нужно задавать координаты X и Y (Pos X, Pos Y).



7. Чуть ниже в том же окне инспектора найдите компонент Text, в нем можно задать надпись, которая будет отображаться на хосте, а также настройки шрифтов:



8. После проделанных операций на сцене вы должны увидеть созданный холст (Canvas), с расположенным на нем текстовым полем с надписью Score:



После выполнения всех пунктов из раздела 4.1 на экране игрока в правой верхней части экрана должно отображаться поле для размещения текстовой информации о количестве пойманных объектов DragonEgg.

4.2 Добавление очков за пойманные объекты

1. Добавление очков должно происходить в том случае, когда объект DragonEgg (драконье яйцо) пересекается с энергетическим щитом EnergyShield, то есть считается пойманным. Для того чтобы определять событие столкновения DragonEgg и EnergyShield, можно использовать сценарии DragonEgg.cs и EnergyShield.cs.

2. В сценарии EnergyShield.cs уже есть метод private void OnCollisionEnter(Collision coll). Пока метод отвечает только за то, что уничтожает яйцо дракона после пересечения. Мы можем использовать этот же метод для начисления очков за пойманное яйцо DragonEgg.

3. Откройте сценарий EnergyShield.cs и добавьте в него строки кода, выделенные жирным шрифтом:

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public classEnergyShield : MonoBehaviour
{
    public Text scoreGT;
    void Start()
    {
        GameObject scoreGO = GameObject.Find("Score");
        scoreGT = scoreGO.GetComponent<Text>();
        scoreGT.text = "0";
    }
    void Update()
    {
        Vector3 mousePos2D = Input.mousePosition;
        mousePos2D.z = -Camera.main.transform.position.z;
                                    Vector3    mousePos3D    =
Camera.main.ScreenToWorldPoint(mousePos2D);
        Vector3 pos = this.transform.position;
        pos.x = mousePos3D.x;
        this.transform.position = pos;
    }
    private void OnCollisionEnter(Collision coll)
    {
        GameObject Collided = coll.gameObject;
        if (Collided.tag == "DragonEgg")
        {
            Destroy(Collided);
        }
        int score = int.Parse(scoreGT.text);
        score += 1;
        scoreGT.text = score.ToString();
    }
}
```

// End Code

Скриншот листинга приведен ниже. Пронумерованными рамками на рисунке ниже показаны новые части кода, которые следует написать на данном этапе.


```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class EnergyShield : MonoBehaviour
7   {
8       public Text scoreGT;
9
10      void Start()
11      {
12          GameObject scoreGO = GameObject.Find("Score");
13          scoreGT = scoreGO.GetComponent<Text>();
14          scoreGT.text = "0";
15      }
16
17      void Update()
18      {
19          Vector3 mousePos2D = Input.mousePosition;
20          mousePos2D.z = -Camera.main.transform.position.z;
21          Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
22          Vector3 pos = this.transform.position;
23          pos.x = mousePos3D.x;
24          this.transform.position = pos;
25      }
26
27      private void OnCollisionEnter(Collision coll)
28      {
29          GameObject Collided = coll.gameObject;
30          if (Collided.tag == "DragonEgg")
31          {
32              Destroy(Collided);
33          }
34          int score = int.Parse(scoreGT.text);
35          score += 1;
36          scoreGT.text = score.ToString();
37      }
38  }
```

Описание новых команд листинга в примере выше:

– С помощью команды using UnityEngine.UI подключается библиотека, позволяющая использовать интерфейс пользователя .UI (User Interface), доступная внутри движка Unity.

– Далее объявляется переменная scoreGT типа public, которая будет хранить в себе счетчик пойманных объектов DragonEgg.

– GameObject scoreGO = GameObject.Find(“Score”); – находит на сцене игровой объект с именем Score, и присваивает ссылку на объект переменной scoreGO.

– scoreGT = scoreGO.GetComponent<Text>(); – находит компонент Text внутри игрового объекта scoreGO и присваивает ссылку на него в public-поле scoreGT.

– scoreGT.text = "0"; – публичному полю scoreGT присваивается начальное значение равное нулю. Обратите внимание на то, что только благодаря строке в начале сценария using UnityEngine.UI; можно получать доступ к компоненту Text в коде на C#;

– int score = int.Parse(scoreGT.text); – получает текст, записанный в поле ScoreCounter и преобразует его в целое число типа int. Далее это число записывается в переменную с именем score;

– score += 1; – значение переменной увеличивается на 1 с каждым вызовом функции OnCollisionEnter(Collision coll), т.е. с каждым пойманным яйцом;

– scoreGT.text = score.ToString(); – переменная score преобразуется обратно в строковую переменную и записывается в свойство text объекта scoreGT;

4.3 Уведомление о том, что яйцо DragonEgg не было поймано

На данном этапе, если яйцо DragonEgg не было поймано, – то оно удаляется автоматически. Но чтобы можно было завершить раунд и удалить все объекты, упавшие мимо, яйцо DragonEgg должно также уведомлять сценарий DragonPicker.cs о случившемся событии (падении на плоскость Plane). Для этого нужно сделать так, чтобы один сценарий вызывал функцию в другом сценарии.

1. Внесите следующие изменения в сценарий DragonEgg.cs:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DragonEgg : MonoBehaviour
{
    public static float bottomY = -30f;
    void Start()
    {
    }
    private void OnTriggerEnter(Collider other)
    {
        ParticleSystem ps = GetComponent<ParticleSystem>();
        var em = ps.emission;
        em.enabled = true;
        Renderer rend;
        rend = GetComponent<Renderer>();
        rend.enabled = false;
    }
    void Update()
    {
        if (transform.position.y < bottomY)
        {
            Destroy(this.gameObject);
                                    DragonPicker apScript =
Camera.main.GetComponent<DragonPicker>();
            apScript.DragonEggDestroyed();
        }
    }
}
```

// End Code

Скриншот листинга с выделенным новым фрагментом кода показан ниже.

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class DragonEgg : MonoBehaviour
6   {
7       public static float bottomY = -30f;
8
9       void Start()
10      {
11      }
12
13      private void OnTriggerEnter(Collider other)
14      {
15          ParticleSystem ps = GetComponent<ParticleSystem>();
16          var em = ps.emission;
17          em.enabled = true;
18
19          Renderer rend;
20          rend = GetComponent<Renderer>();
21          rend.enabled = false;
22      }
23
24      void Update()
25      {
26          if (transform.position.y < bottomY)
27          {
28              Destroy(this.gameObject);
29              DragonPicker apScript = Camera.main.GetComponent<DragonPicker>();
30              apScript.DragonEggDestroyed();
31          }
32      }
33  }
```

Описание новых команд листинга:

– DragonPicker apScript = Camera.main.GetComponent< DragonPicker >(); – в локальную переменную apScript записывается ссылка на компонент DragonPicker (Script) главной камеры Main Camera.

– После этого появляется возможность напрямую обращаться к переменным и методам экземпляра класса DragonPicker.cs, подключенного к главной камере.

– apScript.DragonEggDestroyed(); – это вызов метода DragonEggDestroyed() класса DragonPicker. Но метода пока не существует, давайте добавим его.

2. Откройте сценарий DragonPicker.cs, добавьте в него новый метод DragonEggDestroyed():

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
    public float energyShieldRadius = 1.5f;
    void Start()
    {
```

```
            for (int i = 1; i <= numEnergyShield; i++)
            {
                                        GameObject tBasketGo =
Instantiate<GameObject>(energyShieldPrefab);
                 tBasketGo.transform.position = new Vector3(0,
energyShieldButtomY, 0);
                tBasketGo.transform.localScale = new Vector3(1
* i, 1 * i, 1 * i);
            }
        }
        void Update()
        {
        }
        public void DragonEggDestroyed()
        {
    GameObject[]                tDragonEggArray                =
GameObject.FindGameObjectsWithTag("DragonEgg");
            foreach (GameObject tGO in tDragonEggArray)
            {
                Destroy(tGO);
            }
        }
    }
```

// End Code

Скриншот кода приведен ниже.

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class DragonPicker : MonoBehaviour
6   {
7       public GameObject energyShieldPrefab;
8       public int numEnergyShield = 3;
9       public float energyShieldButtomY = -6f;
10      public float energyShieldRadius = 1.5f;
11
12      void Start()
13      {
14          for (int i = 1; i <= numEnergyShield; i++)
15          {
16              GameObject tBasketGo = Instantiate<GameObject>(energyShieldPrefab);
17              tBasketGo.transform.position = new Vector3(0, energyShieldButtomY, 0);
18              tBasketGo.transform.localScale = new Vector3(1 * i, 1 * i, 1 * i);
19          }
20      }
21
22      void Update()
23      {
24      }
25
26      public void DragonEggDestroyed()
27      {
28          GameObject[] tDragonEggArray = GameObject.FindGameObjectsWithTag("DragonEgg");
29          foreach (GameObject tGO in tDragonEggArray)
30          {
31              Destroy(tGO);
32          }
33      }
34  }
```

Описание новых команд листинга:

– метод DragonEggDestroy имеет тип public (то есть является общедоступным), чтобы его можно было вызывать из других классов, например, таких как DragonEgg.cs;

– строки кода внутри созданного метода возвращают массив всех существующих игровых объектов DragonEgg и с помощью цикла foreach выполняется обход всех найденных объектов с их последующим уничтожением.

4.4 Уничтожение EnergyShield после потери драконьего яйца DragonEgg

После потери DragonEgg должен удаляться один энергетический щит EnetgyShield. После потери трех энергетических щитов – игра должна останавливаться. Для этого сценарий DragonPicker.cs должен содержать следующие строки кода (приведем листинг целиком, с указанием новых строк кода):

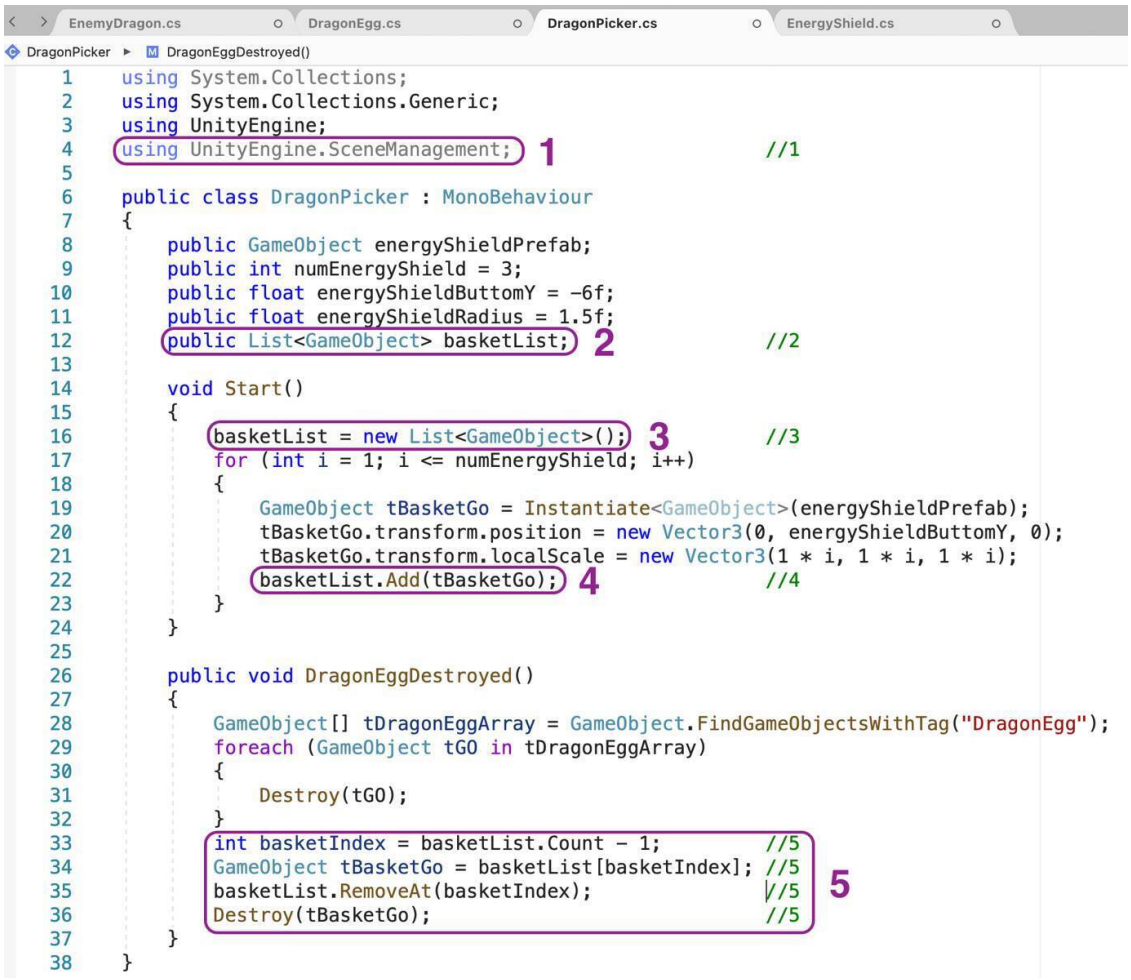
// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement; //1
public classDragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
    public float energyShieldRadius = 1.5f;
    public List<GameObject> basketList; //2
    void Start()
    {
        basketList = new List<GameObject>(); //3
        for (int i = 1; i <= numEnergyShield; i++)
        {
                                    GameObject tBasketGo =
Instantiate<GameObject>(energyShieldPrefab);
                tBasketGo.transform.position = new Vector3(0,
energyShieldButtomY, 0);
                tBasketGo.transform.localScale = new Vector3(1
* i, 1 * i, 1 * i);
            basketList.Add(tBasketGo); //4
        }
    }
    void Update()
    {
    }
    public void DragonEggDestroyed()
    {
                            GameObject[] tDragonEggArray =
GameObject.FindGameObjectsWithTag("DragonEgg");
        foreach (GameObject tGO in tDragonEggArray)
        {
            Destroy(tGO);
        }
        int basketIndex = basketList.Count - 1; //5
        GameObject tBasketGo = basketList[basketIndex]; //5
        basketList.RemoveAt(basketIndex); //5
```

```
            Destroy(tBasketGo); //5
        }
    }
```

// End Code

Скриншот кода приведен ниже.




```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;                  //1
5
6   public class DragonPicker : MonoBehaviour
7   {
8       public GameObject energyShieldPrefab;
9       public int numEnergyShield = 3;
10      public float energyShieldButtomY = -6f;
11      public float energyShieldRadius = 1.5f;
12      public List<GameObject> basketList;             //2
13
14      void Start()
15      {
16          basketList = new List<GameObject>();        //3
17          for (int i = 1; i <= numEnergyShield; i++)
18          {
19              GameObject tBasketGo = Instantiate<GameObject>(energyShieldPrefab);
20              tBasketGo.transform.position = new Vector3(0, energyShieldButtomY, 0);
21              tBasketGo.transform.localScale = new Vector3(1 * i, 1 * i, 1 * i);
22              basketList.Add(tBasketGo);              //4
23          }
24      }
25
26      public void DragonEggDestroyed()
27      {
28          GameObject[] tDragonEggArray = GameObject.FindGameObjectsWithTag("DragonEgg");
29          foreach (GameObject tGO in tDragonEggArray)
30          {
31              Destroy(tGO);
32          }
33          int basketIndex = basketList.Count - 1;     //5
34          GameObject tBasketGo = basketList[basketIndex]; //5
35          basketList.RemoveAt(basketIndex);           //5
36          Destroy(tBasketGo);                         //5
37      }
38  }
```

Описание новых команд листинга в примере выше:

– часть кода #1 подключает библиотеку SceneManagement для возможности работы со сценами. Нам она потребуется в дальнейшем;

– часть кода #2 и #3 определяет новый список типа List<GameObject>;

– часть кода #4 в конце цикла for добавляет каждый созданный EnergyShield в список basketList;

– часть кода #5 уничтожает один щит с каждым очередным срабатыванием метода DragonEggDestroyed();

3. Теперь, если запустить игру и потерять все три энергетических щита EnergyShield, то программа завершиться с исключением IndexOutOfRange, т. к. на данный момент в игре не прописано, что требуется сделать программе в случае потери всех корзин.

4. Чтобы исключение не возникало, в метод DragonEggDestroyed() сценария DragonPicker.cs следует добавить строки, возвращающие игру в исходное состояние, когда игрок теряет все три энергетических щита (что в нашей концепции игры символизирует жизни):

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public classDragonPicker : MonoBehaviour
{
    public GameObject energyShieldPrefab;
    public int numEnergyShield = 3;
    public float energyShieldButtomY = -6f;
    public float energyShieldRadius = 1.5f;
    public List<GameObject> basketList;
    void Start()
    {
        basketList = new List<GameObject>();
        for (int i = 1; i <= numEnergyShield; i++)
        {
                                        GameObject   tBasketGo   =
Instantiate<GameObject>(energyShieldPrefab);
                tBasketGo.transform.position = new Vector3(0,
energyShieldButtomY, 0);
               tBasketGo.transform.localScale = new Vector3(1
* i, 1 * i, 1 * i);
            basketList.Add(tBasketGo);
        }
    }
    void Update()
    {
    }
    public void DragonEggDestroyed()
    {
                            GameObject[]   tDragonEggArray   =
GameObject.FindGameObjectsWithTag("DragonEgg");
        foreach (GameObject tGO in tDragonEggArray)
        {
            Destroy(tGO);
        }
        int basketIndex = basketList.Count - 1;
        GameObject tBasketGo = basketList[basketIndex];
        basketList.RemoveAt(basketIndex);
        Destroy(tBasketGo);
        if (basketList.Count == 0)
        {
            SceneManager.LoadScene("_0Scene");
        }
    }
}
// End Code
```

Так как листинг DragonPicker.cs уже достаточно большой, приведем только скриншот метода DragonEggDestroyed, в который были добавлены новые строки кода.

```
26          public void DragonEggDestroyed()
27          {
28              GameObject[] tDragonEggArray = GameObject.FindGameObjectsWithTag("DragonEgg");
29              foreach (GameObject tGO in tDragonEggArray)
30              {
31                  Destroy(tGO);
32              }
33              int basketIndex = basketList.Count - 1;
34              GameObject tBasketGo = basketList[basketIndex];
35              basketList.RemoveAt(basketIndex);
36              Destroy(tBasketGo);
37              if (basketList.Count == 0)
38              {
39                  SceneManager.LoadScene("_0Scene");
40              }
41          }
42      }
```

Теперь проверьте, что после потери всех трех энергетических щитов происходит перезапуск сцены, а счетчик заработанных очков сбрасывается (снова становится равным нулю).

Выводы

В четвертой части мы проделали большую работу, – добавили в игру смысл с помощью простого добавления счетчика очков. В качестве дополнительного задания попробуйте сохранять рекорд игрока. Для этого стоит изучить работу методов PlayerPrefs.Set() и PlayerPrefs.Get(). С помощью данных методов можно запоминать переменные для отдельных пользователей во время игры. При повторном входе в игру эти переменные сохранят свои значения.

Часть 5. Доработка сцен и создание игрового меню

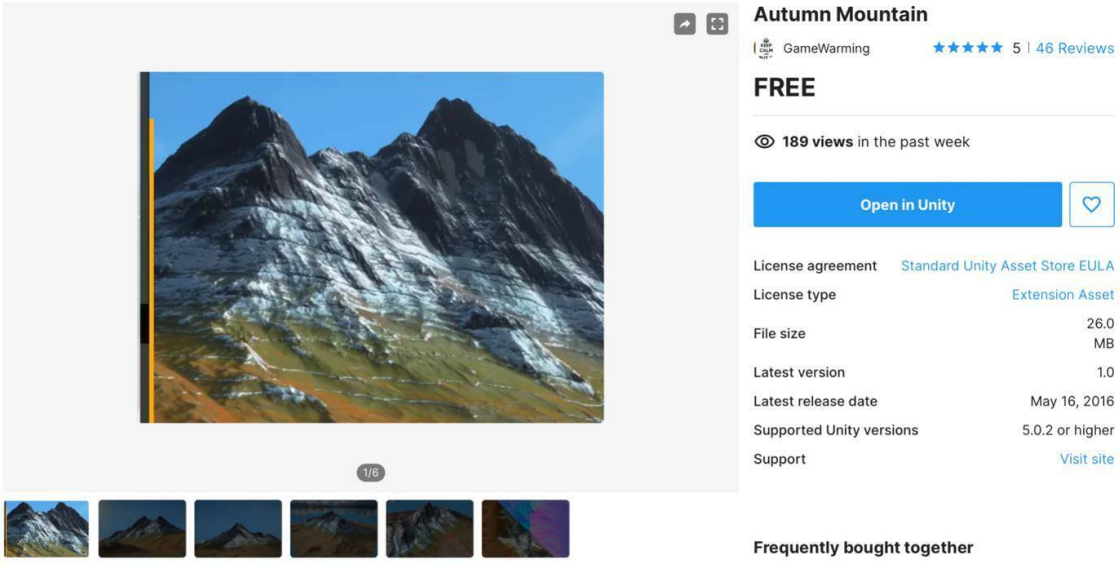
Введение

В этом разделе мы поработаем над улучшением внешнего вида сцены, а также создадим дополнительную стартовую сцену.

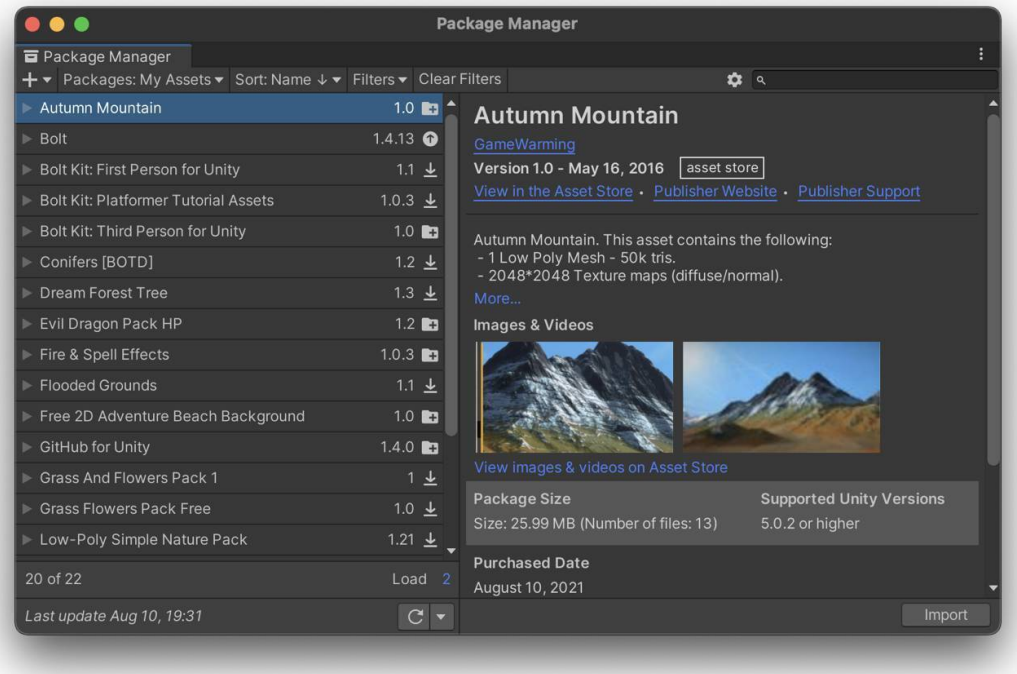
Ранее мы работали с одной сценой – **_0Scene**, на которой происходила вся разработка игры. В этом разделе мы добавим новую стартовую сцену, на которую будет попадать пользователь сразу после запуска игры.

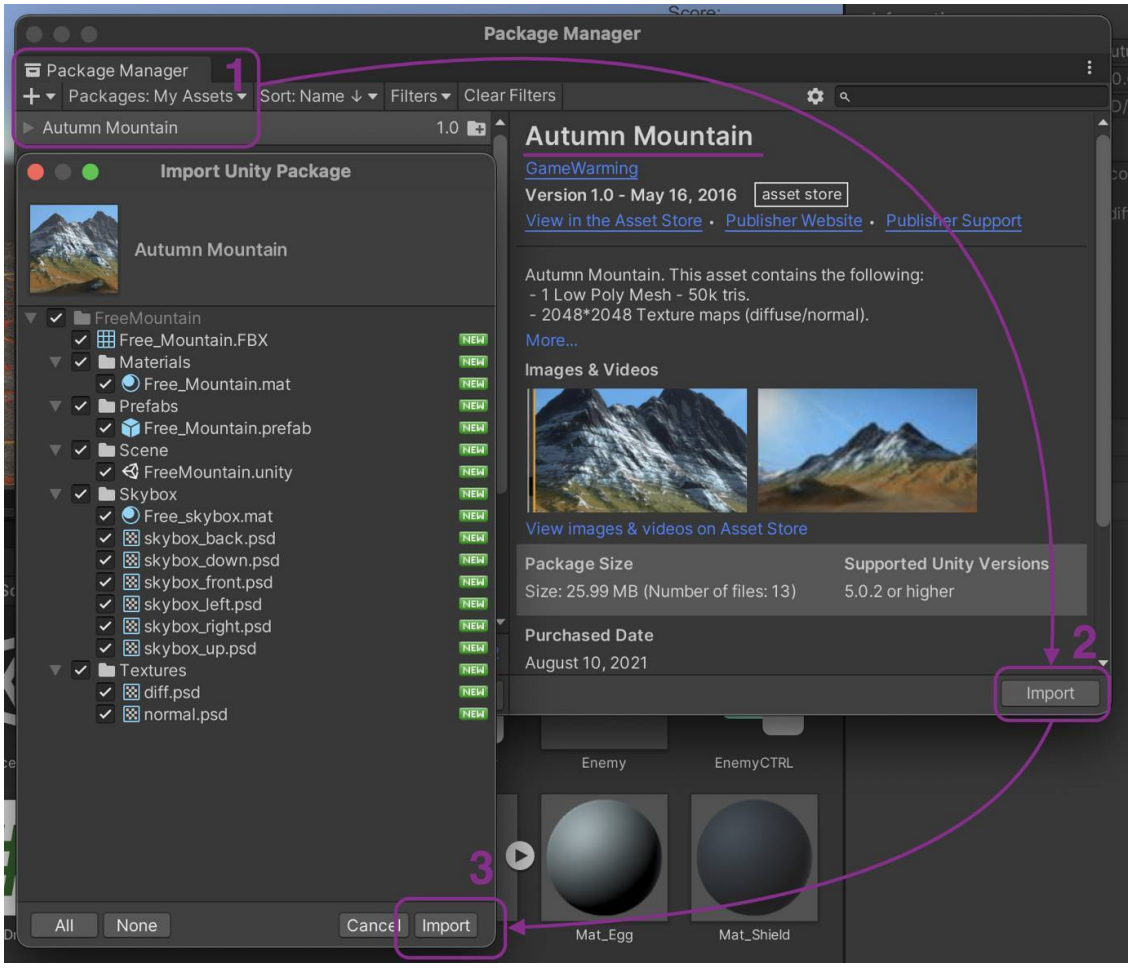
5.1 Доработка локации в сцене _1Scene

Чтобы локация выглядела более привлекательно, добавим в качестве декораций горы и немного доработаем оформление сцены. Для этого нам понадобится еще один Asset-пак под названием Autumn Mountain из уже знакомого Asset Store. Перейдите на сайт assetstore.unity.com и найдите нужный ассет:

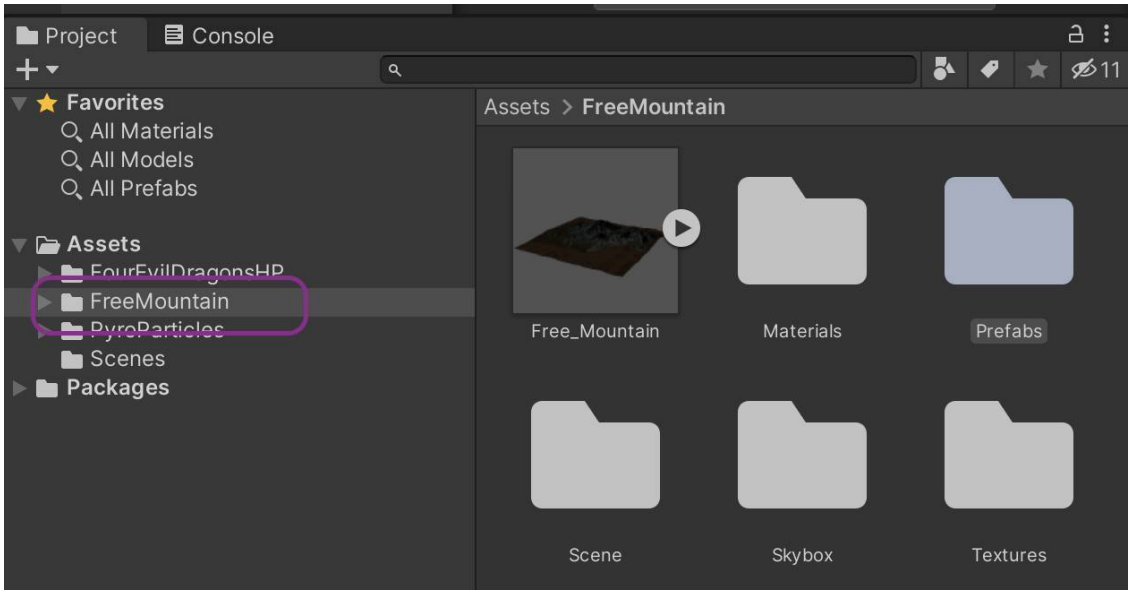


Также как и ранее добавьте ассет в свою коллекцию, нажав на сайте кнопку “Add to My Assets”. После этого вернитесь в среду разработки Unity и открыв Window – Package Manager скачайте и импортируйте пакет Autumn Mountain:

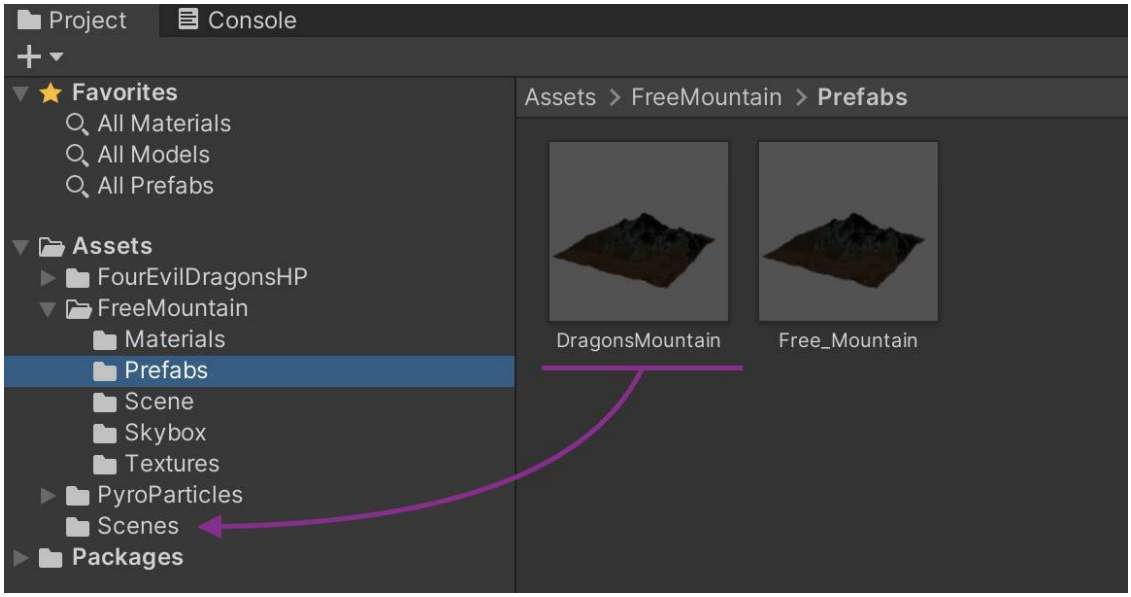




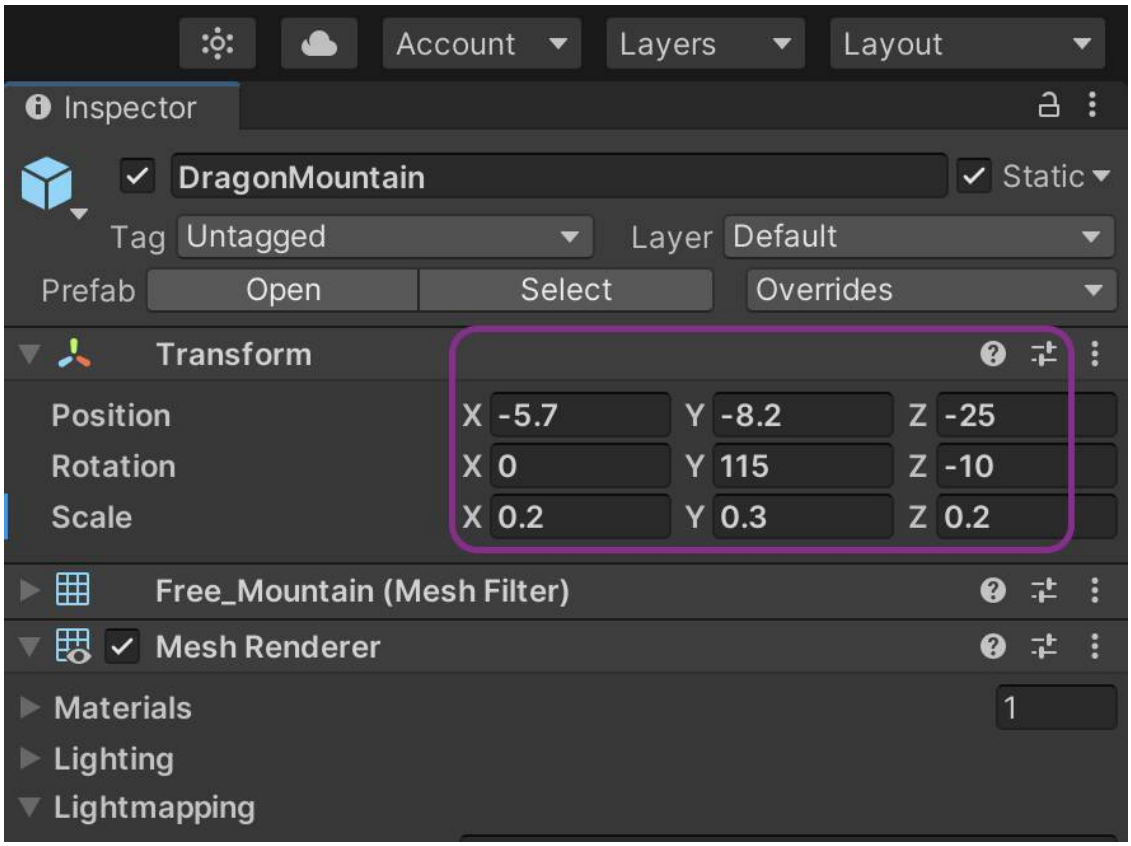
После импорта в папке Assets (см. окно Project) появится новая папка с именем Free Mountains, содержащая текстуры, префабы и материалы добавленного ассет-пака:



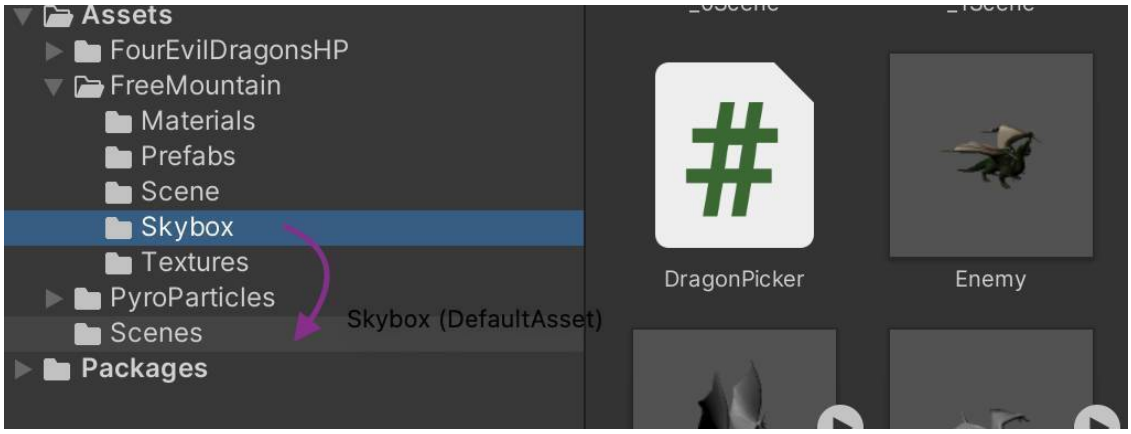
Давайте создадим дубликат префаба Free_Mountain, для этого выделите объект и нажмите сочетание клавиш Ctrl+D (или Command+D). Переименуйте созданную копию объекта в DragonMountain и переместите его в папку Assets/Scenes, как показано на рисунке ниже:



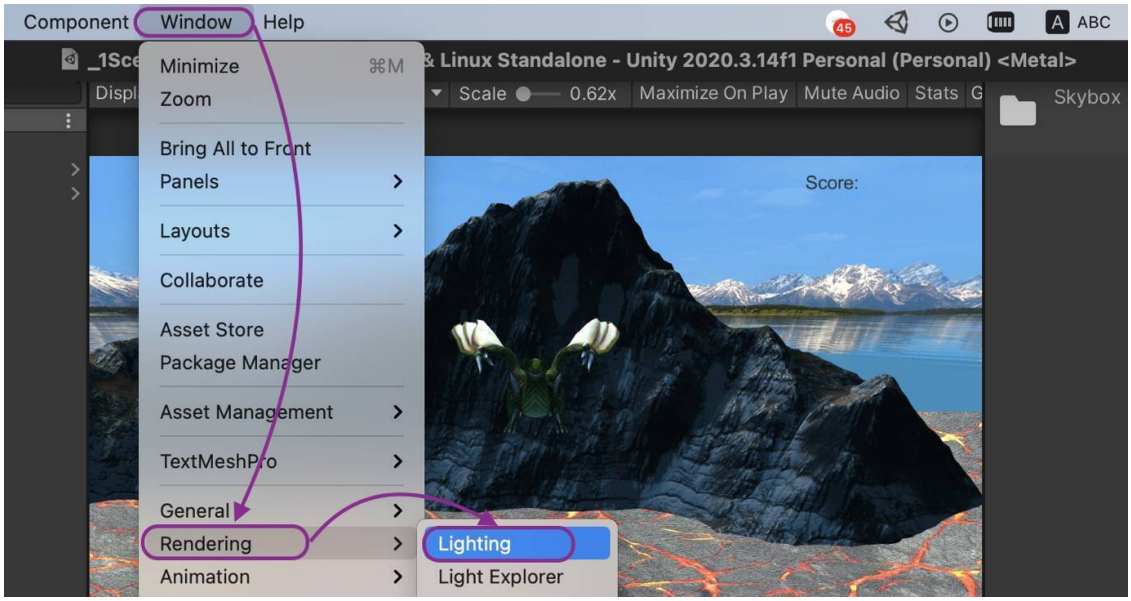
Убедитесь, что выбрана сцена с именем _1Scene и перетащите объект DragonMountain.prefab в окно Hierarchy. Измените настройки компонента Transform добавленного префаба с горой:



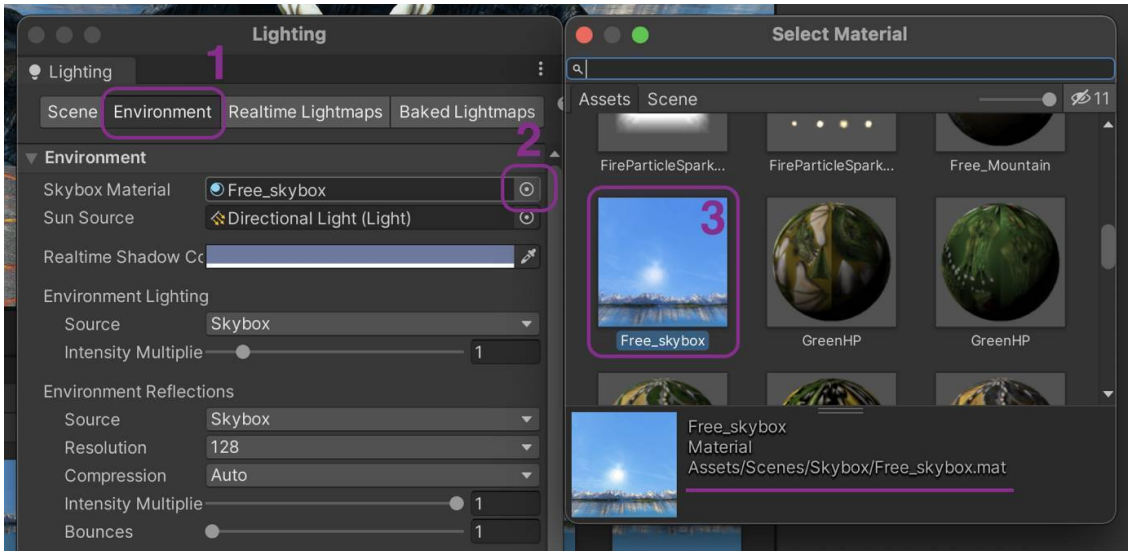
Несмотря на то, что горы создают достаточно хорошую декорацию, нам не хватает текстур на заднем плане сцены. В последнем скачанном нами из Asset Store текстур-паке для этой цели можно использовать текстуры из папки Assets/FreeMountain/Skybox. Переместите (можно также перетаскиванием) папку Skybox из FreeMountaion в папку Scenes:



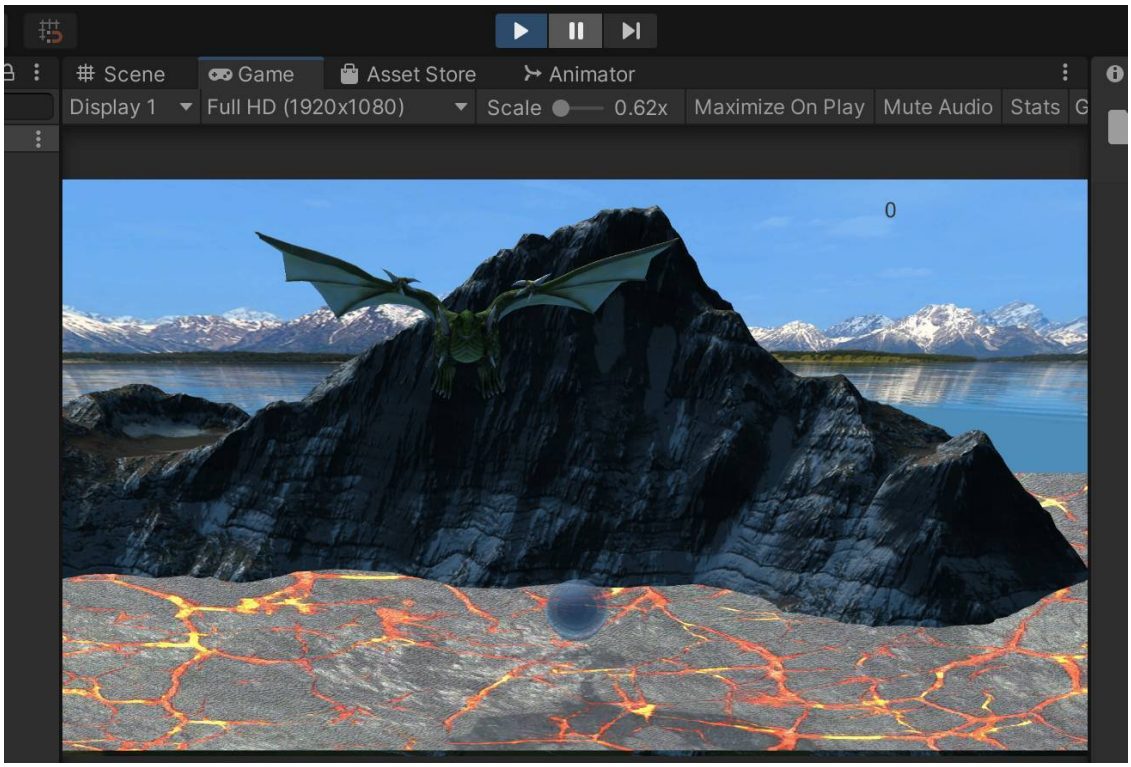
Чтобы применить текстуру Skybox к сцене, откройте в верхнем меню Window – Rendering – Lighting:



В настройках окружения Environment, в поле Skybox Material следует выбрать материал с именем Free_skybox.mat:



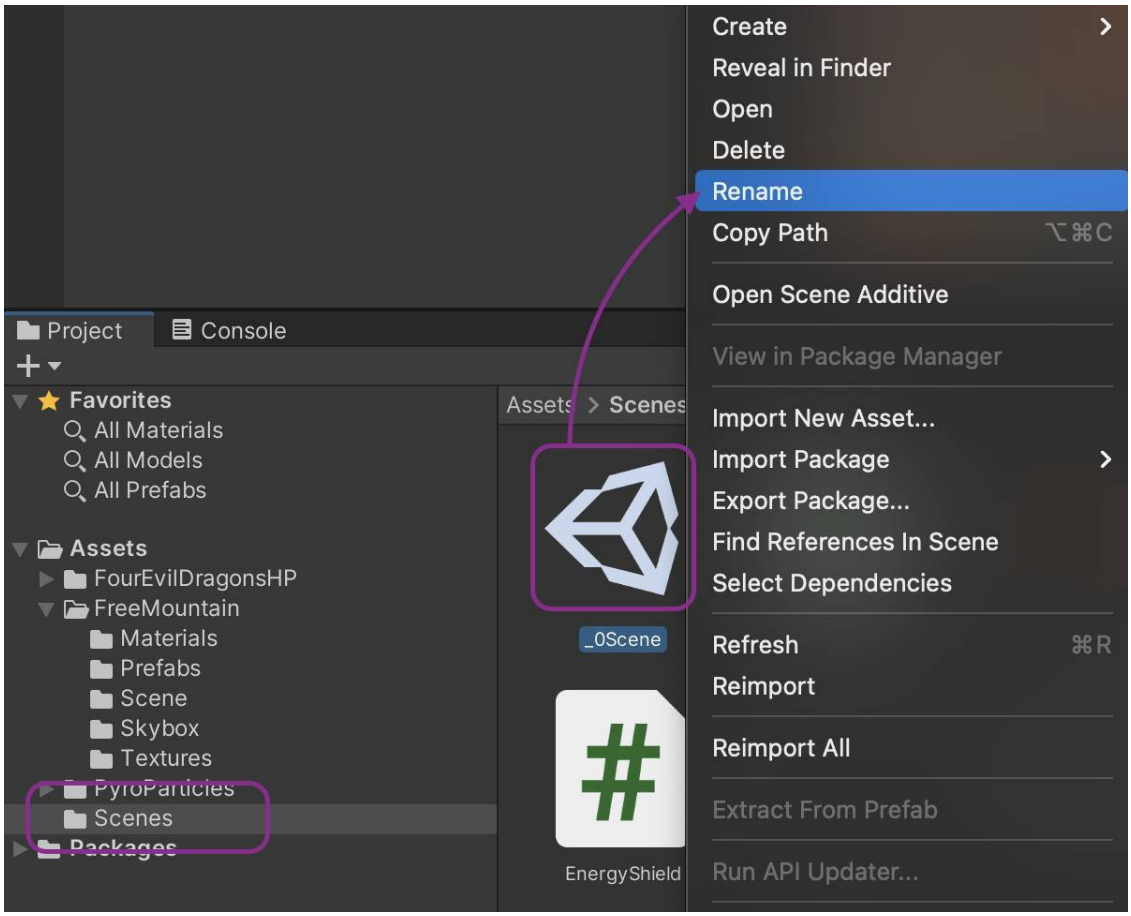
После этого сцена должна выглядеть так как показано на рисунке ниже. На сцену добавлена гора, а на заднем плане размещены текстуры с изображением голубого неба и других снежных гор:



Сохраните сцену _1Scene, кликнув по ней правой кнопкой мыши и выбрав из контекстного меню Save Scene.

5.2 Создание стартовой сцены

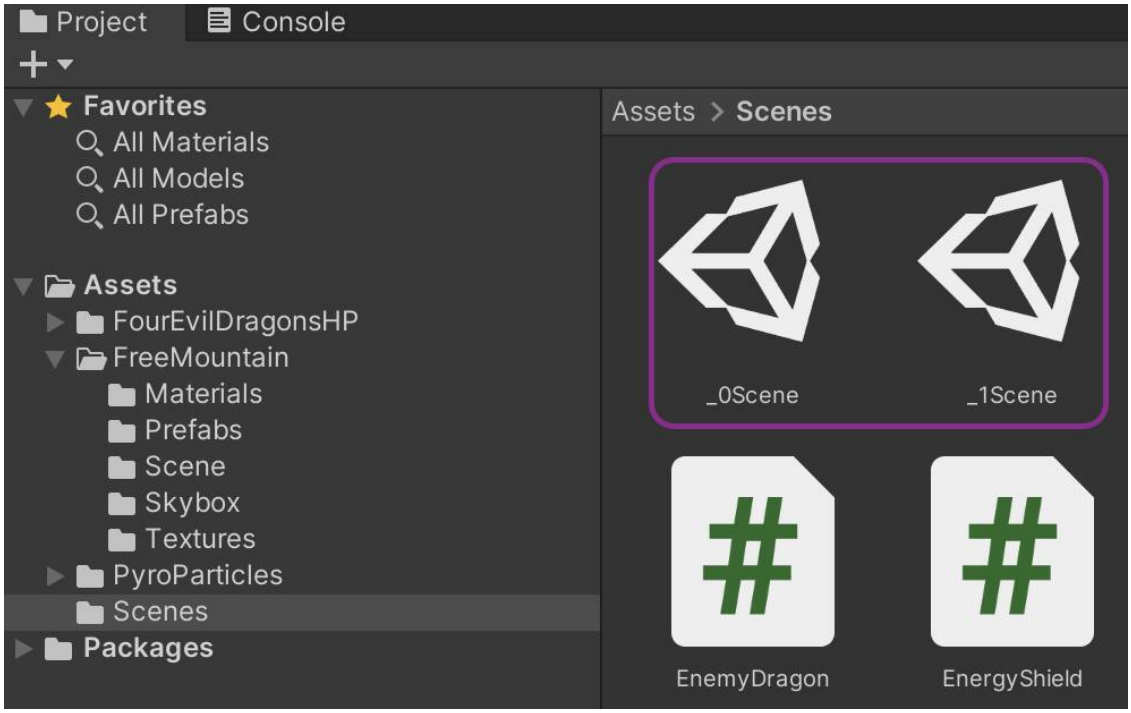
Теперь мы добавим новую стартовую сцену, на которую будет попадать пользователь сразу после запуска игры. Правила хорошего тона при разработке требуют четкой иерархии в объектах, сценах, скриптах и т. д. Переименуйте текущую _0Scene в _1Scene. Так мы обозначим, что игровая сцена смещается по иерархии загрузки на уровень ниже.



Добавить новую сцену можно, создав копию текущей сцены, для этого выделите объект _1Scene и нажмите сочетание клавиш Ctrl+D (или Command+D). Переименуйте созданную копию объекта в _0Scene, это будет имя новой созданной сцены. После этого в папке должно находиться две сцены:

_0Scene – копия сцены, в которой мы создадим стартовое меню;

_1Scene – основная сцена с драконом и горами, с которой мы работали основную часть нашего практикума.



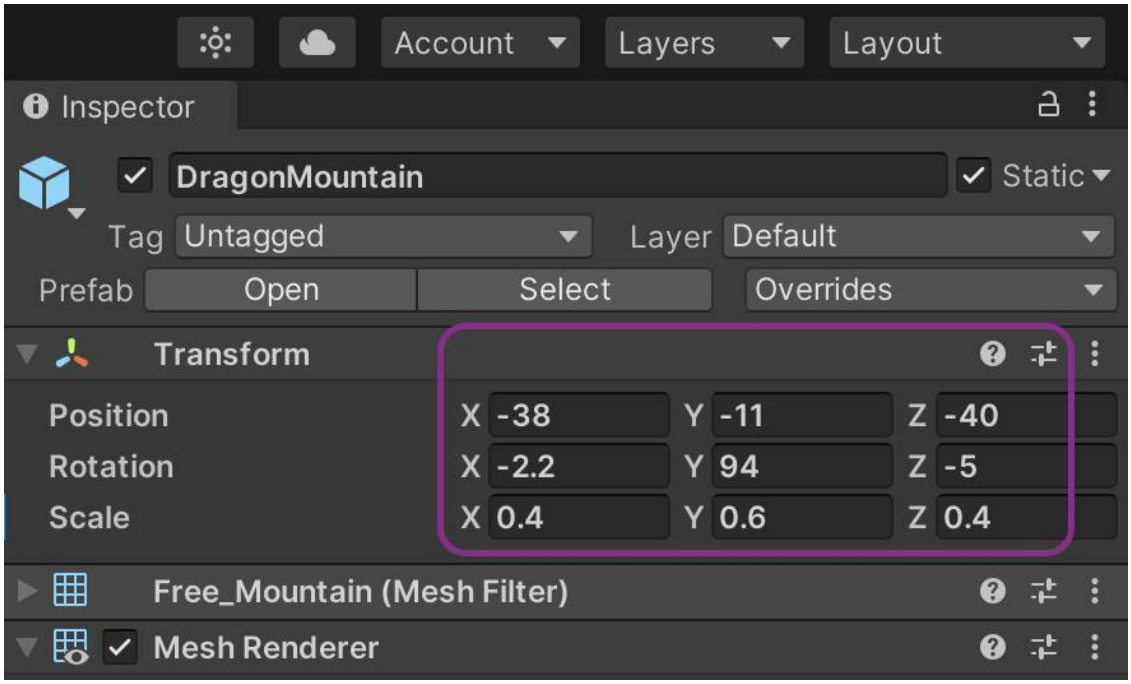
Далее мы настроим стартовую сцену _0Scene таким образом, чтобы на ней было удобно разместить элементы меню навигации. Но для начала давайте внесем некоторые изменения в имеющуюся структуру сцены.

1. Удалите объект Ground, для этого в окне Hierarchy кликните по нему правой кнопкой мыши и в контекстном меню нажмите Delete.

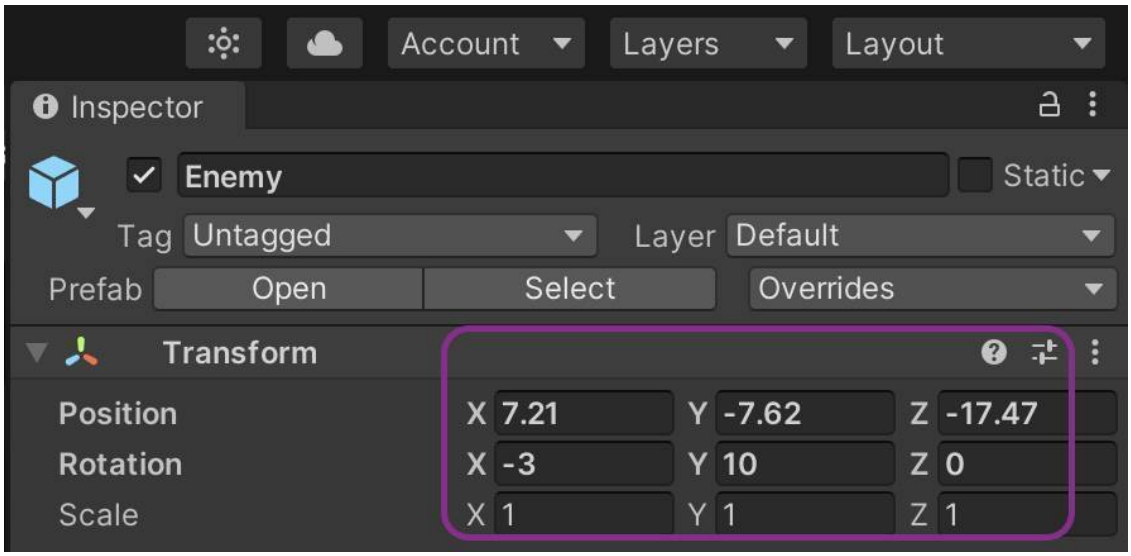
2. Удалите объект Score, который находится внутри холста Canvas.

3. Выберите объект Main Camera и в окне Inspector удалите компонент с подключенным скрипт-файлом DragonPicker.cs (кликните по трем точкам справа от названия компонента и выберите Remove Component). В этой сцене он нам не потребуется.

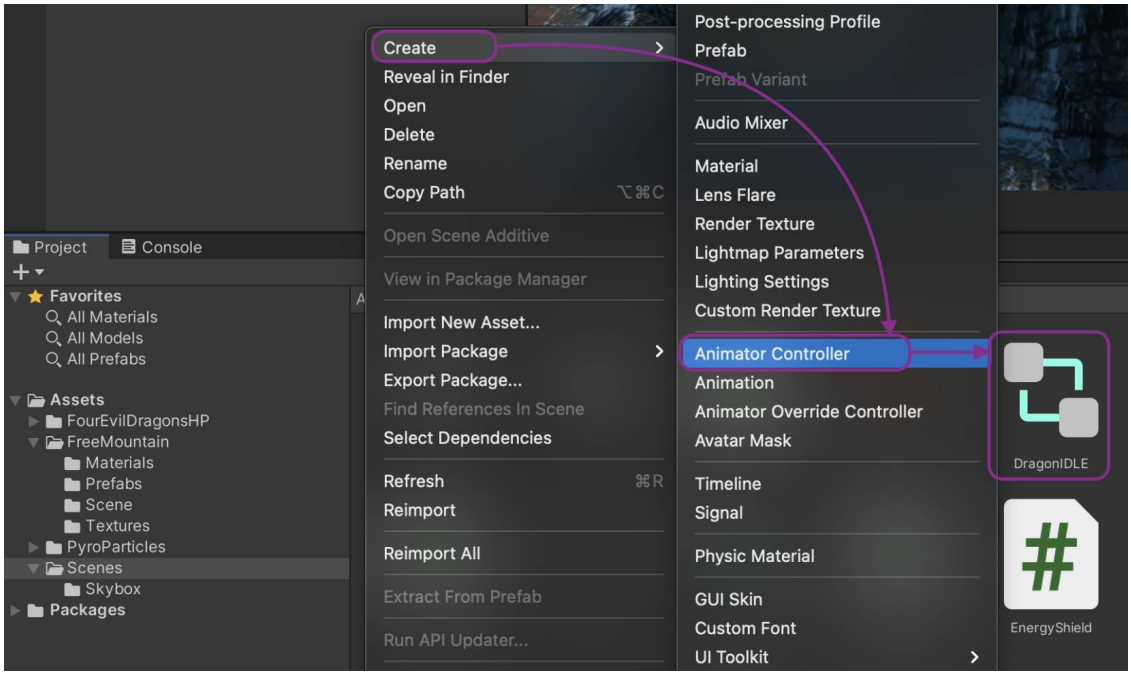
4. В окне Hierarchy выберите объект DragonMountain и в окне Inspector настройте компонент Transform так, как показано на рисунке ниже. Это позволит несколько изменить масштаб горы (за счет увеличения Scale) и изменить ее ориентацию в пространстве, чтобы она располагалась с более удачного ракурса при входе в игру:



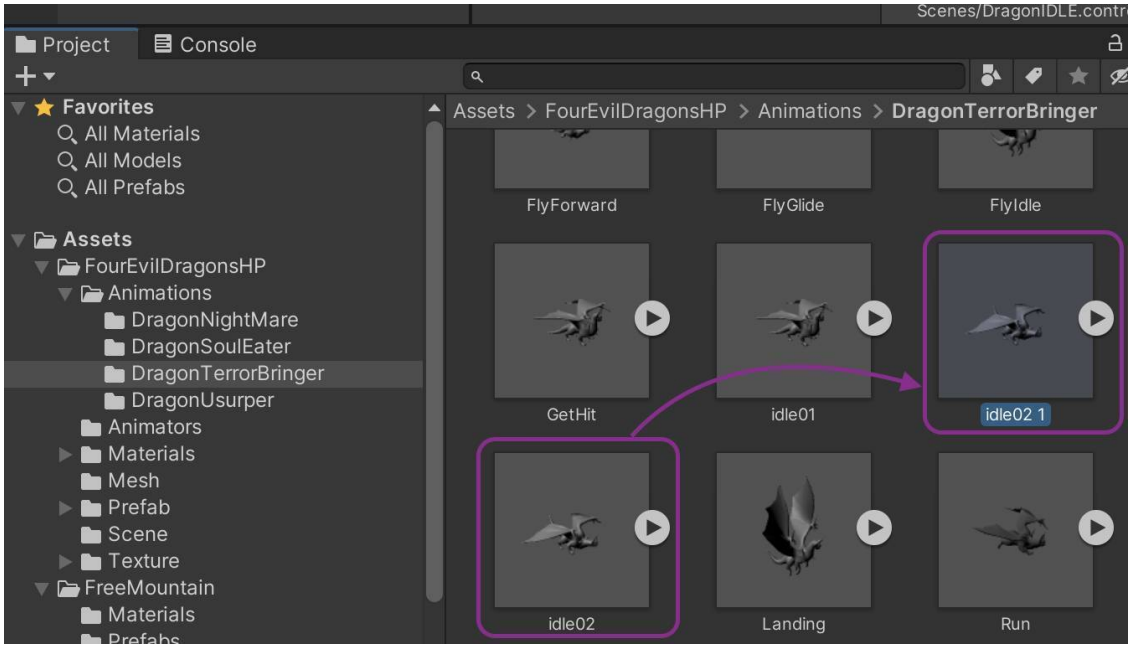
5. Далее выберите объект Enemy (игровой персонаж – дракон) и также измените в инспекторе настройки Transform, как показано на рисунке ниже. Это позволит переместить дракона в область впадины в горе, которую будет хорошо видно при входе на стартовую сцену игры:



6. Давайте изменим анимацию дракона на стартовой сцене. Для этого создадим отдельный контроллер. Кликните правой кнопкой мыши внутри папки Scenes и выберите из контекстного меню Create – Animator Controller, дайте контроллеру имя DragonIDLE.

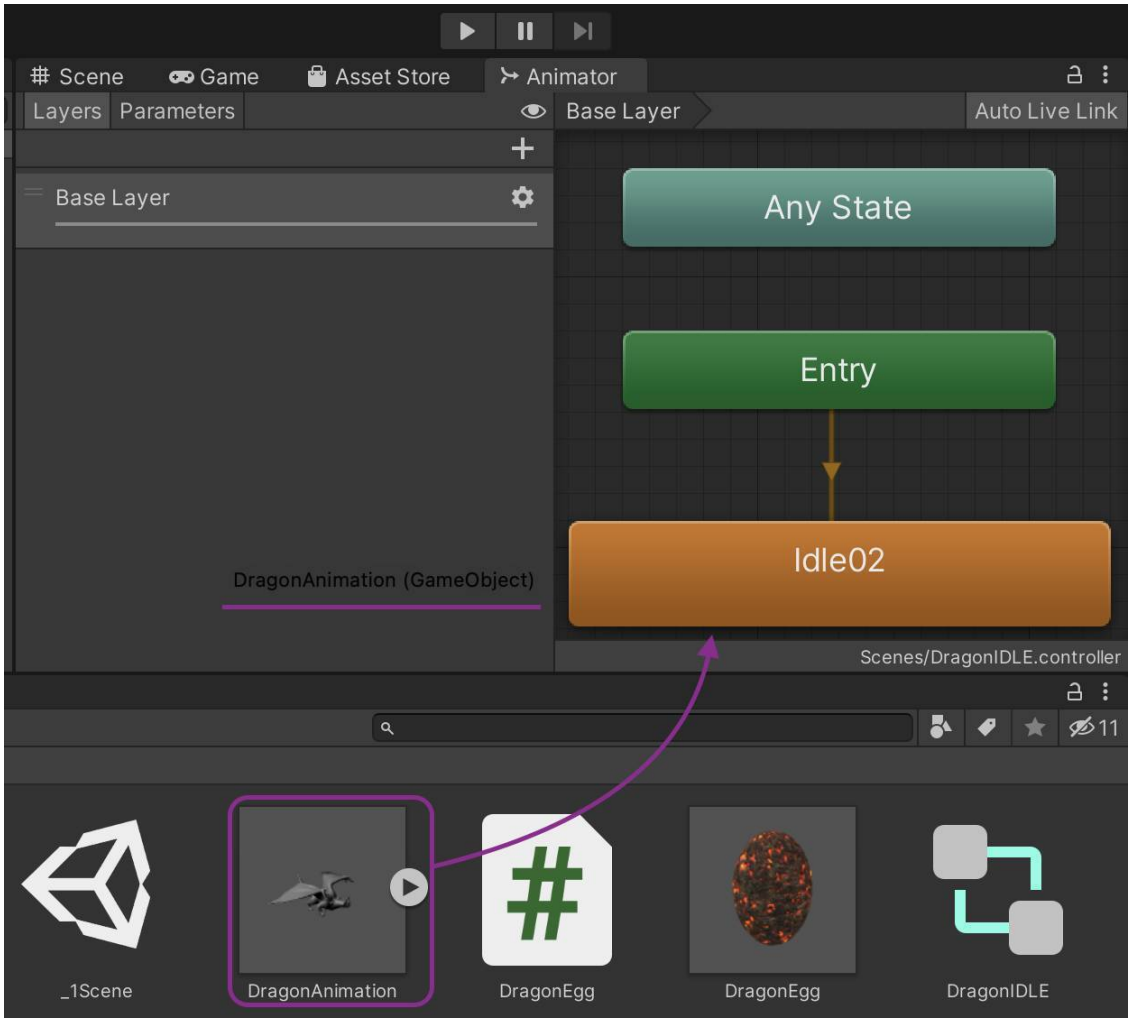


7. В папке Assets – Animations – DragonTerrorBringer найдем подходщую анимацию, например idle02, создайте ее копию (Ctrl + D или Command + D):

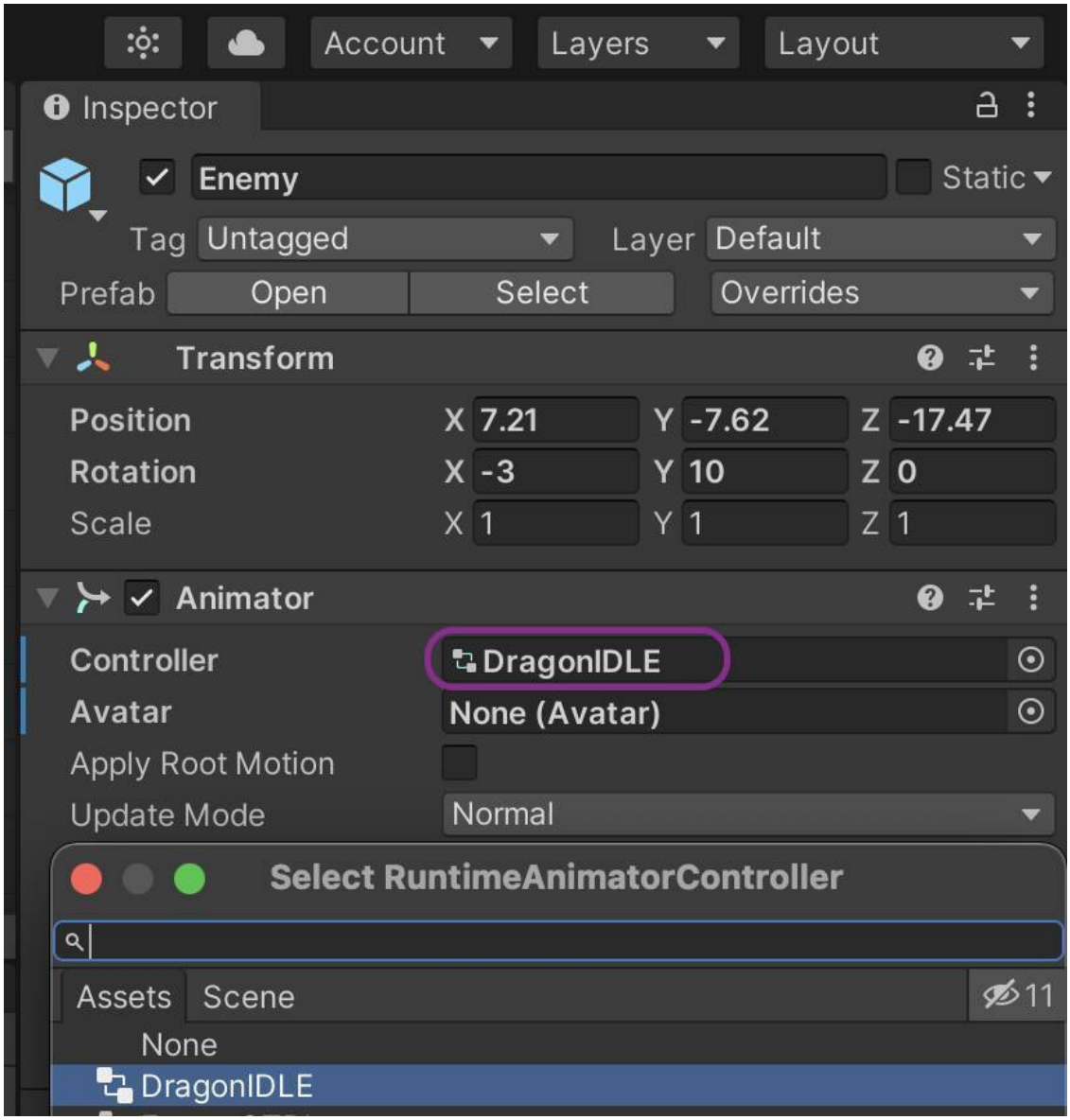


8. Дайте дубликату имя DragonAnimation и перенесите его в папку Scene.

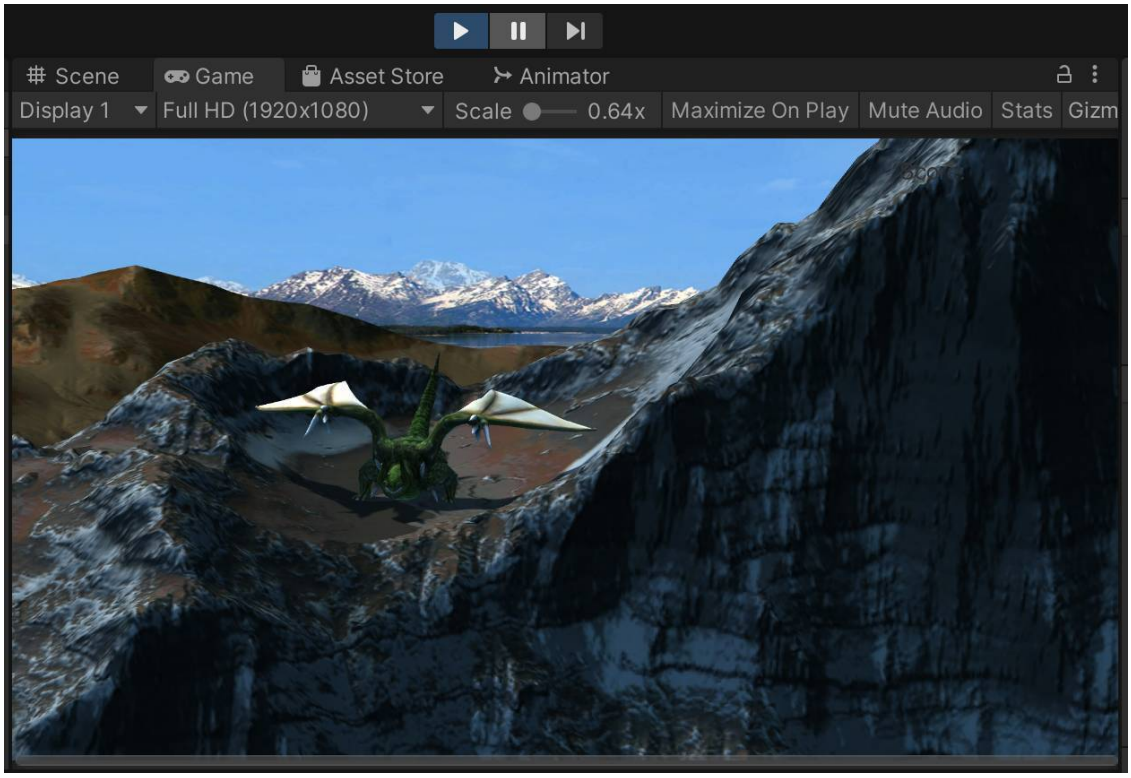
9. В папке Scene откройте контроллер DragonIDLE и перетяните в него только что добавленную в папку анимацию DragonAnimation. Это значит то, что мы создали контроллер для состояния простоя дракона, внутри контроллера находится анимация дракона, сидящего на поверхности земли:



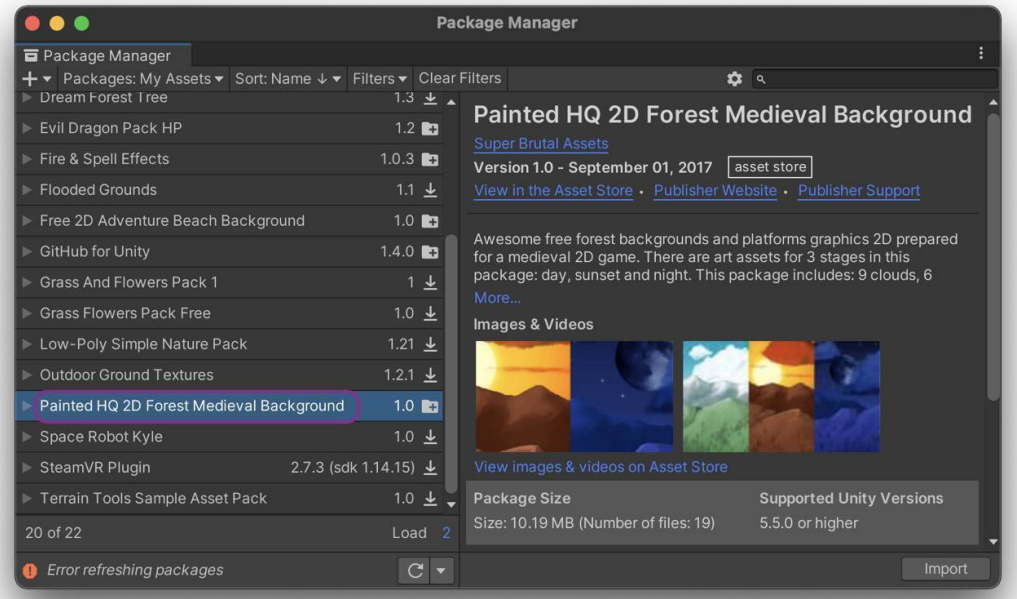
10. Теперь выберите дракона Enemy в окне Hierarchy, и после этого в окне инспектора в правой части в компоненте Animator выберите другой контроллер, с именем DragonIDLE.



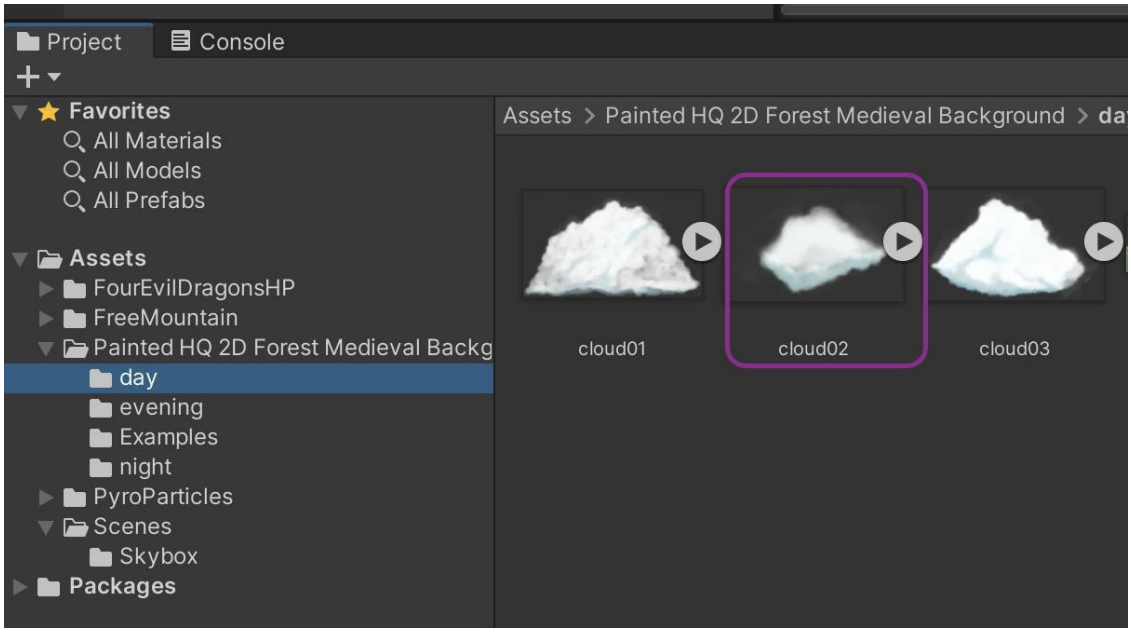
11. Сохраните сцену _0Scene и запустите ее (кнопка Run). Теперь при запуске сцены вы должны увидеть гору и анимированного дракона, сидящего на ее поверхности:



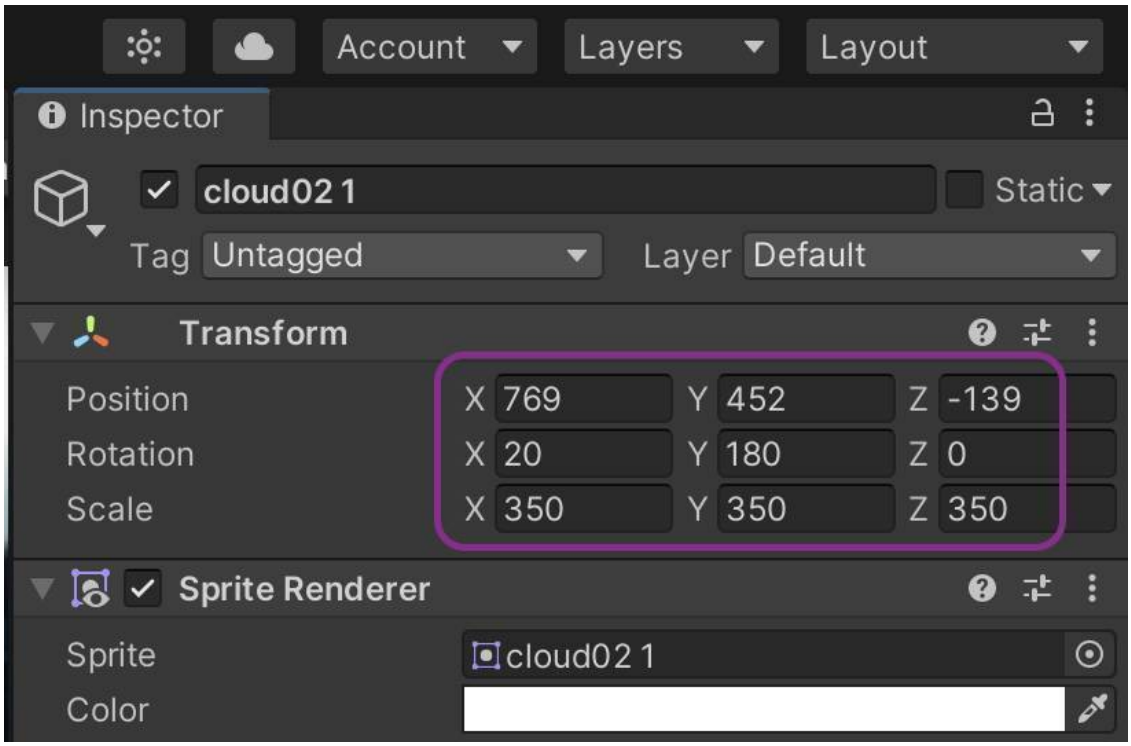
12. Для того, чтобы сцена выглядела еще более живо, можно добавить анимацию движения облаков из ассета с названием Painted HQ 2D Forest Medieval Background. Мы уже неоднократно находили нужный ассет-пак на сайте Asset Store и добавляли его в наш проект. Просто повторите необходимые действия и импортируйте пакет с текстурами.



13. Найдите в виде облака cloud02 из скачанного пакета в папке day:

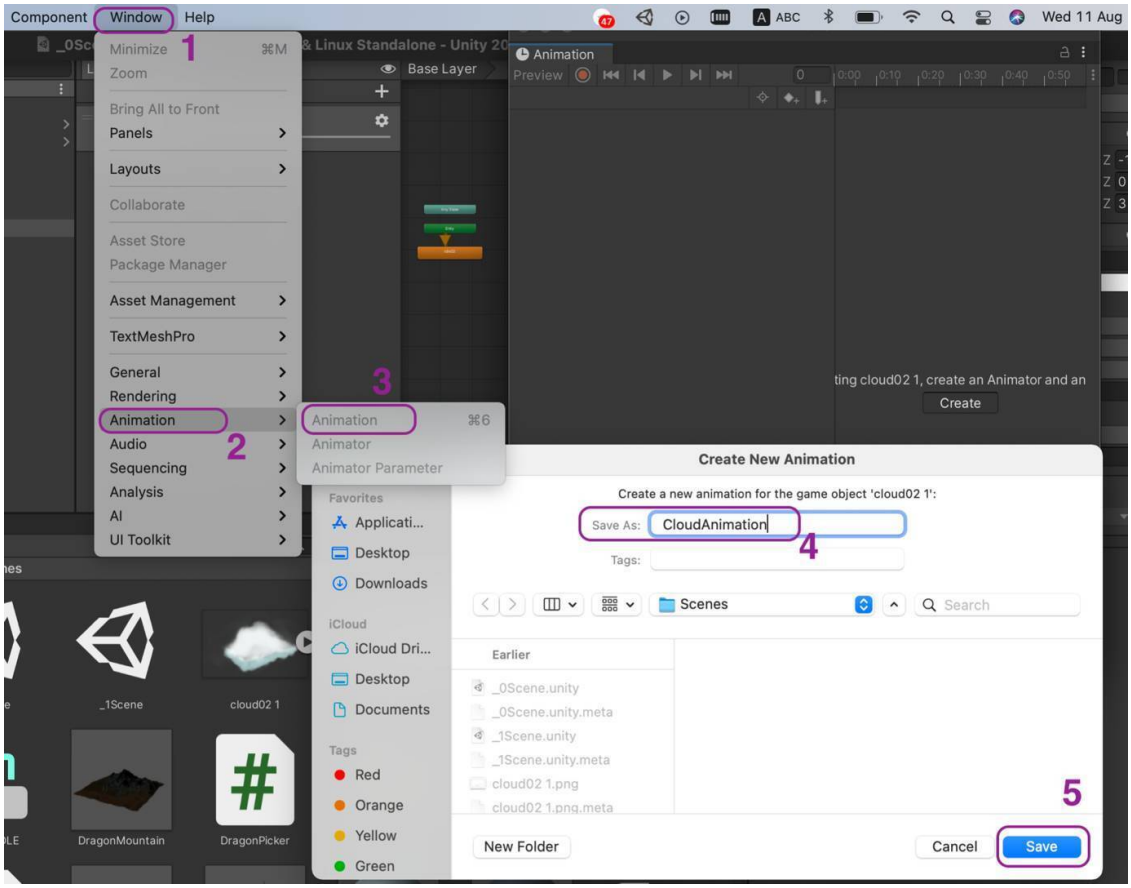


14. Создайте дубликат объекта и переместите его в нашу рабочую папку с проектов Assets / Scenes. После этого перетащите текстуру с облаком в окно Hierarchy так, чтобы оно оказалось внутри холста Canvas (можете при перетаскивании навести объект прямо на Canvas). Разместите объект в левой верхней части сцены (Canvas). Настройки показаны на рисунке ниже:

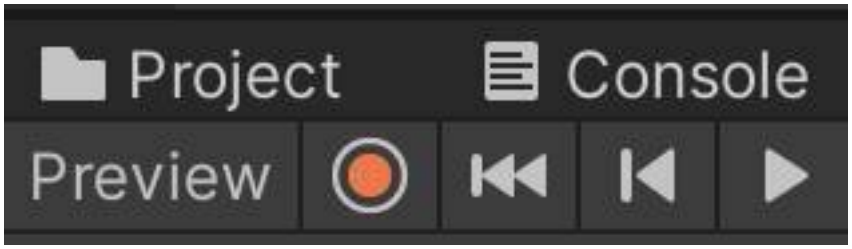


15. Чтобы создать анимацию движения облака, в Unity откройте Windows – Animation – Animation. Так, вы попадете в окно создания анимации. Убедитесь, что в окне иерархии объектов (Hierarchy) выбран игровой объект cloud02 1, после чего в окне Animation нажмите

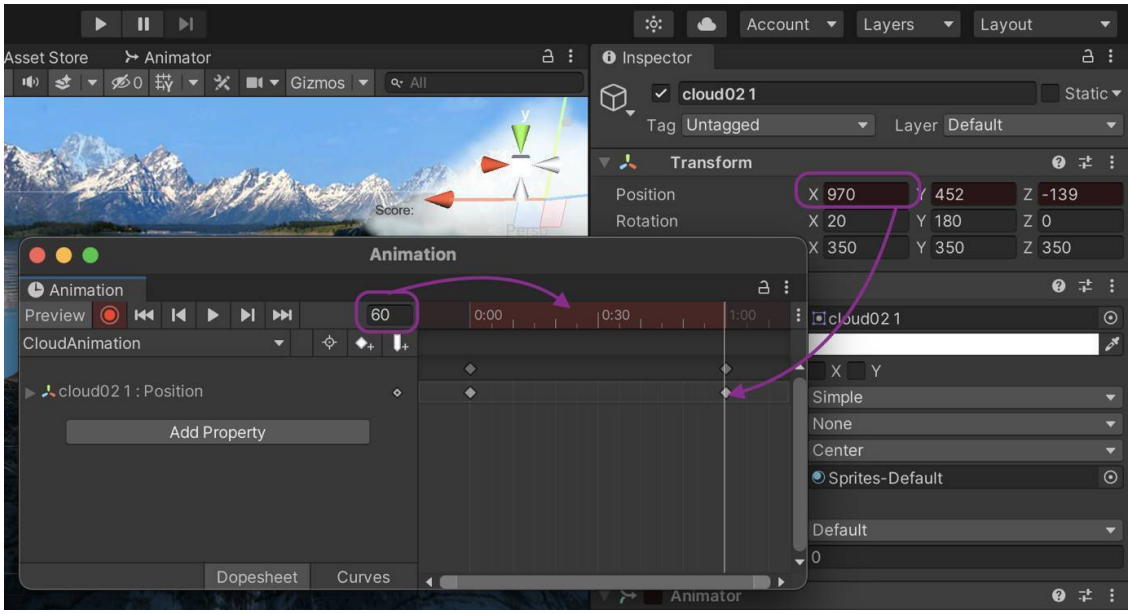
Create, чтобы создать новую анимацию для этого игрового объекта. Задайте имя новой анимации CloudAnimation, рисунок ниже:



16. Теперь мы можем перейти к созданию анимации. В окне Animation нажмите кнопку Enable keyframe recording mode.

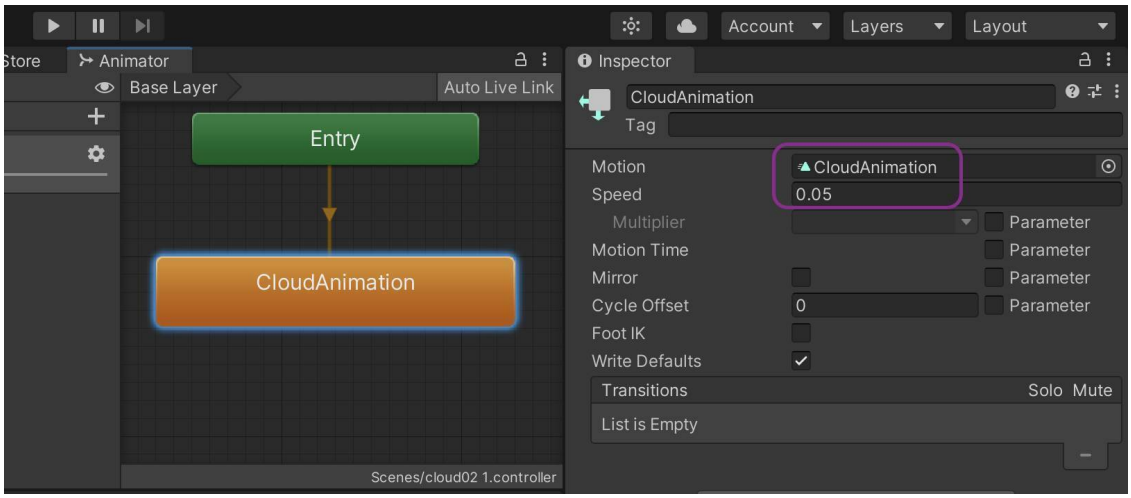


В режиме записи установите сначала начальное положение облака в левой части экрана (Position x = 770), после чего переместите отметку тайминга по шкале на 60 секунд, и установите новое положение (Position x = 970).



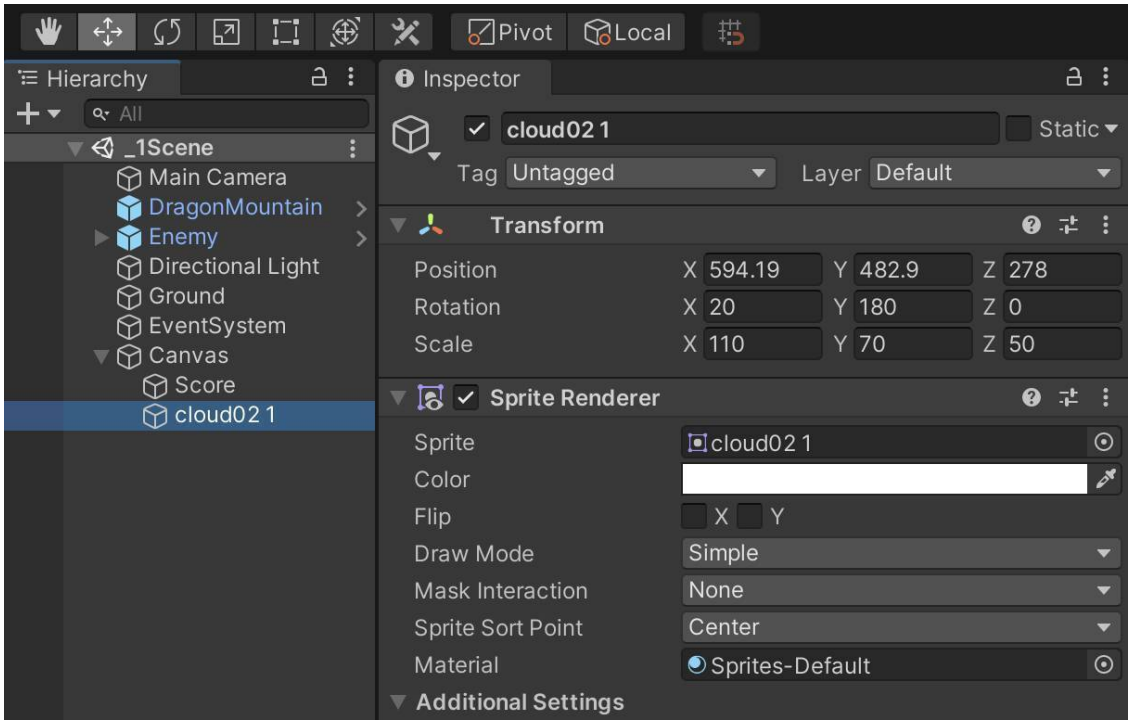
17. После этого остановите запись анимации нажав снова Enable keyframe recording mode. Теперь при запуске облако будет передвигаться из левой части экрана в правую, однако происходит это достаточно быстро. Чтобы это исправить откройте Windows – Animation – Animator. Так вы попадете в дерево анимаций проекта.

18. Выберите анимацию CloudAnimation и в инспекторе подкорректируйте параметр Speed. Сделать это можно двумя способами. В первом варианте чтобы скорость движения облака стала более приемлемой, установите значение speed = 0.05 (см. рисунок ниже). Или же можно показать еще один вариант увеличения времени анимации. Вернитесь в запись анимации и просто переместите «ромбы» с анимацией с 1:00, например на 3:00. Мы же остановимся на первом варианте.

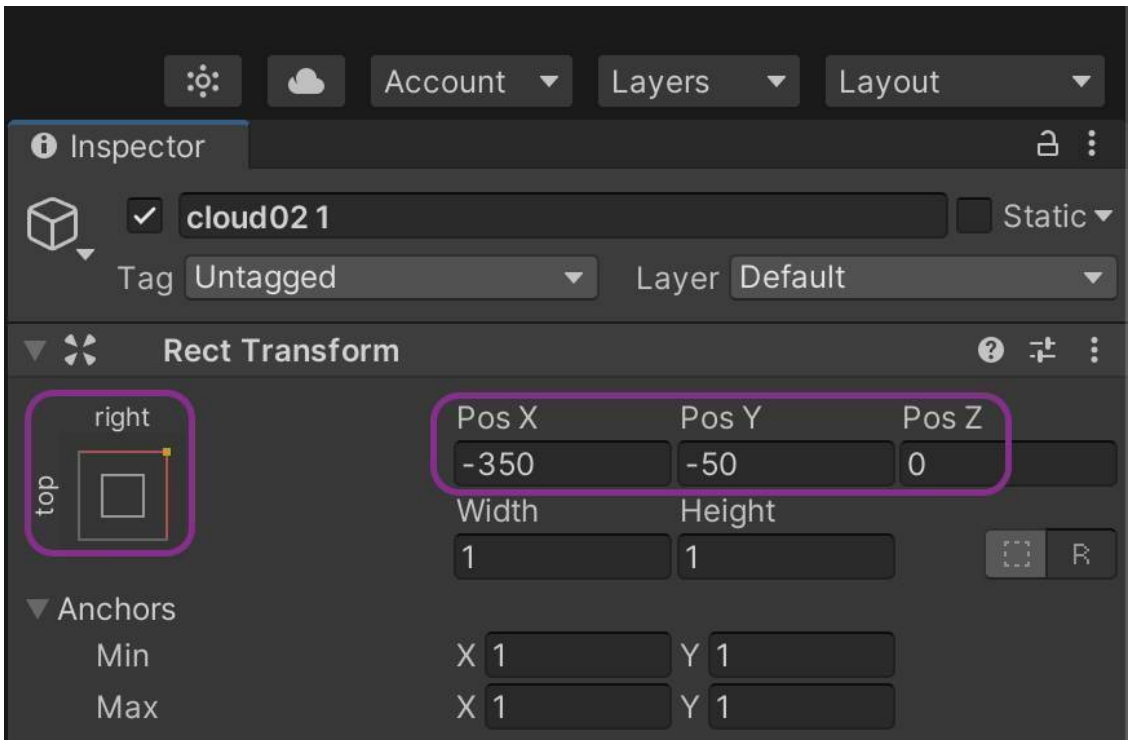


19. Сохраните сцену _0Scene. Запустите ее снова и убедитесь, что облако перемещается с приемлемой скоростью.

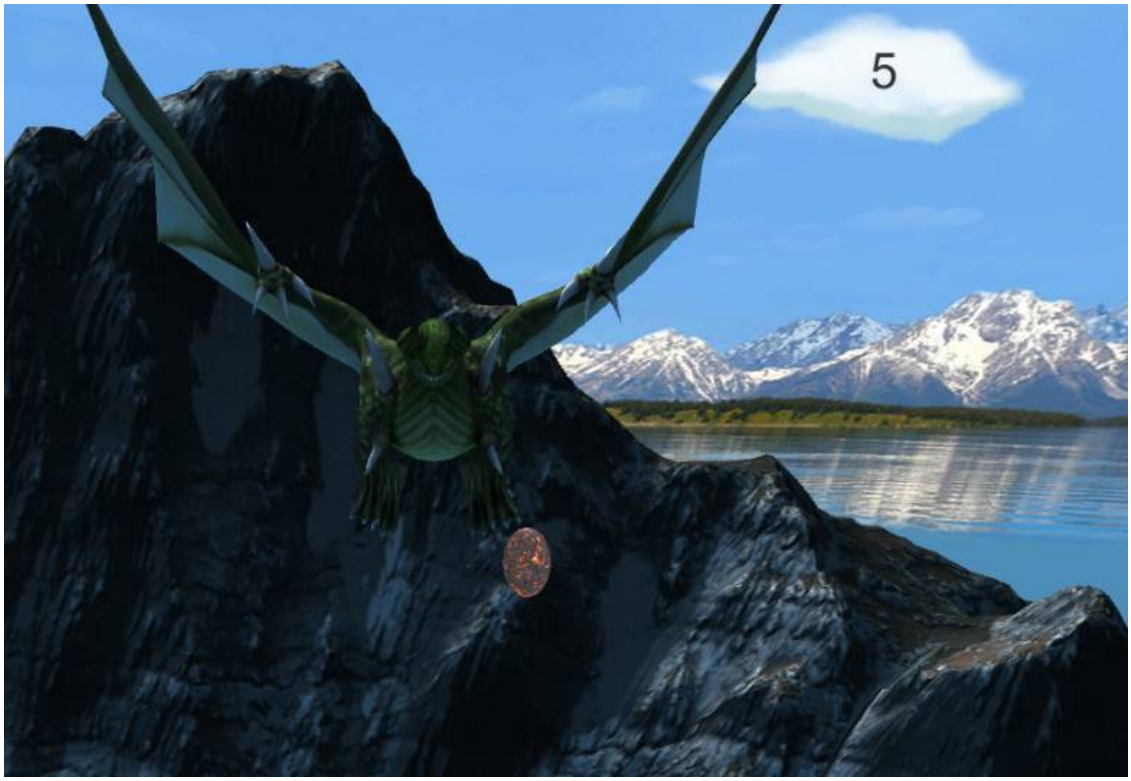
20. Переключаться между сценами и дорабатывать их можно в любой момент. Давайте, к примеру, вернемся к сцене _1Scene (кликните по ней два раза) и поместим в область счетчика очков облако, чтобы цифры было видно более четко. Переместите облако с именем cloud02 1 из папки Asset – Scenes внутрь Canvas:



21. Добавьте объекту с облаком компонент с именем Rect Transform (кнопка Add Component в окне Inspector) и установите параметры якоря, как показано на рисунке ниже. Это позволит привязать объект точно к правой верхней части экрана вне зависимости от разрешения экрана:



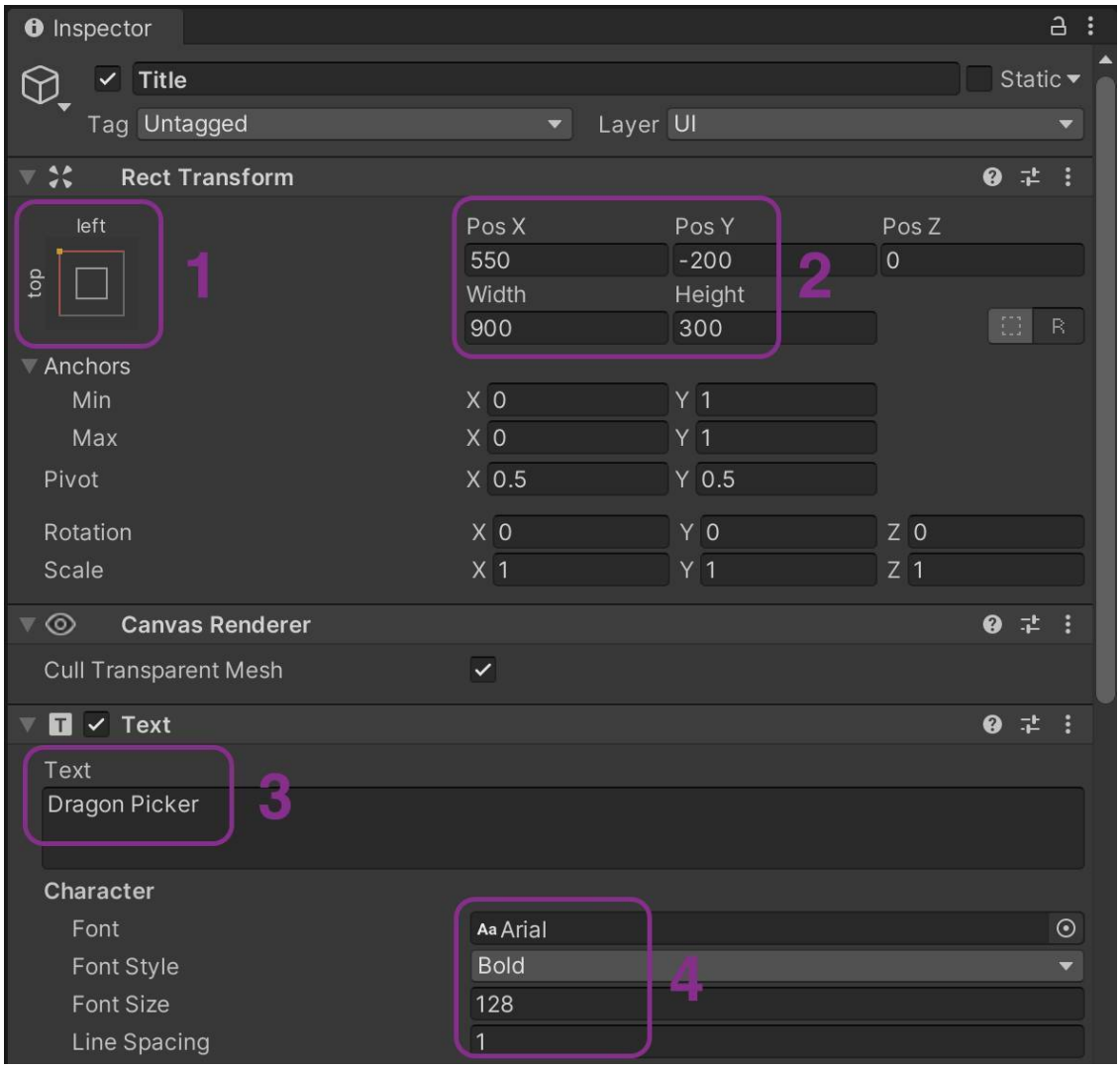
22. Теперь, если запустить сцену снова, то в правой верхней части экрана вы заметите облако под счетчиком очков:



В следующем подразделе мы создадим игровое меню и научимся реализовывать переключение между сценами.

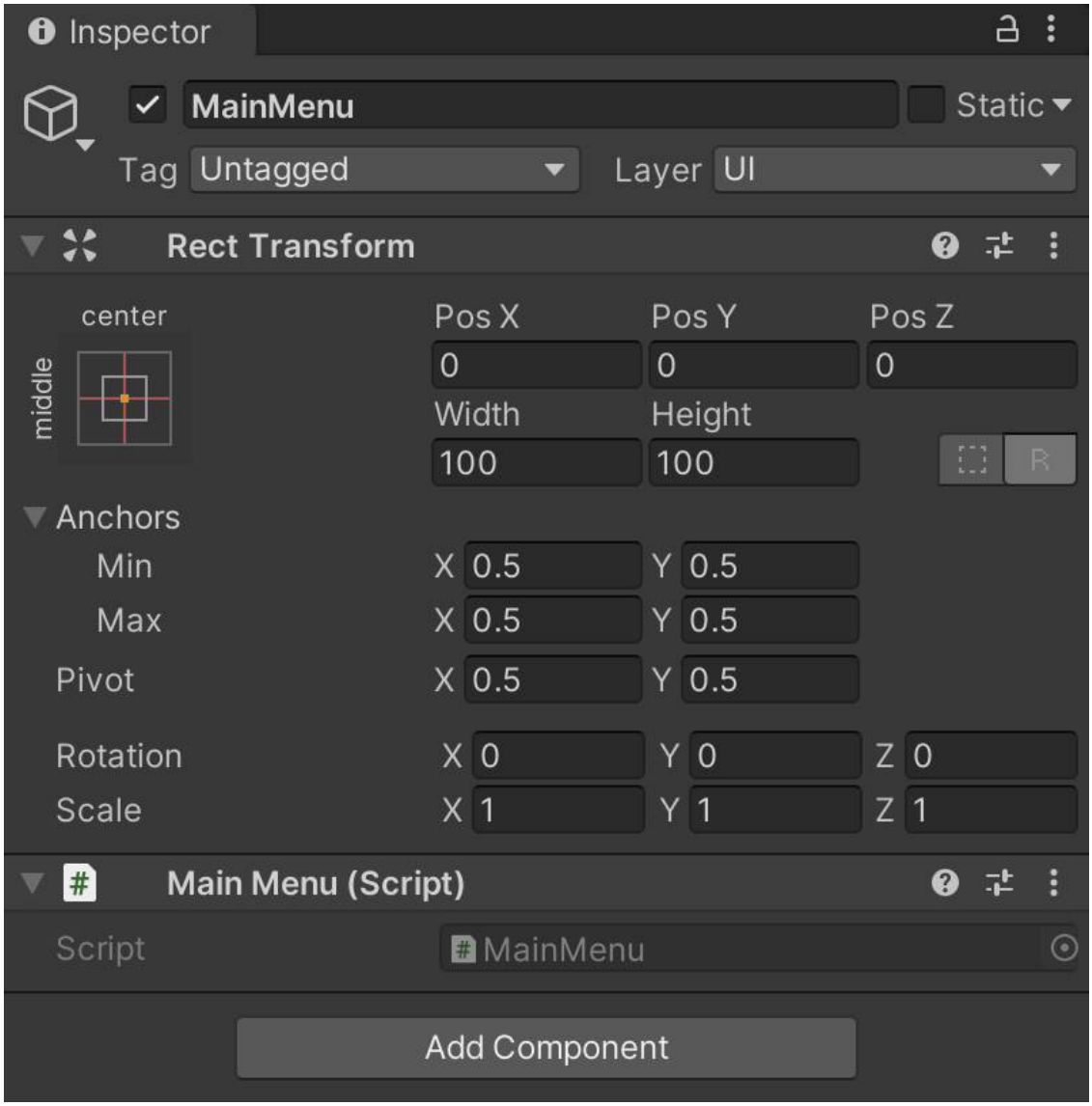
5.3 Создание игрового меню

1. Создадим заголовок на сцене с названием игры. Выберите в верхнем меню GameObject – UI – Text. Дайте имя новому объекту Title и также как делали ранее – переместите его внутрь Canvas (если этого не произошло автоматически). Настройте Title в окне инспектора (Inspector) так, как показано на рисунке ниже:



2. В итоге мы должны получить надпись с названием игры Dragon Picker в верхней левой части экрана. На свое усмотрение вы можете изменить цвета, размер и стиль надписи. Можете даже самостоятельно в Unity Asset Store найти наиболее понравившиеся вам шрифты с текстурами и анимацией.

3. Перейдем к созданию игрового меню. Для создания меню создайте пустой объект, выбрав в верхнем меню GameObject – CreateEmpty, который будет содержать кнопки переключения между сценами. В панели Hierarchy слева появится пустой объект. Назовите его MainMenu, также создайте и подключите к объекту соответствующий скрипт MainMenu.cs, рисунок ниже:



4. Сделайте элемент объект MainMenu дочерним для Canvas.

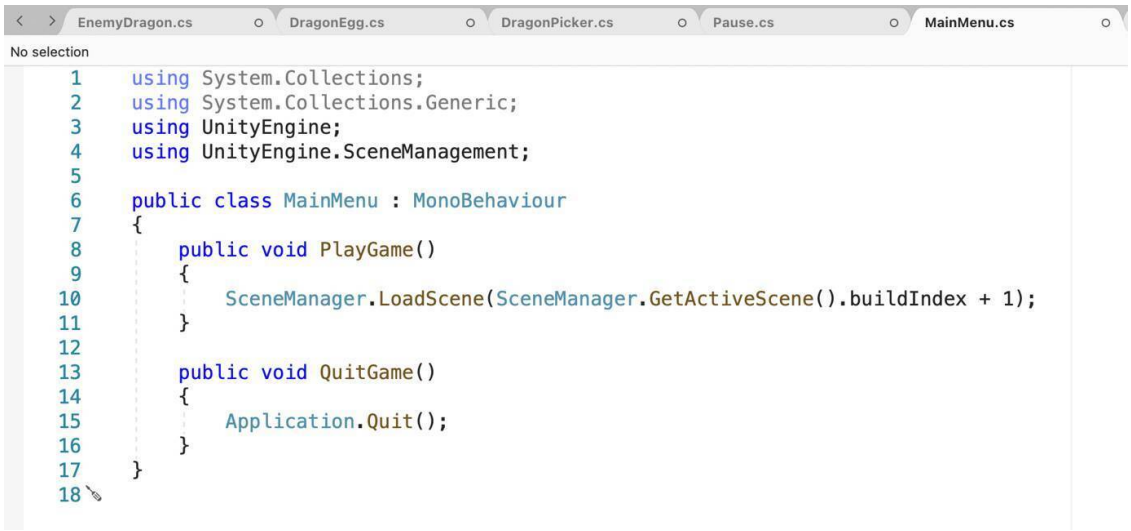
5. Добавьте в скрипт MainMenu.cs следующий код, который будет отвечать за переключение между сценами:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {

  SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
+ 1);
    }
    public void QuitGame()
```

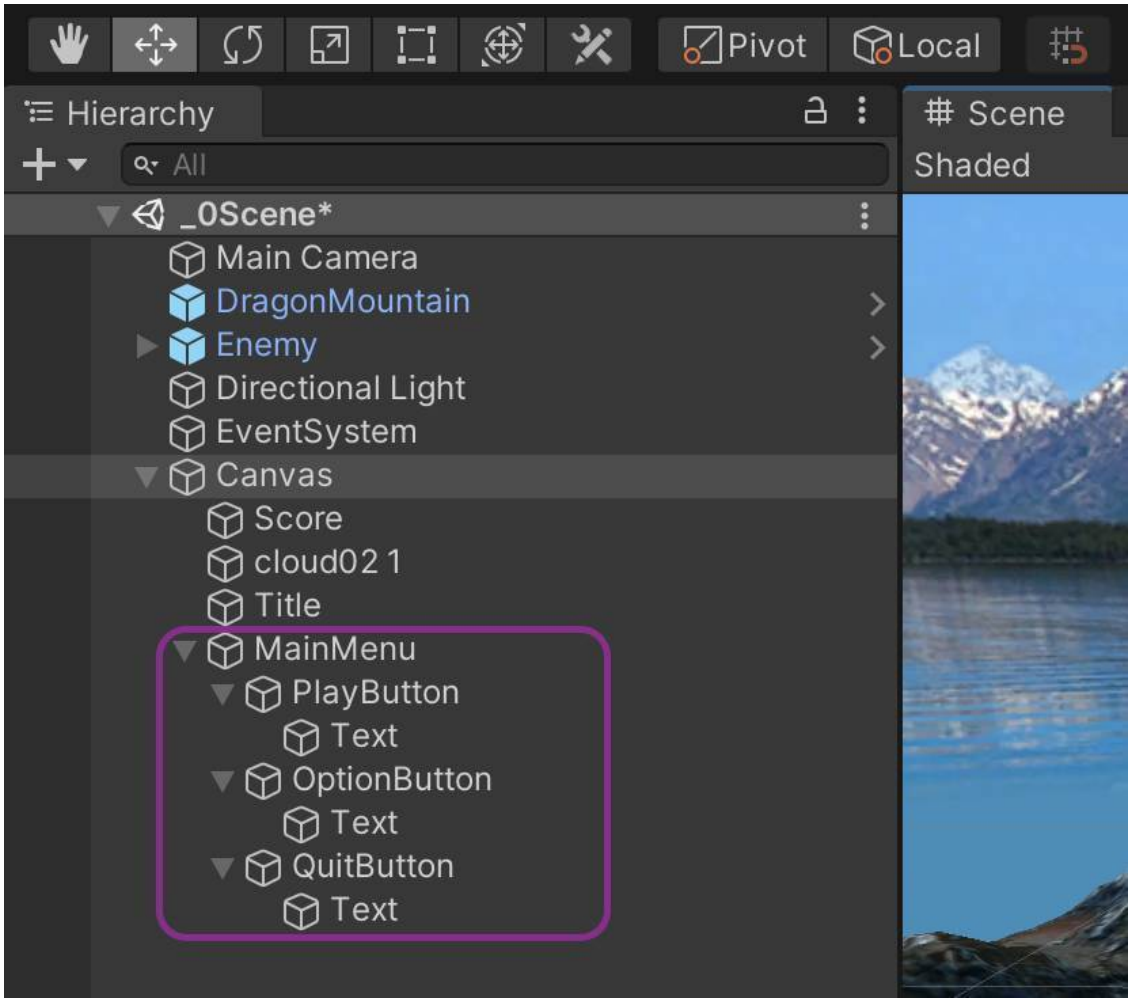
```
        {
            Application.Quit();
        }
    }
```

// End Code

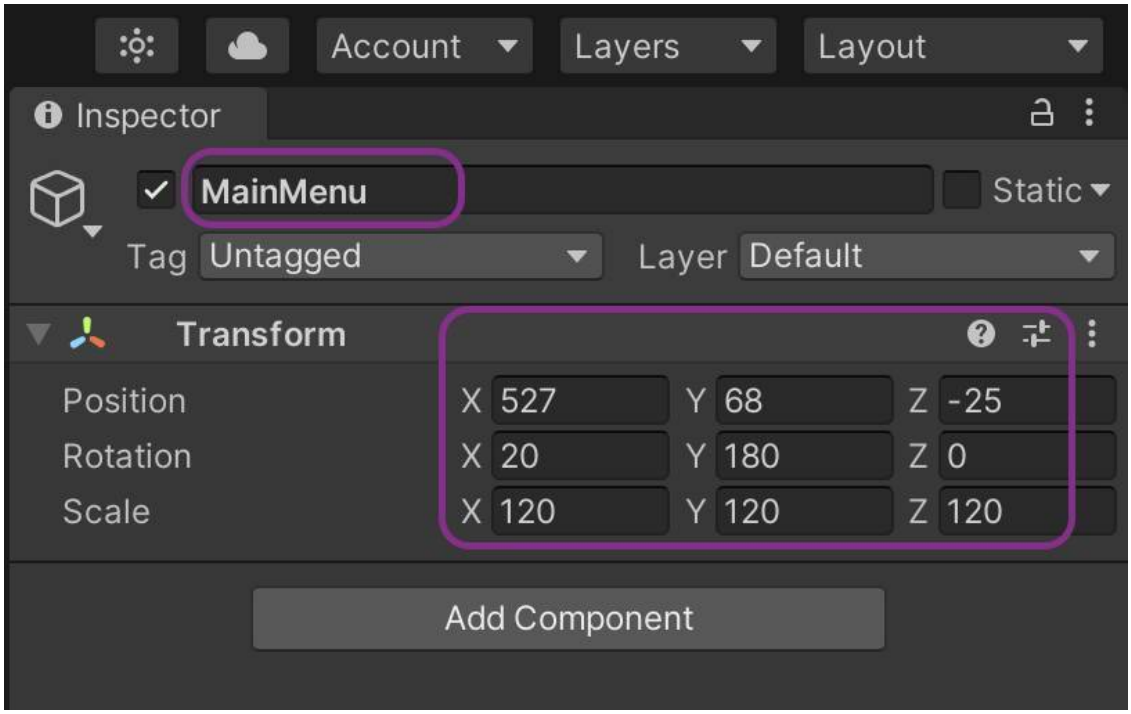


6. Добавьте в панель иерархии (Hierarchy) элементы UI графического интерфейса: GameObject – UI – Button (3 штуки);

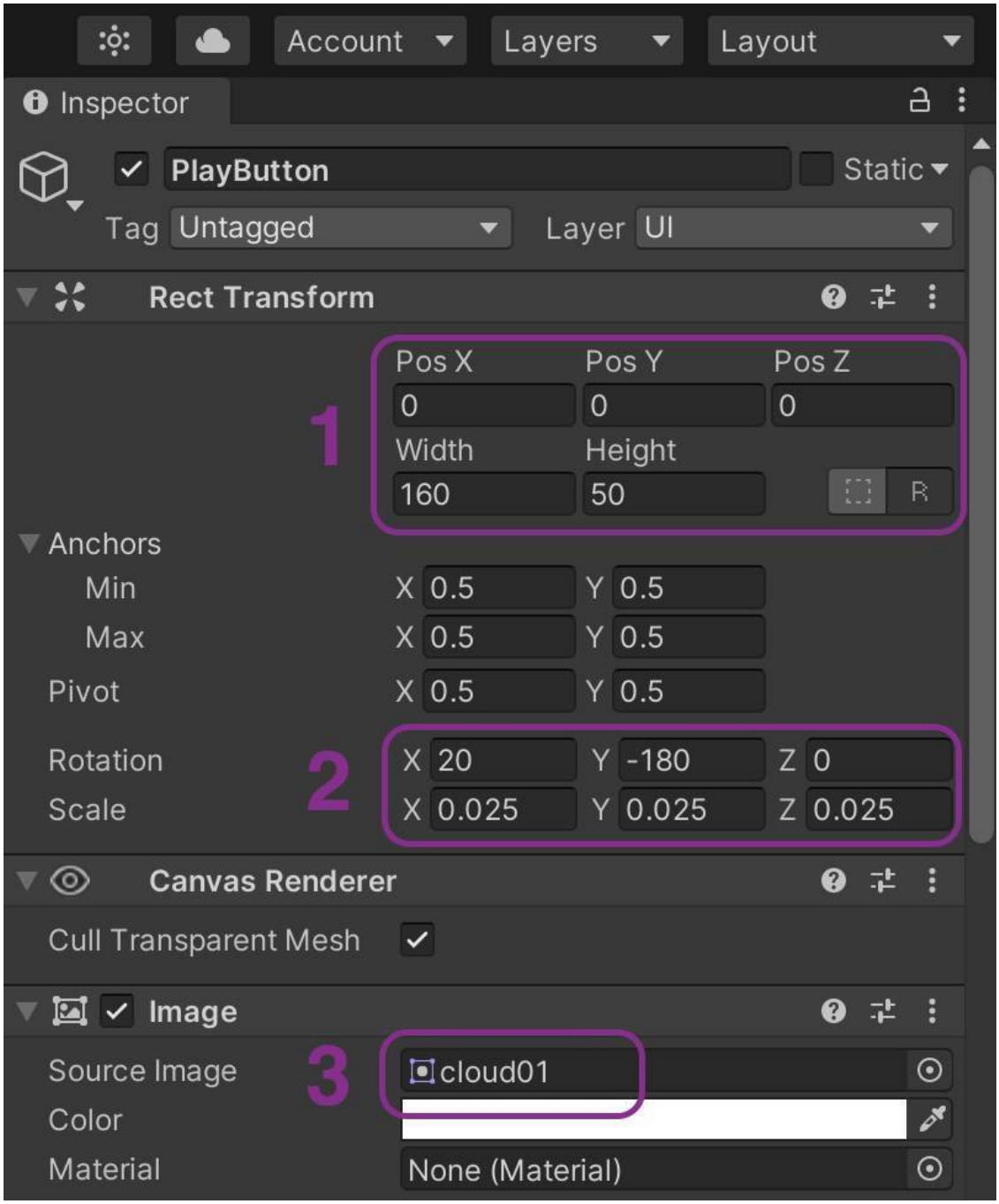
7. Сделайте созданный объект Button дочерним для MainMenu. Обратите внимание, что внутри кнопки Button автоматически создается дочерний элемент Text. Переименуйте созданные кнопки Button в PlayButton, OptionButton и QuitButton. Должна получиться структура кнопок и надписей такая, какая показана на рисунке ниже:



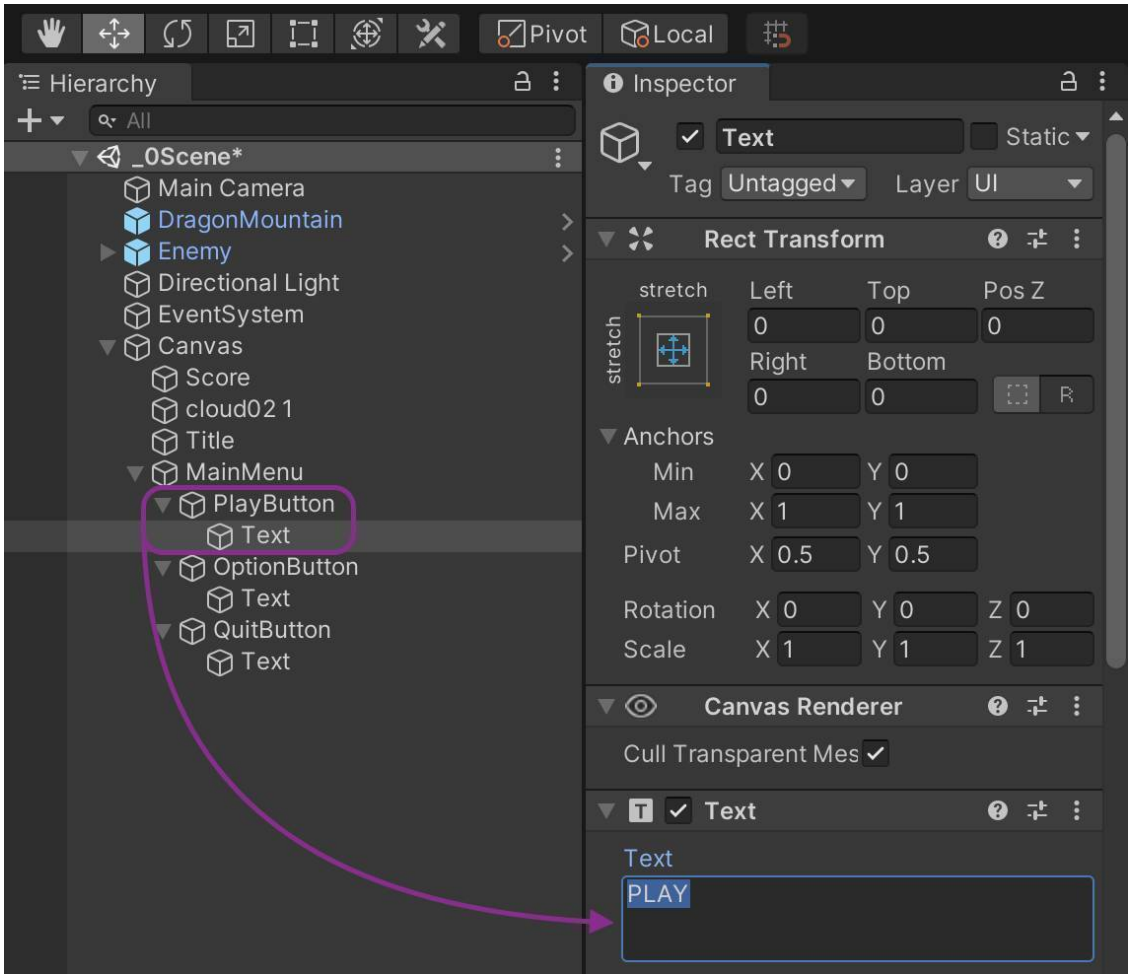
8. Далее мы настроим оформление надписей кнопок и их расположите таким образом, чтобы интерфейс выглядел как стандартное меню с элементами управления. Кнопки имеют стандартные настройки, которые задают их расположение, внешний вид при различных вариантах срабатывания и т. д. В окне Hierarchy выберите элемент MainMenu и установите его параметры Transform, как указано ниже:



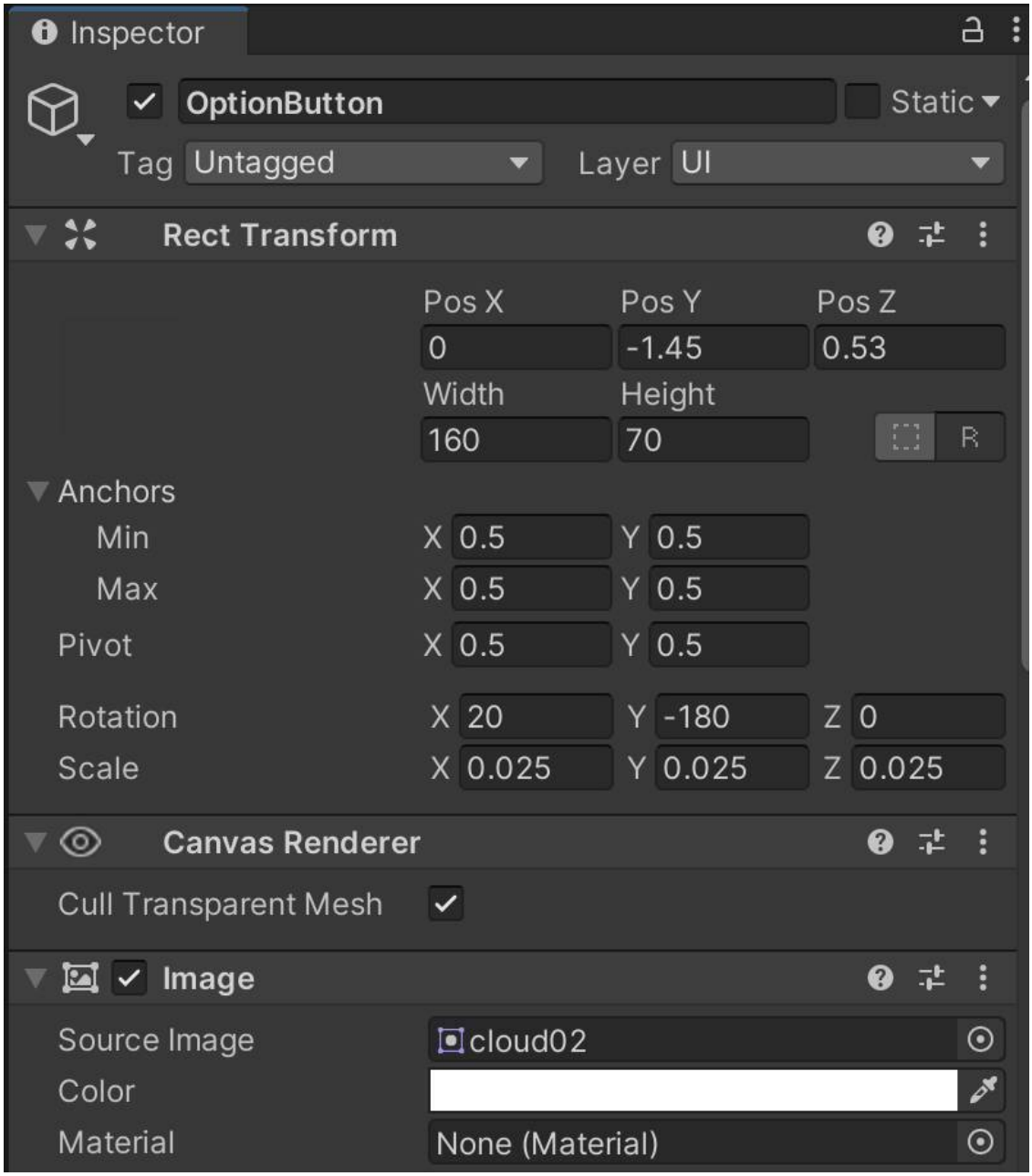
9. Далее настроим кнопки. Начнем с PlayButton. Важные настройки кнопки находятся во вкладке Button в окне Inspector, эти настройки можно использовать при создании различных эффектов анимации кнопки без дополнительных текстур. Некоторые настройки отвечают за внешний вид при нажатии/наведении/выборе и т. д. Мы будем использовать текстуру облака в качестве оформления кнопки. Настройте кнопку Button так, как показано на рисунке ниже:



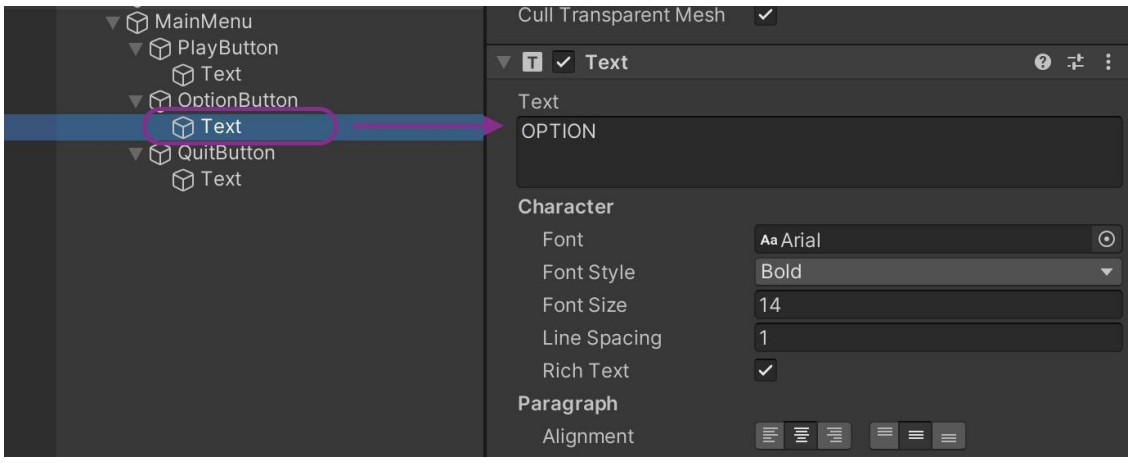
10. Чтобы изменить надпись на кнопке, разверните элемент PlayButton, кликните по элементу Text внутри. Далее в окне Inspector (для удобства его положение на скриншоте ниже было изменено) найдите компонент Text и напишите в поле ввода текста – PLAY:



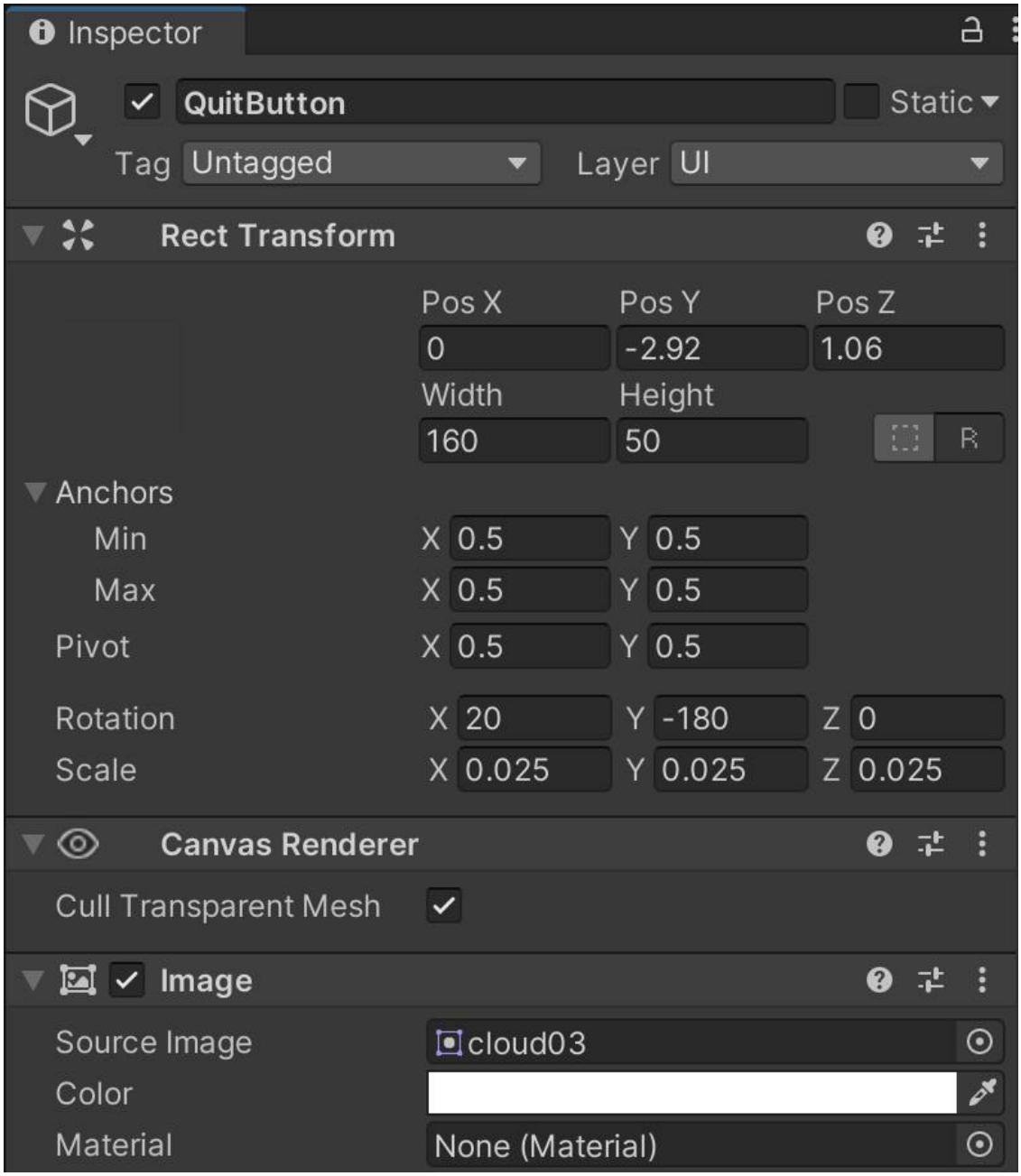
11. Аналогично ниже приведены настройки для кнопки OptionButton:



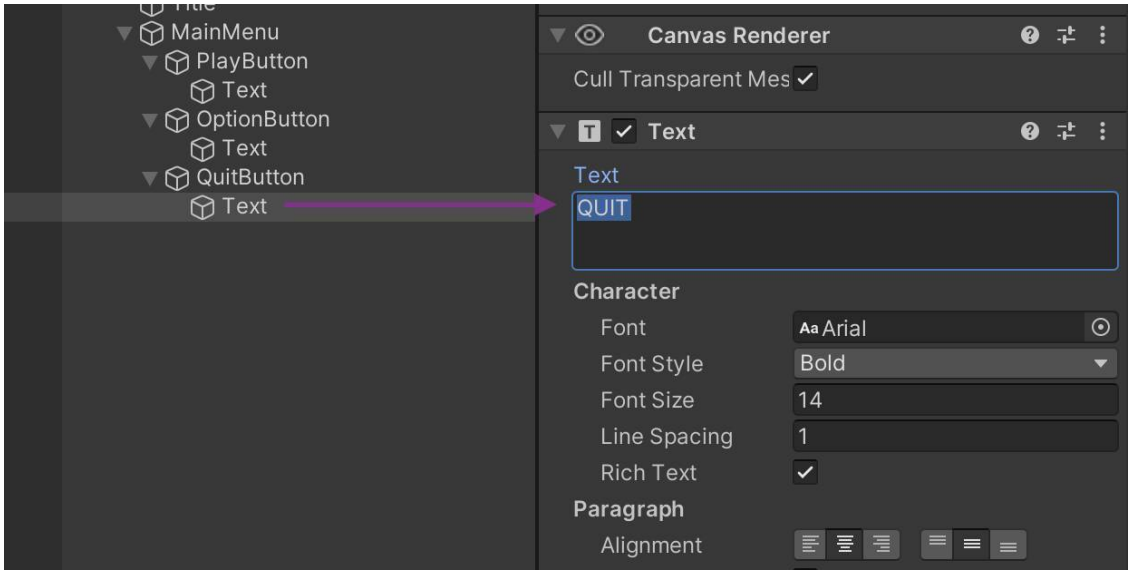
12. Текстовое поле Text внутри OptionButton:



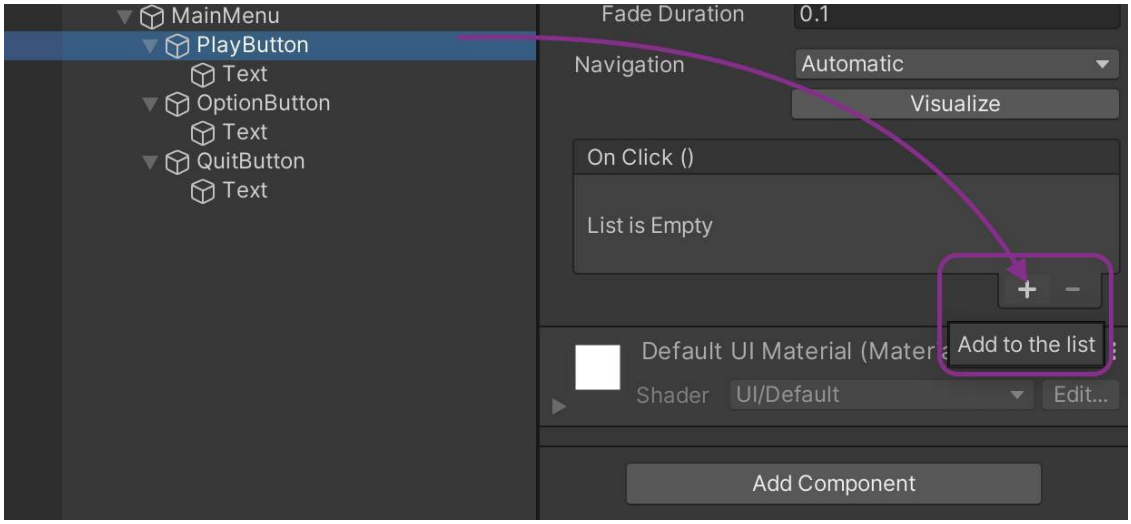
13. Аналогично ниже приведены настройки для кнопки QuitButton:



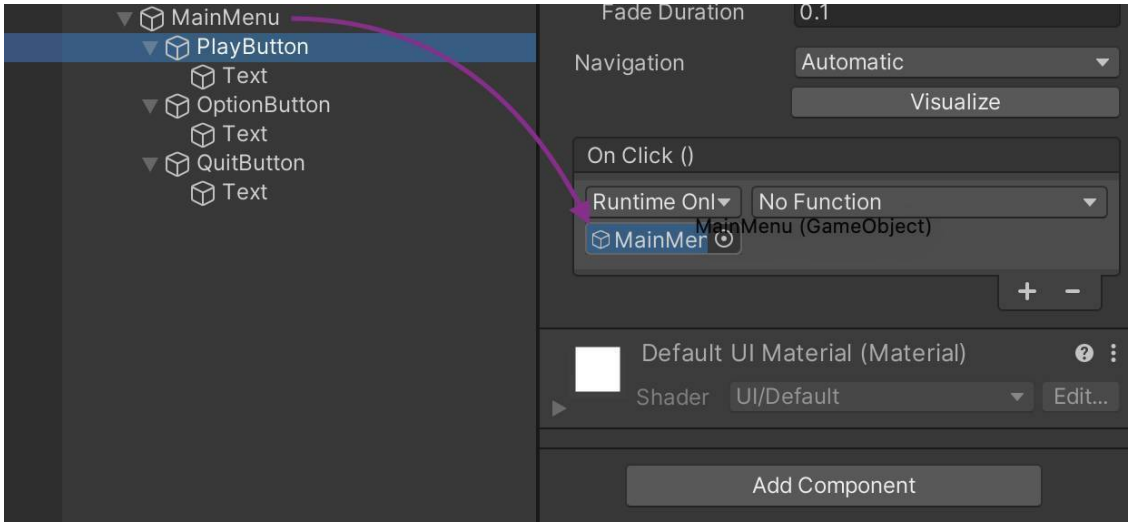
14. Текстовое поле Text внутри QuitButton:



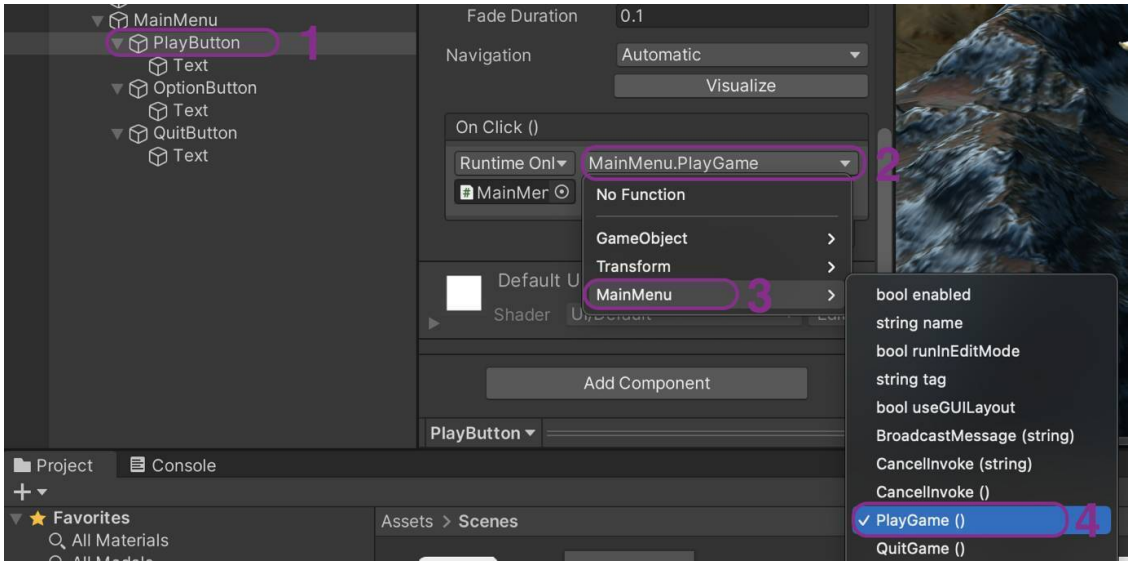
15. За функционал кнопки отвечает нижняя часть компонента Button внутри окна инспектора (Inspector) – On Click(). Выберите кнопку PlayButton. Чтобы настроить действие, которое будет происходить при срабатывании кнопки, нажмите знак “+” в поле On Click ():



16. В появившееся пустое поле для игрового объекта – объект из иерархии (Hierarchy) с именем MainMenu. Таким образом, мы как-бы сообщаем программе, что при нажатии должна вызываться некоторая часть скрипта, привязанная к прикрепленному игровому объекту:

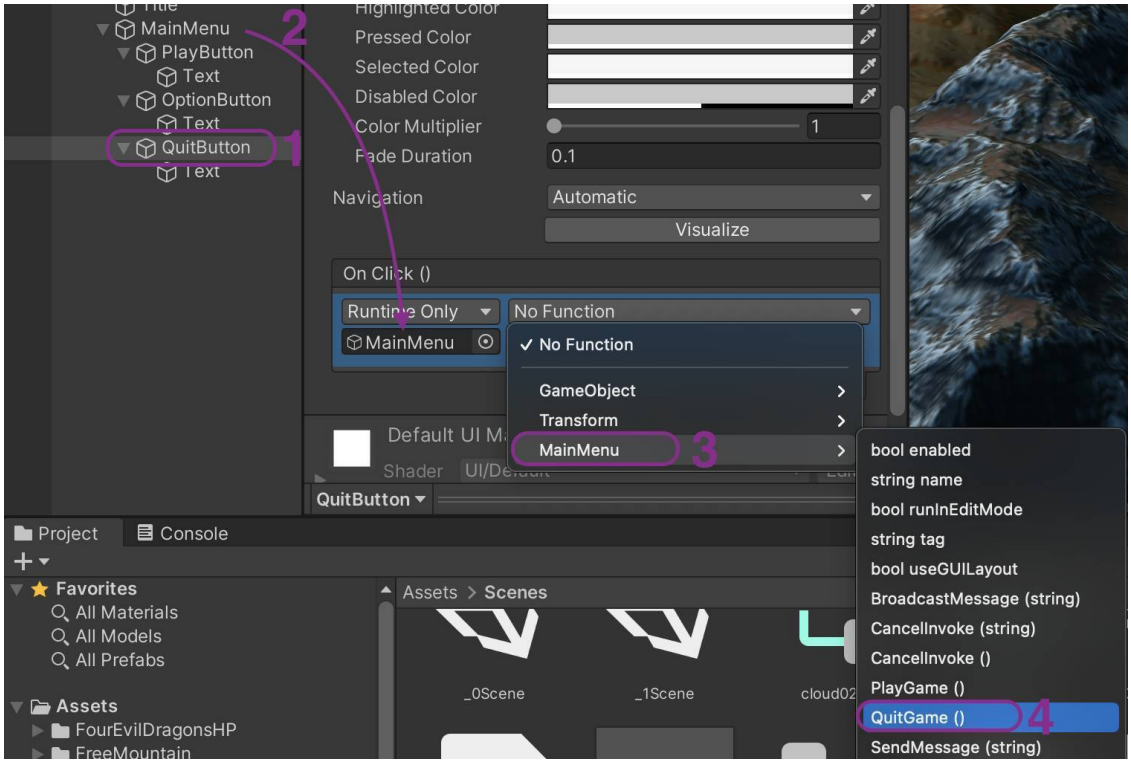


17. Если все сделано правильно, то теперь из вкладки No Functions стало возможным выбрать скрипт MainMenu и метод PlayGame, отвечающий за запуск игры, см. рисунок ниже:

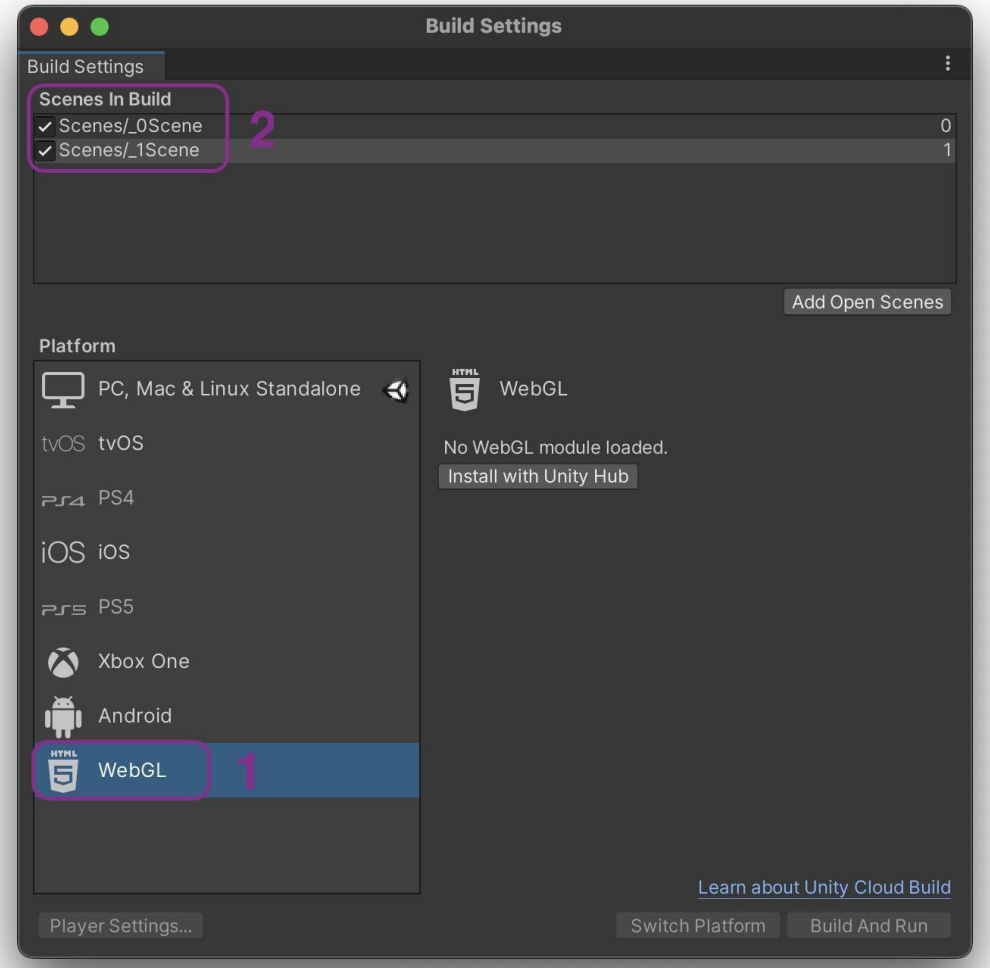


18. Теперь, при нажатии кнопки PlayGame будет происходить запуск игры. Но перед этим нам нужно будет настроить сборку сцен, мы сделаем это буквально через пару пунктов.

19. Настройте кнопку QuitButton. Также нужно добавить событие (знак “+”), переместить в пустое поле объект MainMenu, и в поле No Function выбрать метод QuitGame() из скрипта Main Menu:



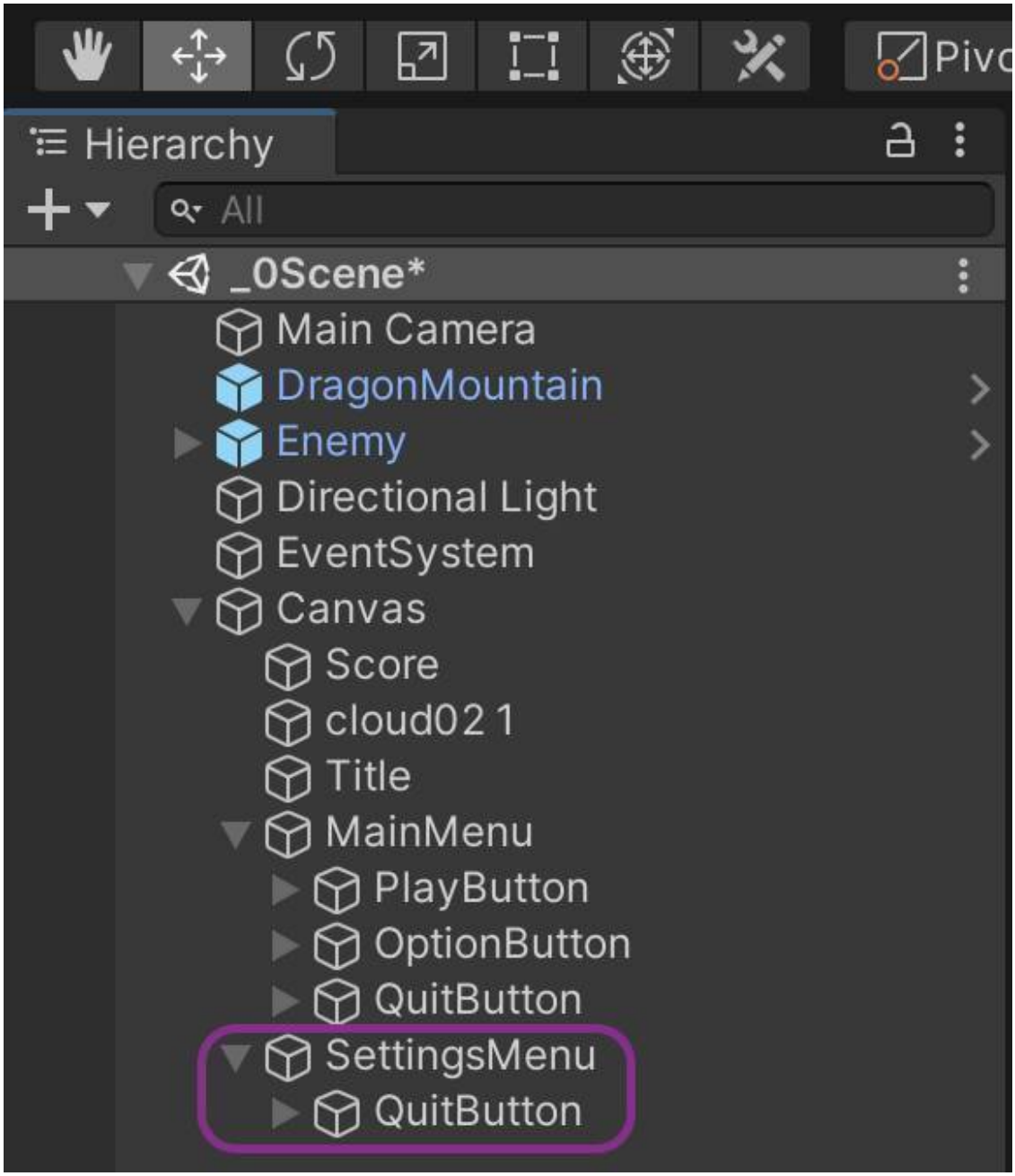
20. Теперь для того, чтобы переключение сцен работало корректно, нужно настроить Build игры и добавить созданные сцены в сборку. Нажмите File – Build Settings, перетащите созданные сцены _0Scene и _1Scene в поле Scenes In Build в порядке их загрузки (первой загружается сцена с меню), см. рисунок ниже:

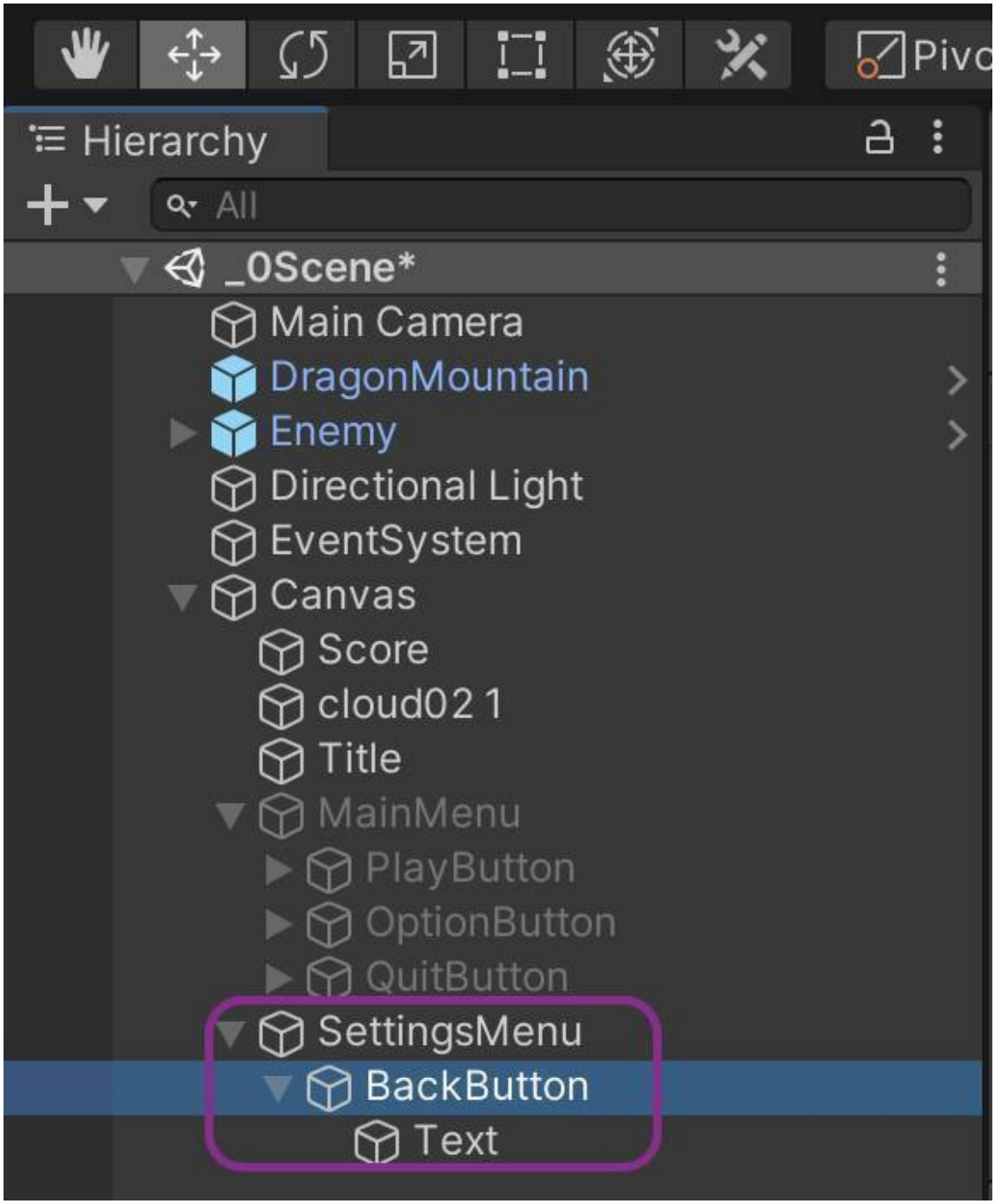


21. После этого можете закрыть окно Build Settings и проверить корректность работы меню, нажав кнопку Run. После нажатия кнопки Play игры должна запускаться, а при нажатии кнопки Quit – завершить работу (однако в рамках симуляции в Unity она может не отрабатывать).

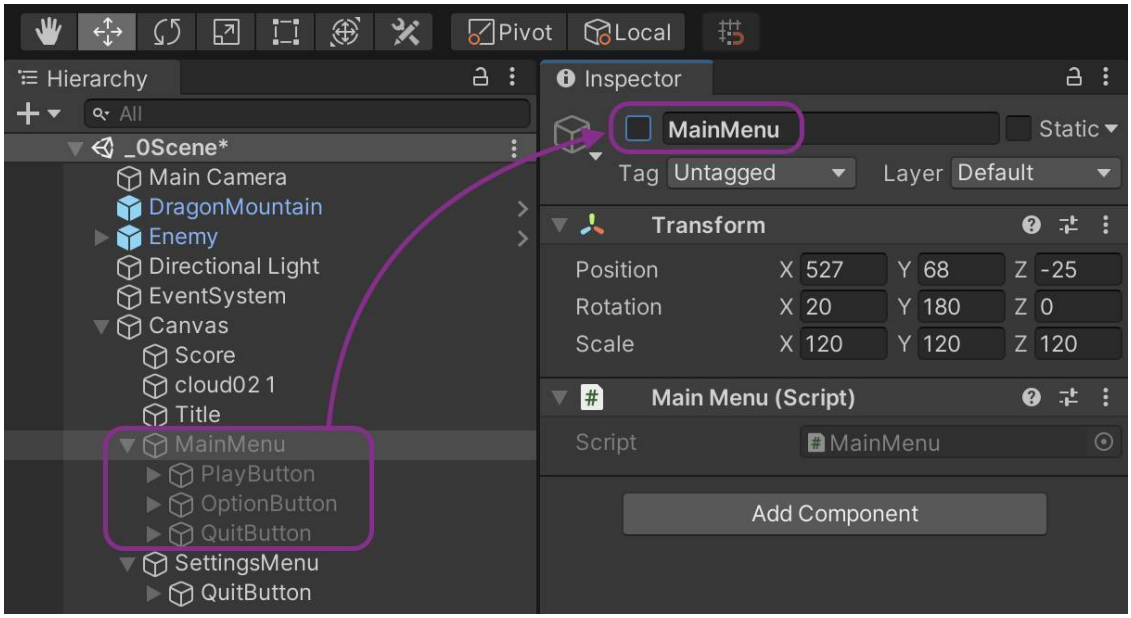
22. Вернемся к кнопке Option. Перед тем как запрограммировать кнопку Option, нужно создать дополнительные элементы меню. При нажатии на кнопку Option у нас будут появляться дополнительные элементы меню и скрываться элементы главного меню MainMenu.

23. Самостоятельно создайте дополнительное меню с названием SettingsMenu (можно создать дубликат MainMenu) и удалите из SettingsMenu две верхние кнопки PlayButton и OptionButton. Единственную оставшуюся кнопку переименуйте в BackButton (вернуться назад).


Hierarchy
All
_0Scene*
Main Camera
DragonMountain
Enemy
Directional Light
EventSystem
Canvas
Score
cloud02 1
Title
MainMenu
PlayButton
OptionButton
QuitButton
SettingsMenu
QuitButton



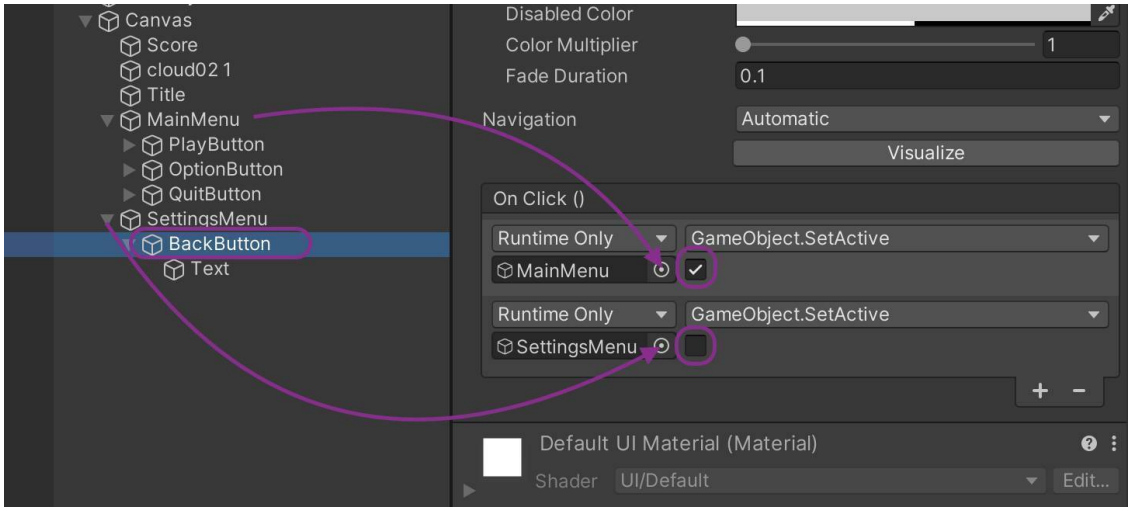
24. Чтобы объект главного меню MainMenu не мешался при создании нового SettingsMenu в том же окне, вы можете его временно отключить, сняв галочку напротив названия объекта в окне Inspector:



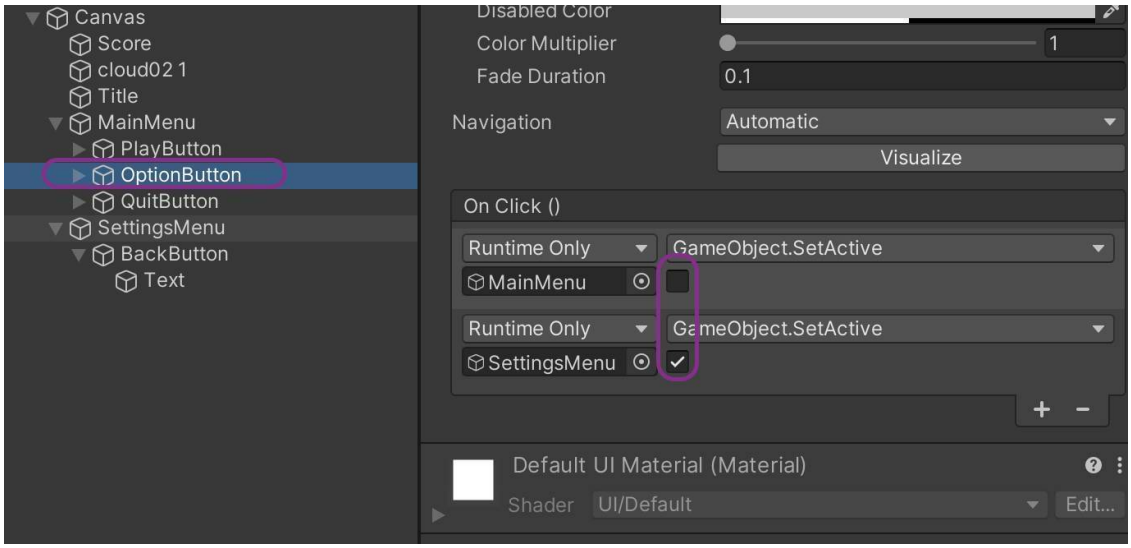
25. В поле объекта Text внутри кнопки BackButton напишите “BACK”, – надпись, которая будет отображаться в меню.

26. Созданное SettingsMenu должно выглядеть следующим образом:

27. Для программирования кнопок BackButton и OptionButton не обязательно писать дополнительный код. Давайте рассмотрим реализацию переключения элементов меню, просто настроив активацию игровых объектов при нажатии кнопок. Выберите кнопку BackButton и в окне Inspector настройте ее параметры On Click():, как показано на рисунке ниже:



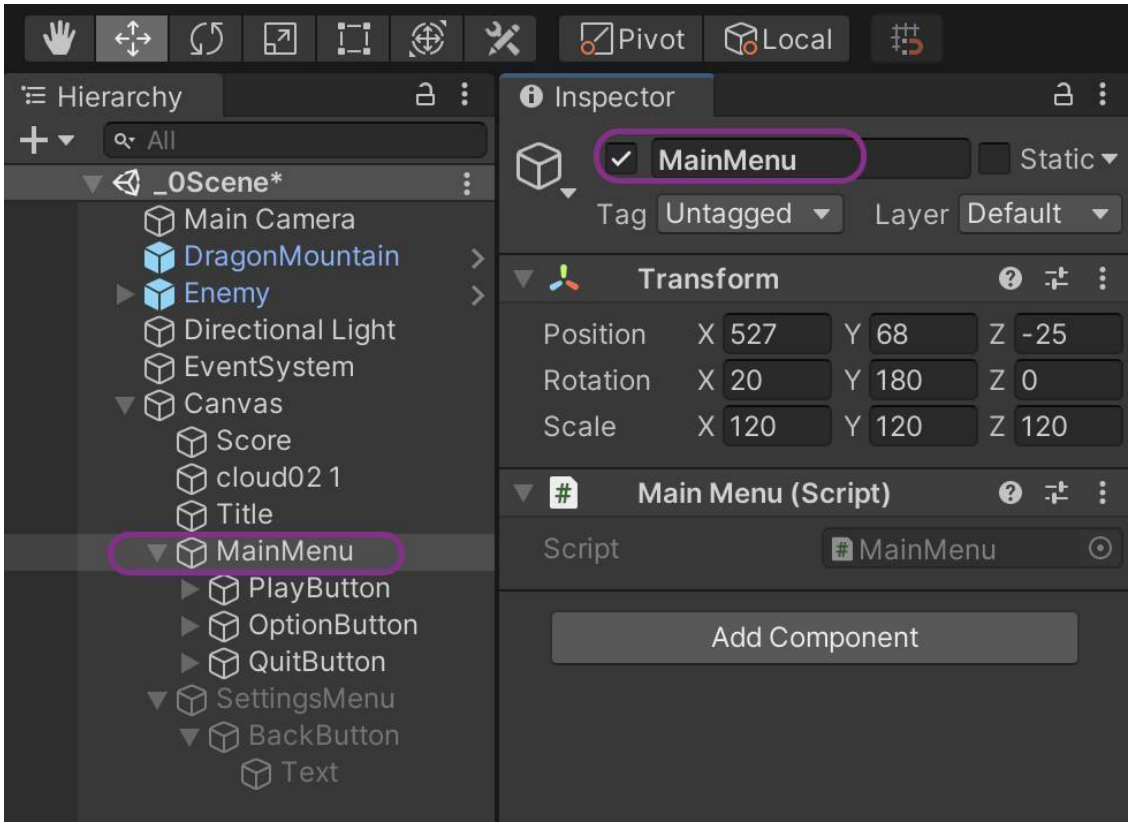
28. Для OptionButton настройка параметра On Click(): показана на рисунке ниже:



29. Чтобы интерфейс работал корректно, при старте игры меню MainMenu должно быть активировано, а SettingMenu – деактивировано. Сделайте так, чтобы галочка в окне Inspector:

– напротив имени MainMenu была установлена;

– напротив имени SettingMenu была снята.


Pivot
Local
Hierarchy
All
_0Scene*
Main Camera
DragonMountain
Enemy
Directional Light
EventSystem
Canvas
Score
cloud02 1
Title
MainMenu
PlayButton
OptionButton
QuitButton
SettingsMenu
BackButton
Text
Inspector
MainMenu
Static
Tag
Untagged
Layer
Default
Transform
Position
X 527
Y 68
Z -25
Rotation
X 20
Y 180
Z 0
Scale
X 120
Y 120
Z 120
Main Menu (Script)
Script
MainMenu
Add Component

5.4 Пауза и возвращение в главное меню

Кнопка паузы будет работать следующим образом:
– При нажатии клавиши пробел (Space) игра будет останавливаться,
– На экране будет появляться надпись PAUSE,
– При повторном клавиши Space, игра продолжится,
– На экране пропадет надпись PAUSE,
– При нажатии клавиши Backspace (стрелка назад), происходит загрузка стартовой сцены.

Для того, чтобы реализовать описанный функционал, нам потребуется скрипт-файл, в котором будет описана работа сцены и условия появления надписи PAUSE. А также текстовое поле с надписью PAUSE. Перейдем к реализации.

1. Откройте сцену с игрой _1Scene.

2. Создайте скрипт-файл с именем Pause.cs, откройте его в редакторе кода и напишите, указанный ниже:

// Start Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Pause : MonoBehaviour
{
    private bool paused = false;
    public GameObject panel;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (!paused)
            {
                Time.timeScale = 0;
                paused = true;
                panel.SetActive(true);
            }
            else
            {
                Time.timeScale = 1;
                paused = false;
                panel.SetActive(false);
            }
        }
        if (Input.GetKeyDown(KeyCode.Escape))
        {
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex
- 1);
            }
    }
```

```
}
```

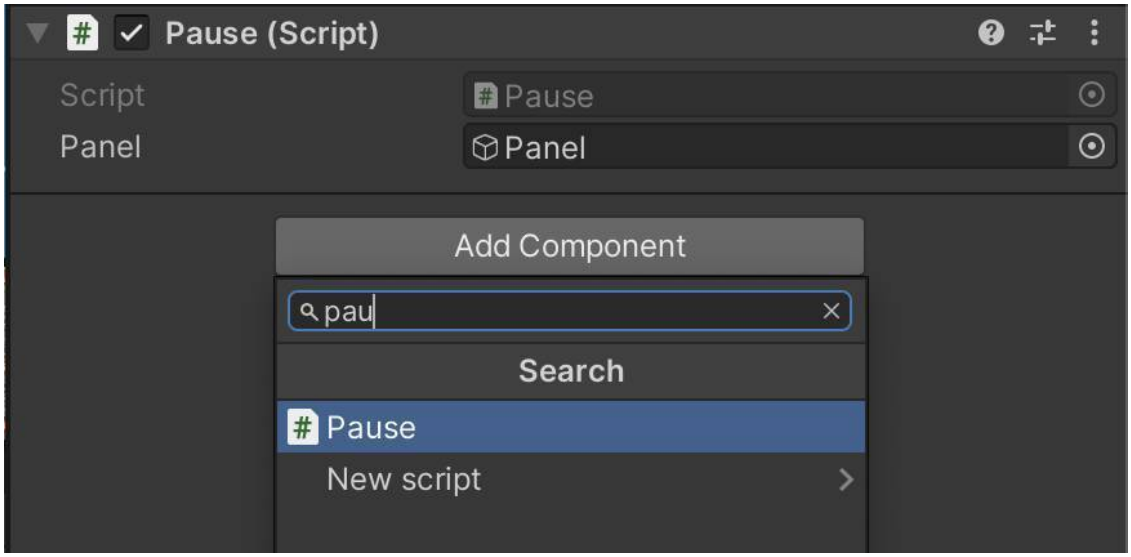
// End Code

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | Pause.cs

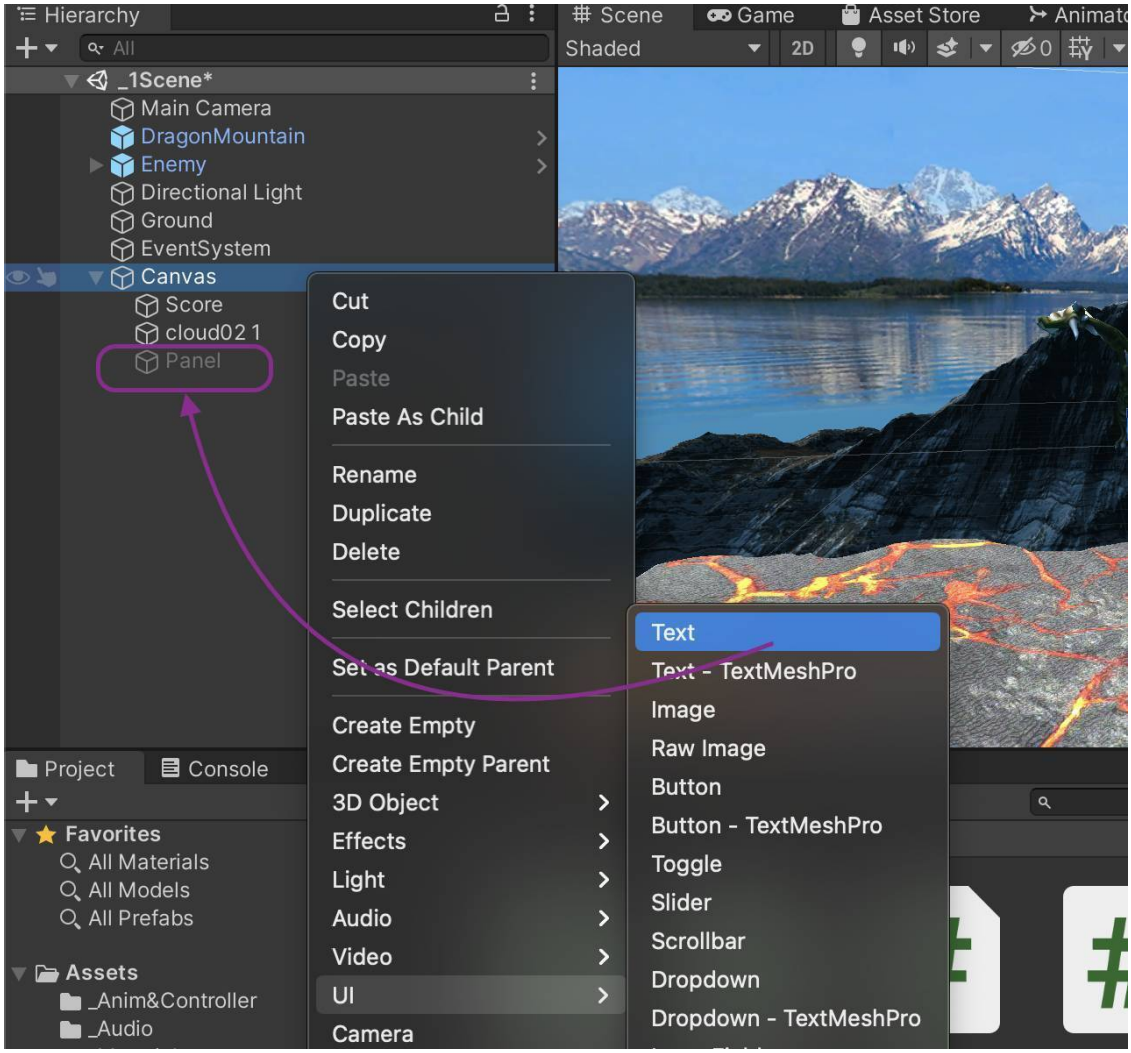
No selection

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

public class Pause : MonoBehaviour
{
    private bool paused = false;
    public GameObject panel;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (!paused)
            {
                Time.timeScale = 0;
                paused = true;
                panel.SetActive(true);
            }
            else
            {
                Time.timeScale = 1;
                paused = false;
                panel.SetActive(false);
            }
        }

        if (Input.GetKeyDown(KeyCode.Escape))
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
        }
    }
}
```

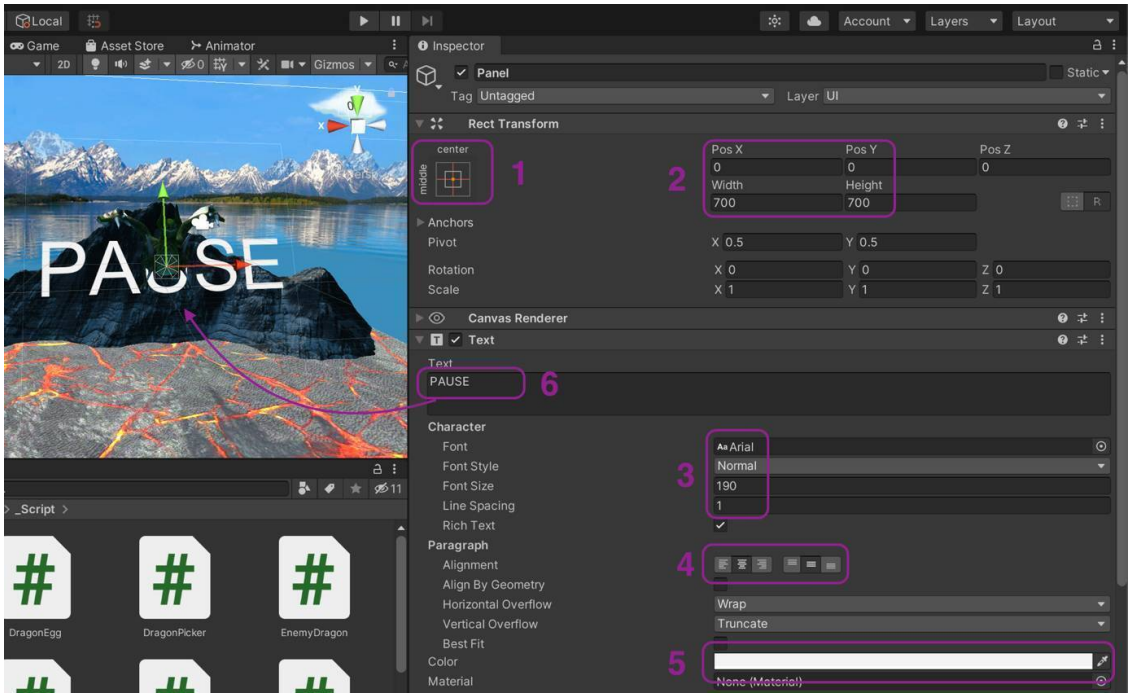
3. Сохраните скрипт-файл и подключите его к объекту на сцене – Main Camera (перетаскиванием как делали ранее, либо выбрав камеру и через кнопку Add Component в окне Inspector):



4. Далее добавим объект текстового типа (Text) и переименуем его в Panel:

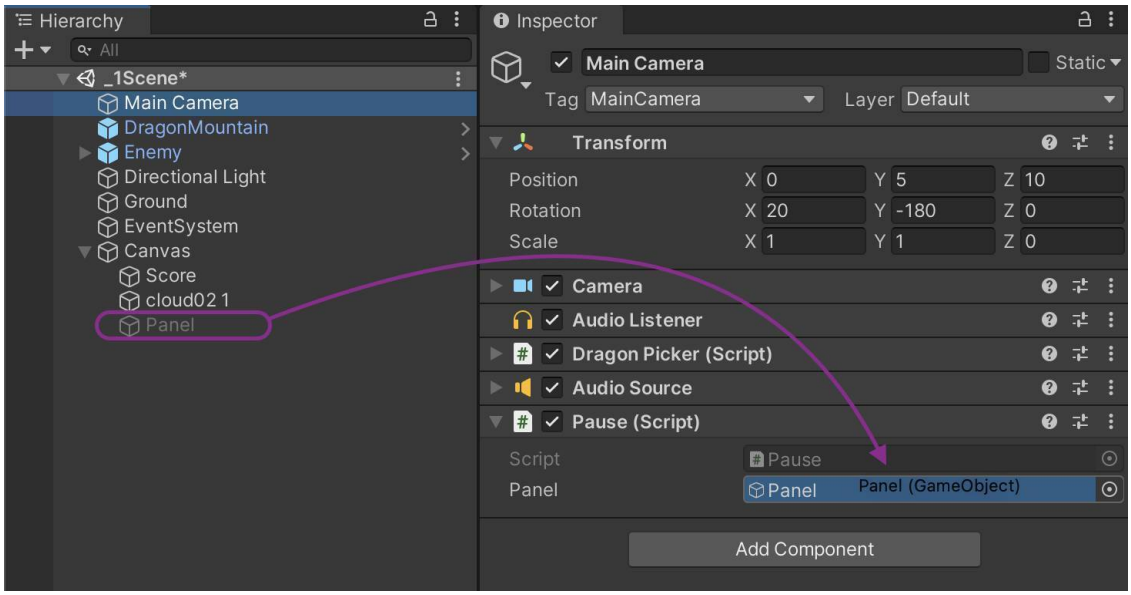


5. Ниже показаны настройки расположения панели Panel (1,2) и размера/ориентации текста (3 – 6).



6. Уберите галочку напротив имени Panel в окне Inspector, чтобы при старте сцены надпись не была активна и не отображалась на экране.

7. Выберите объект Main Camera и переместите в окно переменной Panel в окне Inspector текстовый объект Panel из окна Hierarchy:



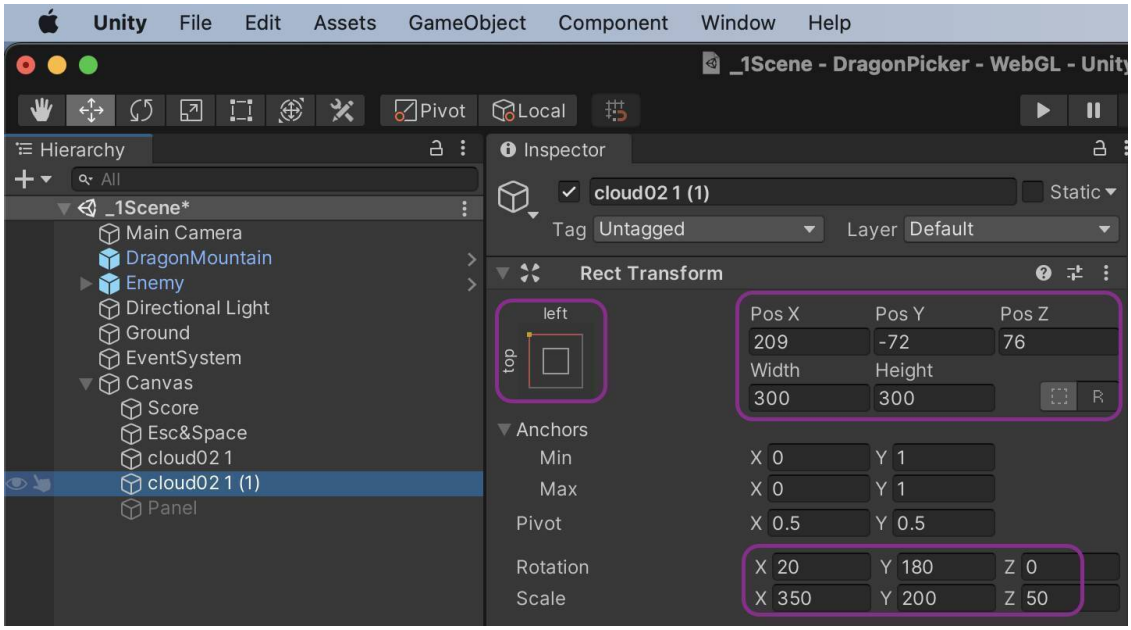
8. Запустите сцену и проверьте, что она работает корректно, а именно:

– При нажатии клавиши Space игра останавливается, при повторном нажатии – продолжается;

– При нажатии клавиши Esc происходит загрузка стартовой сцены с именем_0Scene.

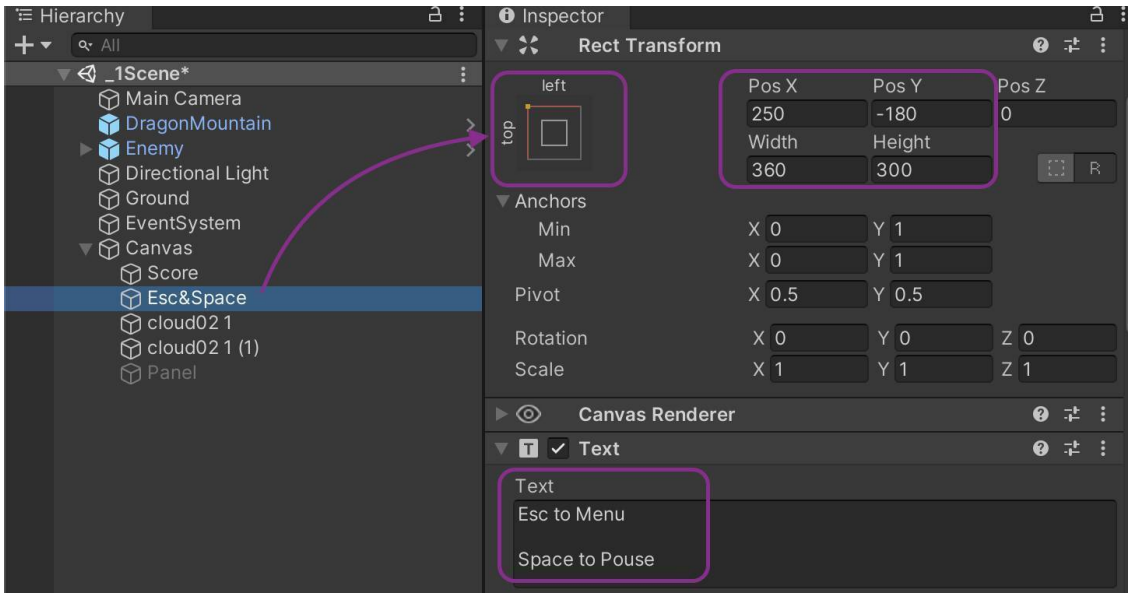
9. Теперь нам осталось лишь проинформировать пользователя о введенном функционале. Для этих целей мы можем разместить элементы графического интерфейса в левой верхней части окна с игрой. Для этого в окне Hierarchy создадим дубликат объекта cloud02 1 (ему

автоматически будет присвоено имя cloud02 1(1)), изменим привязку якоря к левому (left) верхнему (top) краю экрана и настроим параметры Rect Transform с положением и размером как показано ниже:

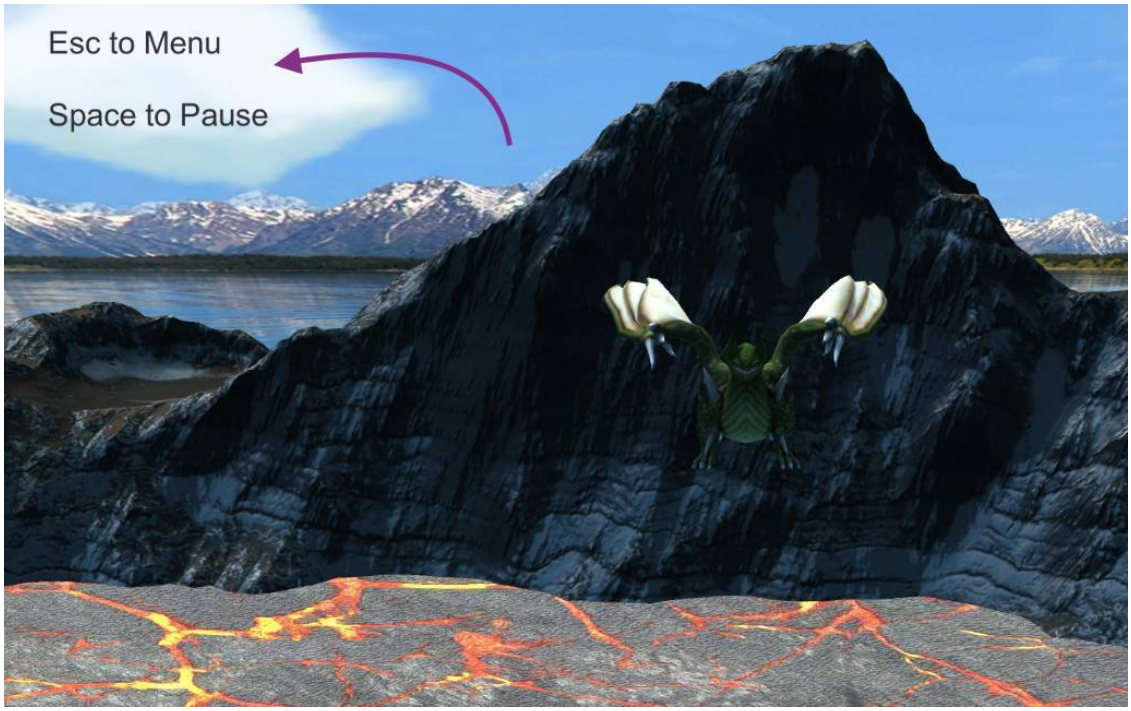


10. В окне Hierarchy создайте дубликат текстового объекта Score и переименуйте его в Ecs&Space.

11. Настройте размер и положение объекта Ecs&Space, как показано на рисунке ниже:



12. В результате выполненных преобразований, в верхней левой части окна с игрой должно появиться две надписи, информирующие пользователя о возможных.



13. Сохраните сцену _1Scene.

Выводы

В пятой части практикума мы создали элементы игрового меню и графического интерфейса, написали функционал, позволяющий взаимодействовать с элементами интерфейса. Научились загружать необходимые сцены в зависимости от нажатых пользователем клавиш, а также активировать и деактивировать отображение различных игровых объектов на экране, таких как надпись Pause и кнопки главного меню.

Как и ранее, ниже будут даны некоторые предложения по улучшению игры, которые вы можете сделать самостоятельно, опираясь на информацию, приведенную в этой части практикума.

– Доработайте внешний вид интерфейса, добавив “парящую” анимацию кнопок, либо анимацию изменения прозрачности/цвета кнопок. Сделать это можно по аналогии с созданием анимации движения облака на стартовой сцене. Только в этот раз анимация будет менять оформление кнопок, либо их координаты по вертикале.

– Подумайте над тем, чего не хватает в разработанном меню. Доработайте меню таким образом, чтобы пользователь имел возможность цикличного движение по игровому прототипу (без тупиковых веток, из которых он не сможет выйти).

– Подумайте над тем, какие настройки могут быть помещены внутри кнопки Option меню SettingsMenu. Подберите для придуманных настроек скины кнопок из Asset Store и добавьте их в SettingsMenu.

Часть 6. Добавление звуковых эффектов

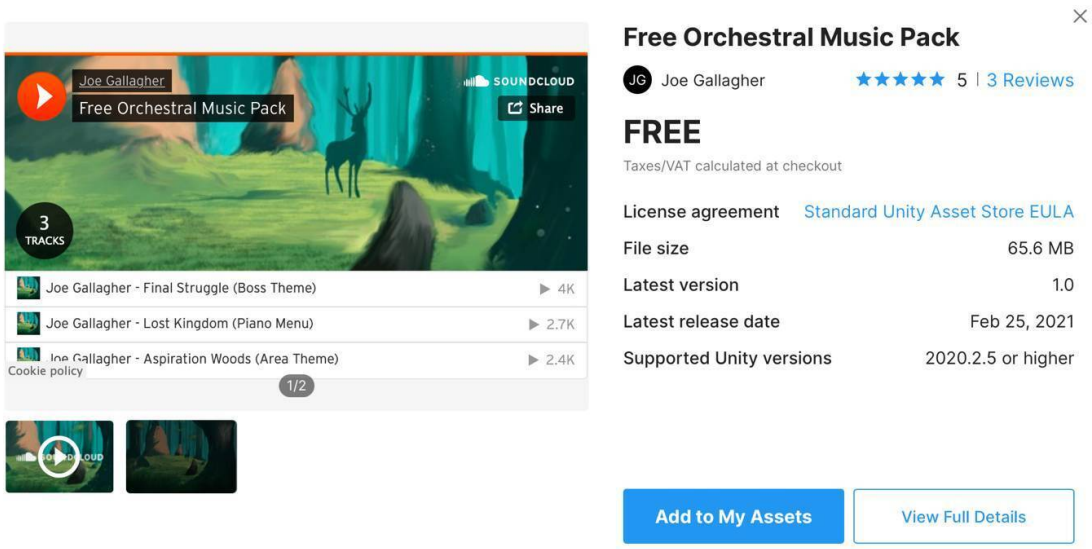
Введение

Без звуковых эффект игра не будет законченной. Кроме того, правильно подобранные звуки позволяют создавать нужное настроение, на звуках бывают завязаны довольно необычные механики, а дорожка на заднем фоне может как оттолкнуть игрока, так и наоборот – вовлечь в нее. В шестой части практикума мы добавим несколько простых звуковых эффектов для ключевых действий в нашей игре.

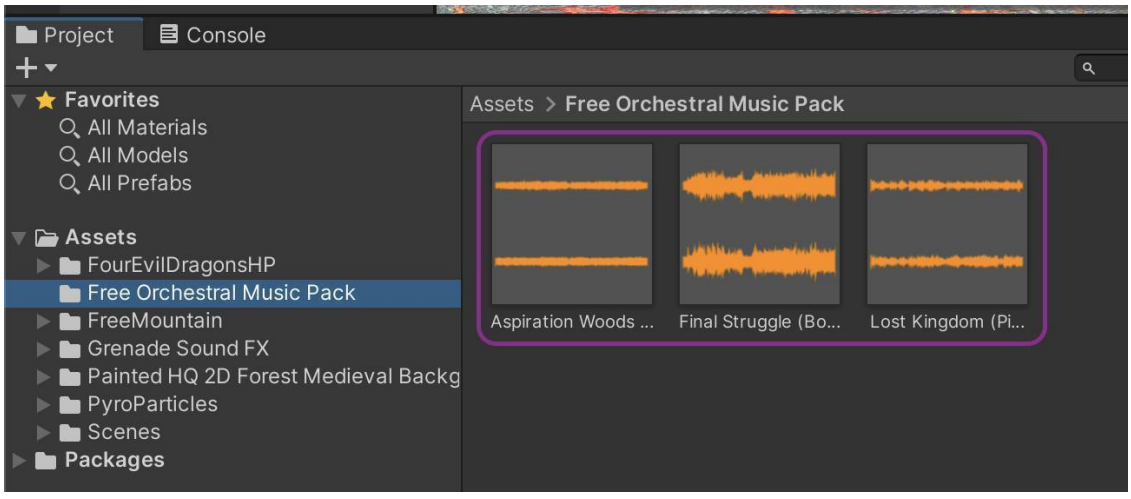
6.1 Добавление фоновых звуковых файлов

1. Добавим звуковые эффекты в проект Unity. Не будем изменять традициям, как и все другие ресурсы для нашей игры, звуковые эффекты мы найдем все там же – на Unity Asset Store.

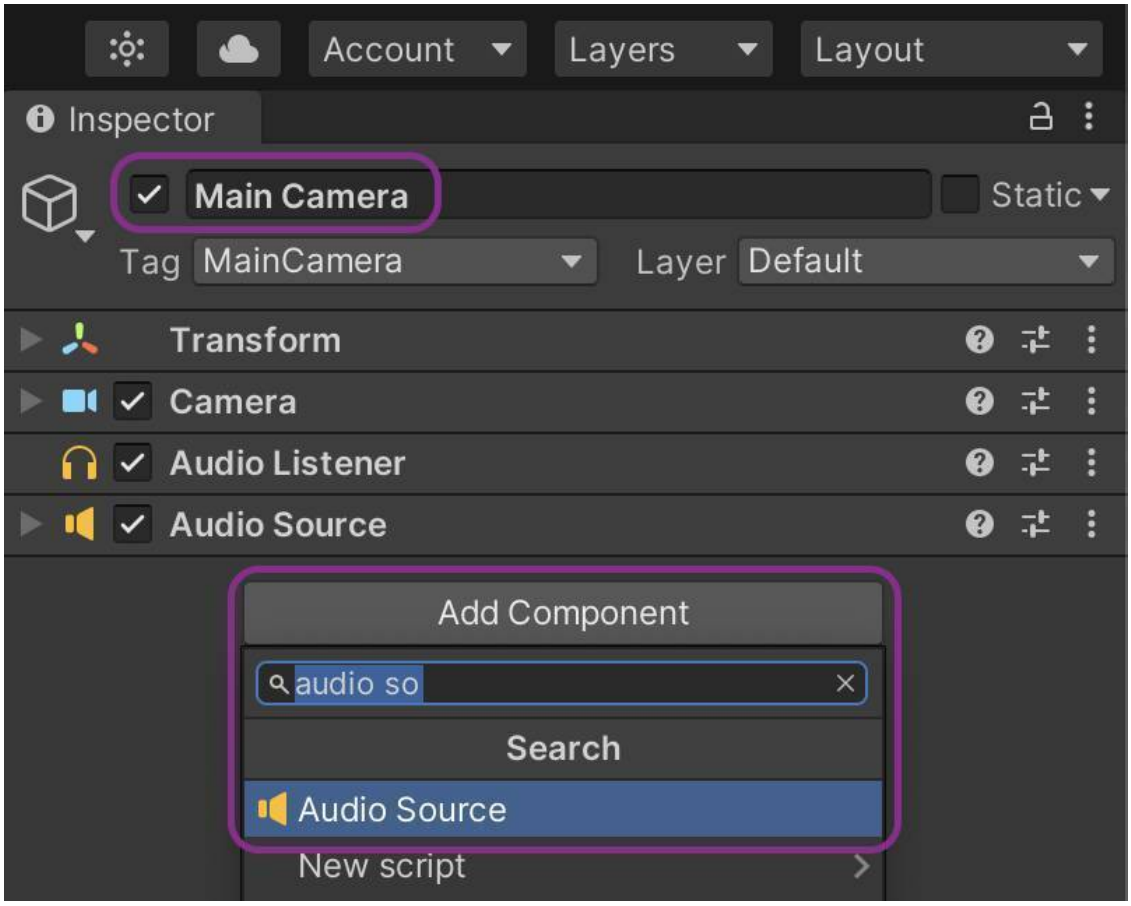
2. В качестве звуковой дорожки на фоне главного меню и сцены с игрой будем использовать аудиофайлы из Asset-пака с именем Free Orchestral Music Pack. Скачайте его и добавьте в проект:



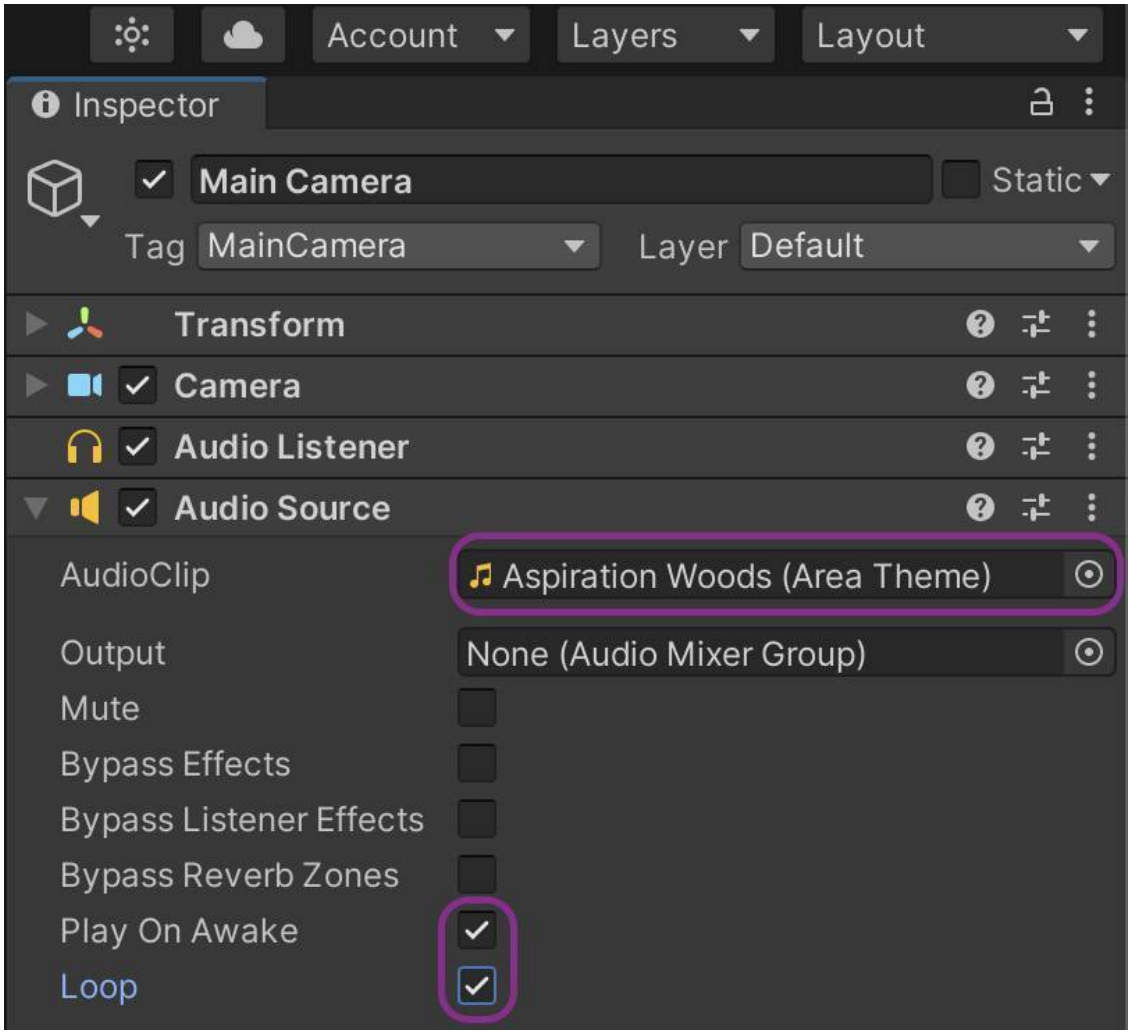
3. После импорта этого пака в среду разработки, в папке с одноименным названием будет доступно три аудиофайла, два из которых мы будем использовать в качестве фоновой музыки.



4. Фоновую музыку мы будет добавлять в качестве компонента Audio Source, прикрепленному к камере Main Camera. Убедитесь, что выбрана сцена _0Scene (стартовая сцена). Левой кнопкой мыши кликните по объекту Main Camera в окне Hierarchy. Далее справа в окне Inspector нажмите кнопку Add Component – Audio Source.

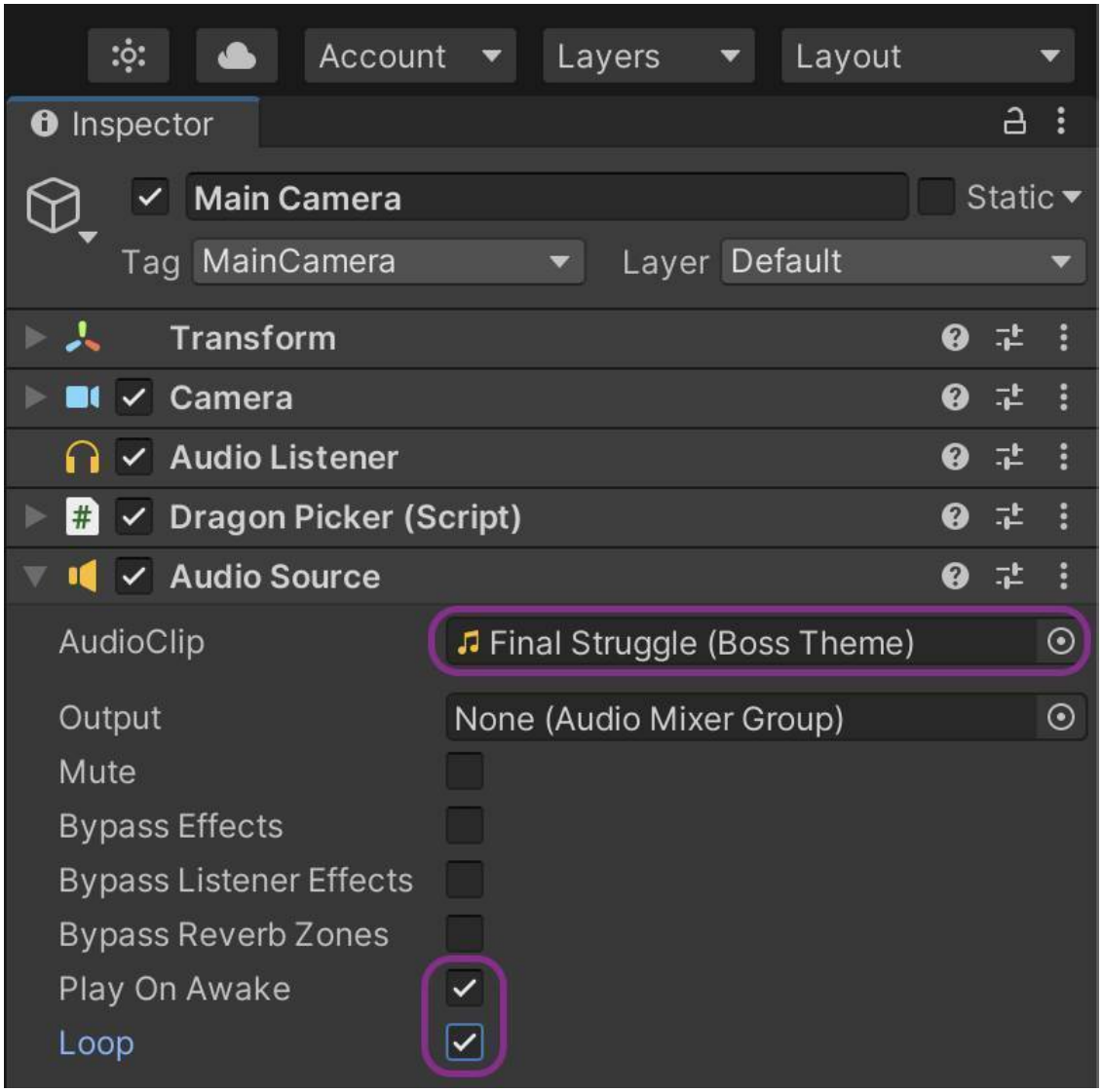


5. Внутри компонента Audio Source в поле AudioClip можно выбрать одну из скачанных композиций, например AspirationWoods, также поставьте галочки напротив чекбоксов с названием Play on Awake (нужна для того, чтобы запускать запись автоматически при старте сцены, а точнее – при появлении объекта Main Camera на сцене) и Loop (нужна для того, чтобы зациклить воспроизведение):



6. Сохраните сцену _0Scene. Запустите сцену, проверьте, что на фоне слышна подключенная звуковая дорожка.

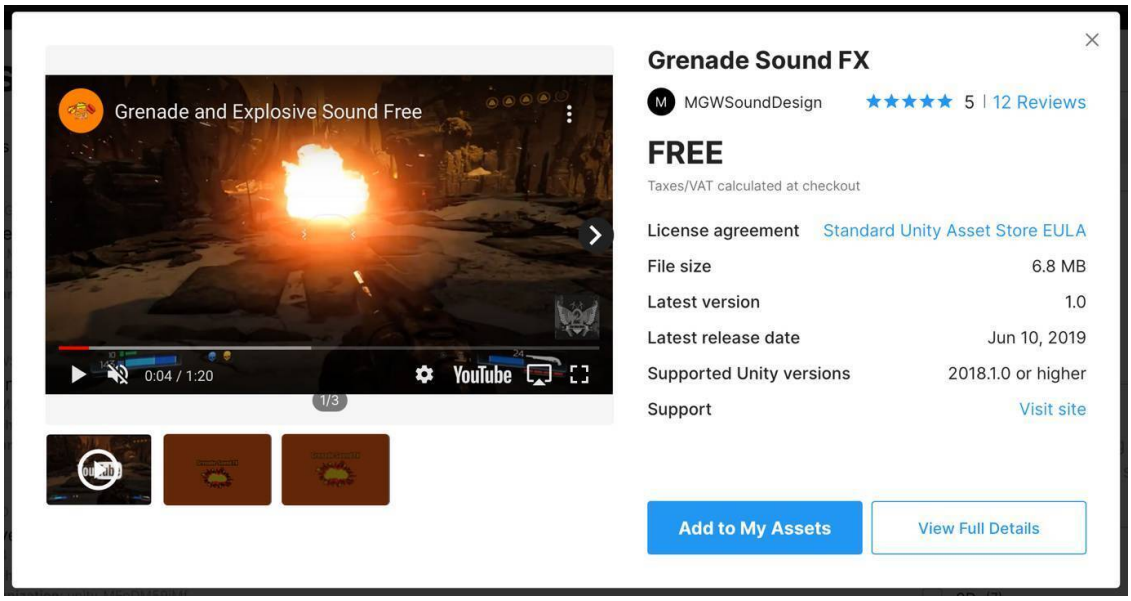
7. Откройте сцену _1Scene. Аналогично добавьте аудиозапись в качестве компонента Main Camera на сцене _1Scene. В качестве аудиозаписи выберите Final Struggle:



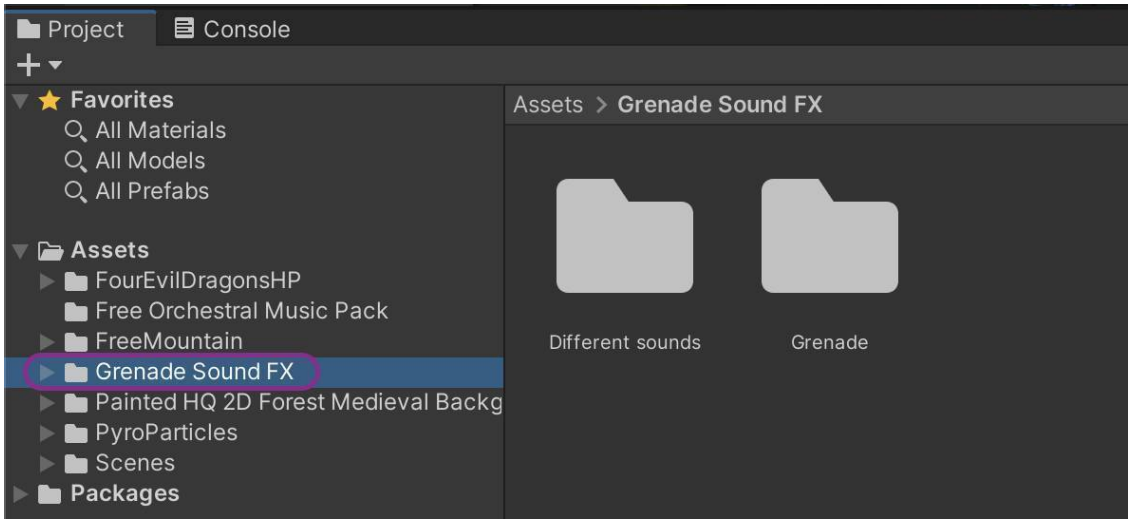
8. Сохраните сцену _1Scene. Запустите сцену, проверьте, что на фоне слышна подключенная звуковая дорожка.

6.2 Добавление звуковых эффектов при взаимодействии с объектами

Основное отличие звуковых эффектов от фоновой музыки отключается тем, что звуковой эффект должен запускаться при наступлении конкретных событий, например, при столкновении двух объектов. Как правило эффект имеет короткую длительность (пара секунд или менее). Для звуковых эффектов нам потребуется отдельный набор аудиофайлов. Один из подходящих нам называется Grenade Sound FX, доступный бесплатно на Unity Asset Store:



1. Добавьте ассет-пак в свою учетную запись (кнопка Add to My Assets), скачайте и импортируйте его через Package Manager в Unity.



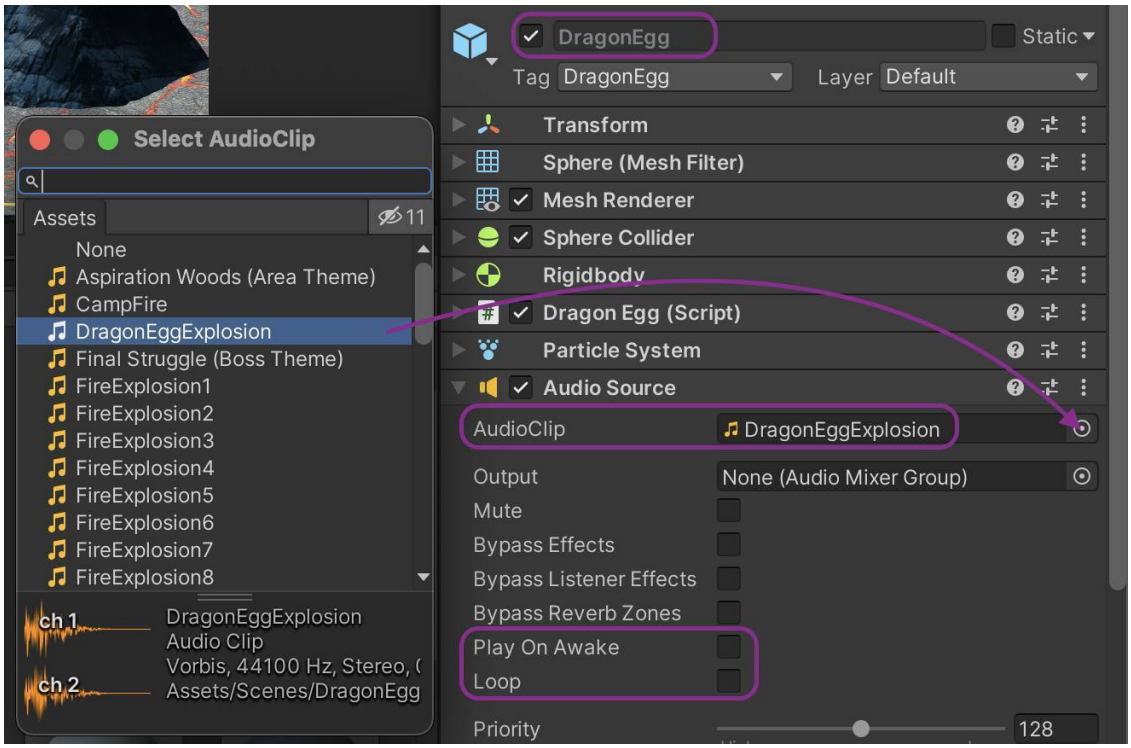
2. Добавленный Grenade Sound FX содержит различные звуки, связанные с бросанием и взрывом гранат. Мы будем использовать два аудиофайла, один для взрыва драконьего яйца при

падении на плоскость Ground, и второй – для ловли драконьего яйца энергетическим шаром EnergySheild.

3. Откройте папку Assets/Granade Sound FX/Granade. Внутри найдите файл с названием Granade2Short, создайте его копию (дубликат) и автоматически созданный дубликат с названием Granade2Short 1 перетащите в папку Assets/Scene, в которой лежат все остальные файлы проекта.

4. Перейдите в папку Assets/Scenes и переименуйте перемещенный ранее файл в DragonEggExplosion.

5. Теперь подключим аудиофайл к объекту DragonEgg. В той же папке выберите объект DragonEgg, в окне Inspector нажмите кнопку Add Component, добавьте компонент Audio Source и в поле Audio Clip из выпадающего списка выберите только что добавленный файл с аудио эффектом взрыва:



6. Теперь нам останется написать условие, при котором будет происходить запуск аудиофайла. Добавить воспроизведение мы можем в скрипт-файл DragonEgg.cs, к тому же там уже есть метод OnTriggerEnter, который срабатывает при пересечении драконьего яйца с триггером. В листинг ниже добавлено несколько строк кода, которые инициализируют аудио-компонент и запускают его при срабатывании триггера. Листинг из файла DragonEgg.cs приводится целиком:

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class DragonEgg : MonoBehaviour
{
    public static float bottomY = -30f;
    public AudioSource audioSource;
    private void OnTriggerEnter(Collider other)
    {
        ParticleSystem ps = GetComponent<ParticleSystem>();
        var em = ps.emission;
        em.enabled = true;
        Renderer rend;
        rend = GetComponent<Renderer>();
        rend.enabled = false;
        audioSource = GetComponent<AudioSource>();
        audioSource.Play();
    }
    void Update()
    {
        if (transform.position.y < bottomY)
        {
            Destroy(this.gameObject);
            DragonPicker apScript = Camera.main.GetComponent<DragonPicker>();
            apScript.DragonEggDestroyed();
        }
    }
}
// End Code
```

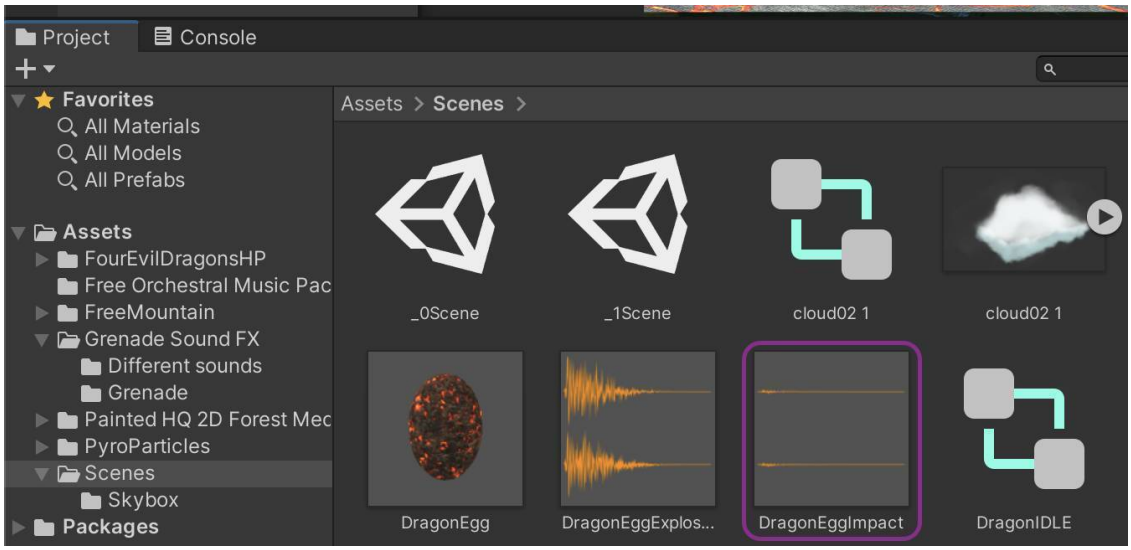
Как и ранее, листинг продублирован ниже в виде скриншота из среды разработки.

```
EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs
No selection
 1  using System.Collections;
 2  using System.Collections.Generic;
 3  using UnityEngine;
 4
 5  public class DragonEgg : MonoBehaviour
 6  {
 7      public static float bottomY = -30f;
 8      public AudioSource audioSource;                    1
 9
10      private void OnTriggerEnter(Collider other)
11      {
12          ParticleSystem ps = GetComponent<ParticleSystem>();
13          var em = ps.emission;
14          em.enabled = true;
15
16          Renderer rend;
17          rend = GetComponent<Renderer>();
18          rend.enabled = false;
19
20          audioSource = GetComponent<AudioSource>();     2
21          audioSource.Play();
22      }
23
24      void Update()
25      {
26          if (transform.position.y < bottomY)
27          {
28              Destroy(this.gameObject);
29              DragonPicker apScript = Camera.main.GetComponent<DragonPicker>();
30              apScript.DragonEggDestroyed();
31          }
32      }
33  }
```

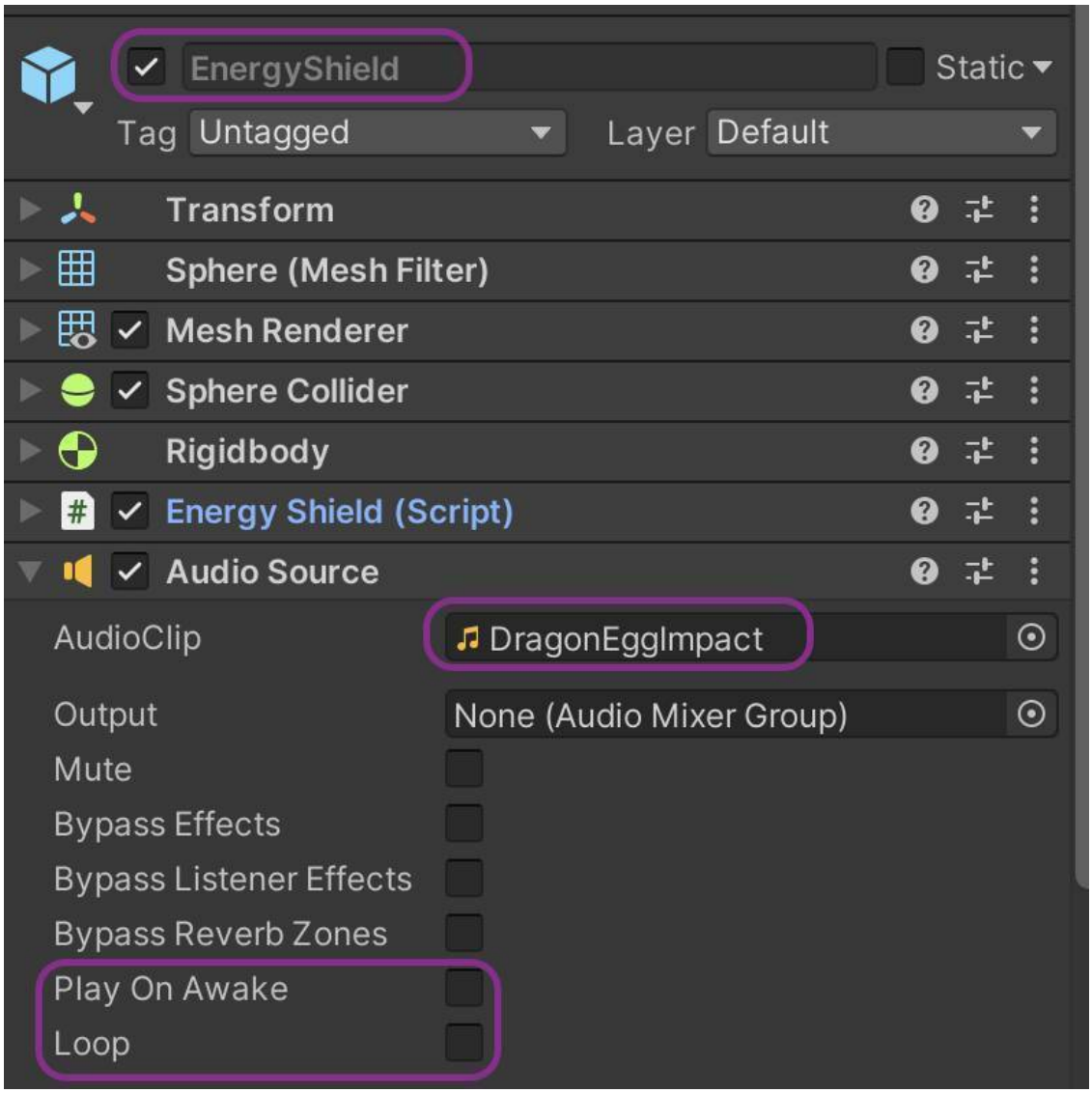
7. Сохраните скрипт-файл, запустите игру и проверьте, что при касании DragonEgg земли Ground, происходит проигрывание звукового эффекта.

8. Аналогично добавим аудиофайл, который будет срабатывать при ловле DragonEgg. Откройте папку Assets/Granade Sound FX/Different sounds. Внутри найдите файл с названием Impact on Snow, создайте его копию (дубликат) и созданный дубликат с названием Impact on Snow 1 перетащите в папку Assets/Scene, в которой лежат все остальные файлы проекта.

9. Переименуйте аудиофайл в DragonEggImpact:



10. Теперь подключим аудиофайл к объекту EnergyShield. В той же папке выберите объект EnergyShield, в окне Inspector нажмите кнопку Add Component, добавьте компонент Audio Source и в поле Audio Clip из выпадающего списка выберите только что добавленный файл с аудио эффектом DragonEggImpact:



11. Также как и ранее, нам осталось написать условие запуска прикрепленного аудио-компонента. В скрипт-файле EmergyShield.cs у нас есть метод private void OnCollisionEnter(Collision coll), в котором описано взаимодействия драконьего яйца и энергетического шара. Поэтому будет логично добавить функционал запуска аудиофайла именно в этом метод. Две новые части кода выделены, как и ранее комментариями. Листинг скрипт-файла EnergyShield.cs приводится целиком:

```
// Start Code
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

```
    public class EnergyShield : MonoBehaviour
    {
        public Text scoreGT;
        public AudioSource audioSource;
        void Start()
        {
            GameObject scoreGO = GameObject.Find("Score");
            scoreGT = scoreGO.GetComponent<Text>();
            scoreGT.text = "0";
        }
        void Update()
        {
            Vector3 mousePos2D = Input.mousePosition;
            mousePos2D.z = -Camera.main.transform.position.z;
                                        Vector3    mousePos3D    =
Camera.main.ScreenToWorldPoint(mousePos2D);
            Vector3 pos = this.transform.position;
            pos.x = mousePos3D.x;
            this.transform.position = pos;
        }
        private void OnCollisionEnter(Collision coll)
        {
            GameObject Collided = coll.gameObject;
            if (Collided.tag == "DragonEgg")
            {
                Destroy(Collided);
            }
            int score = int.Parse(scoreGT.text);
            score += 1;
            scoreGT.text = score.ToString();
            audioSource = GetComponent<AudioSource>();
            audioSource.Play();
        }
    }
```

// End Code

Скриншот листинга с новыми выделенными частями кода 1 и 2 приведен ниже:

EnemyDragon.cs | DragonEgg.cs | DragonPicker.cs | EnergyShield.cs

No selection

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5
6   public class EnergyShield : MonoBehaviour
7   {
8       public Text scoreGT;
9       public AudioSource audioSource;
10      void Start()
11      {
12          GameObject scoreGO = GameObject.Find("Score");
13          scoreGT = scoreGO.GetComponent<Text>();
14          scoreGT.text = "0";
15      }
16      void Update()
17      {
18          Vector3 mousePos2D = Input.mousePosition;
19          mousePos2D.z = -Camera.main.transform.position.z;
20          Vector3 mousePos3D = Camera.main.ScreenToWorldPoint(mousePos2D);
21          Vector3 pos = this.transform.position;
22          pos.x = mousePos3D.x;
23          this.transform.position = pos;
24      }
25      private void OnCollisionEnter(Collision coll)
26      {
27          GameObject Collided = coll.gameObject;
28          if (Collided.tag == "DragonEgg")
29          {
30              Destroy(Collided);
31          }
32          int score = int.Parse(scoreGT.text);
33          score += 1;
34          scoreGT.text = score.ToString();
35          audioSource = GetComponent<AudioSource>();
36          audioSource.Play();
37      }
38  }
```

1

2

12. Сохраните сцену _1Scene, проверьте корректную работу аудиофайлов. При ловле объекта DragonEgg должен срабатывать характерный звук. При падении яйца на землю должен проигрываться звук взрыва.

Выводы

Таким образом, в 6-й части нашего практикума мы добавили четыре звуковых файла, да из которых – фоновая музыка на сценах _0Scene и _1Scene, и два других – срабатывают при ловле объекта DragonEgg, либо при его падении на плоскость Ground. При желании вы можете самостоятельно доработать игру, например:

– добавить звуки взмахов крыльев при движении дракона;

– характерный звук завершения игры при потере всех объектов EnergyShield (жизней);

– звуки нажатия кнопок из главного меню и звуки переходов между сценами. Например, при запуске сцены _1Scene может проигрываться звук драконьего рева;

– звуки потери одной жизни (уничтожении одного EnergyShield из массива).

Часть 7. Подготовка приложения к публикации

Введение

В предыдущих разделах нашего практикума мы разработали игру и теперь настало время показать ее миру. В качестве площадки для размещения игры мы будем использовать simmer.io – это веб-платформа, на которую можно загрузить разработанную игру, сохраненную в формате Unity WebGL. Игра добавляет в веб-браузер простым перетаскиванием. Площадка simmer.io – это своего рода YouTube, в мире игр.

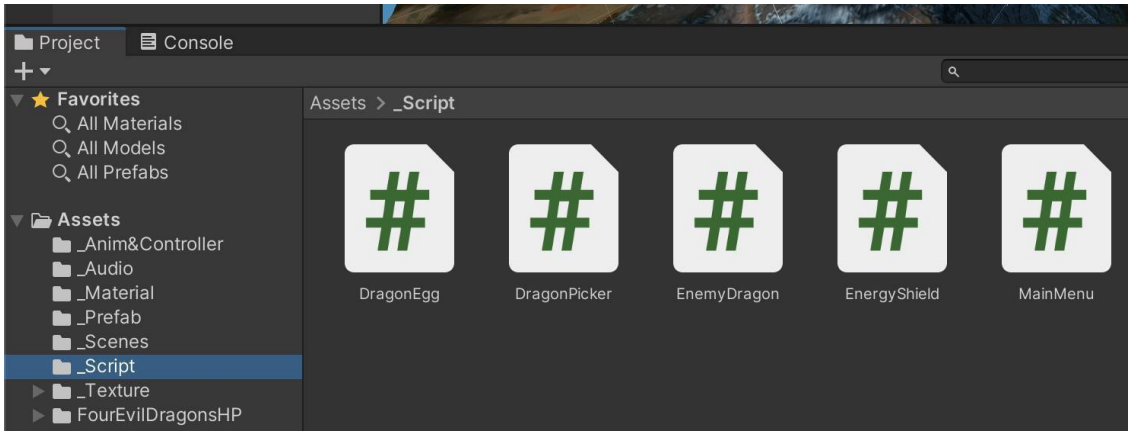
Но перед тем, как перейти к выгрузке игры, нам следует прибраться в файлах проекта. До этого мы сохраняли все файлы, с которыми работали в процессе создания игры в одну папку, что неправильно с точки зрения организации структуры проекта. Однако, мы это делали для того, чтобы можно было наблюдать изменение файлов в динамике при их создании, а также это упрощало описание перехода по папкам при необходимости состыковки различных исходных файлов между собой. В седьмой части практикума мы займемся структурированием файлов проекта по папкам (параграф 7.1) и займемся выгрузкой игры на веб-платформу simmer.io (параграф 7.2).

7.1 Структурирование файлов проекта

1. Откройте папку Assets внутри проекта DragonPicker. Создайте внутри Assets папки с именами:

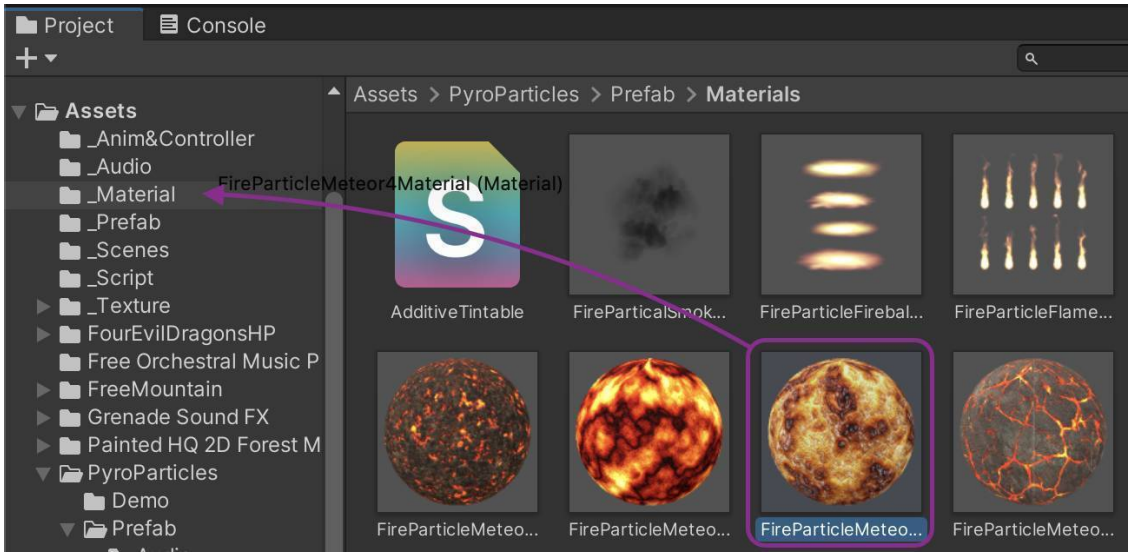
– Script;
– Texture;
– Prefab;
– Material;
– Audio;
– Anim&Controller;
– Scenes (существовала с самого начала).

2. Переместите все созданными нами в ходе практикума скрипты, материалы и другие объекты в соответствующие папки. Перенос файлов не повлияет на функционал игры, это нужно лишь для того, чтобы в будущем (например, если вы захотите расширить функционал игры) вам было более удобно работать с разрастающимся количеством файлов в игре. Так, например, после распределения файлов по папкам, папка Script должна выглядеть так, как показано на рисунке ниже:

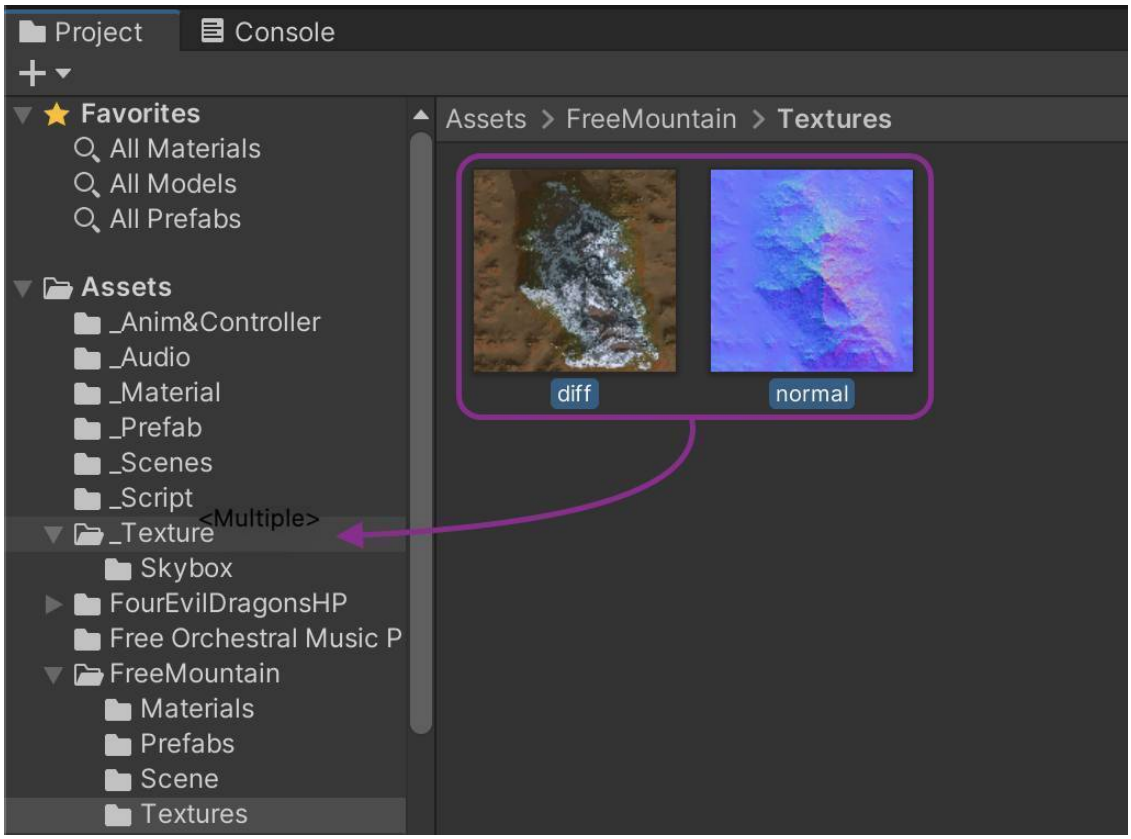


3. Также обратим внимание на то, что не всегда в ходе практикума мы создавали копии исходных файлов, а работали напрямую с файлами из папки, скачанной с Asset Store. При желании вы можете также самостоятельно найти такие файлы (например, это текстура драконьего яйца), создать их дубликаты и поместить в созданную папку Texture.

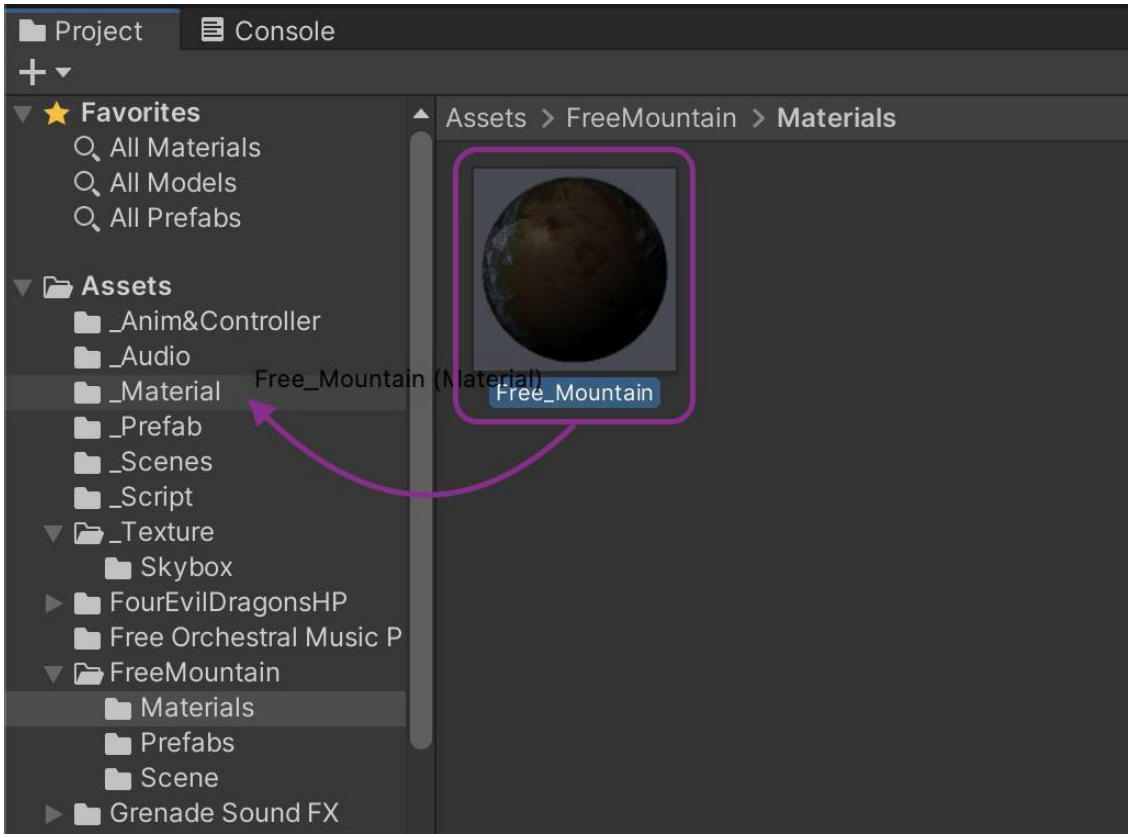
4. Переносим материал FireParticleMeteorMaterial в папку _Material, как показано на рисунке ниже:



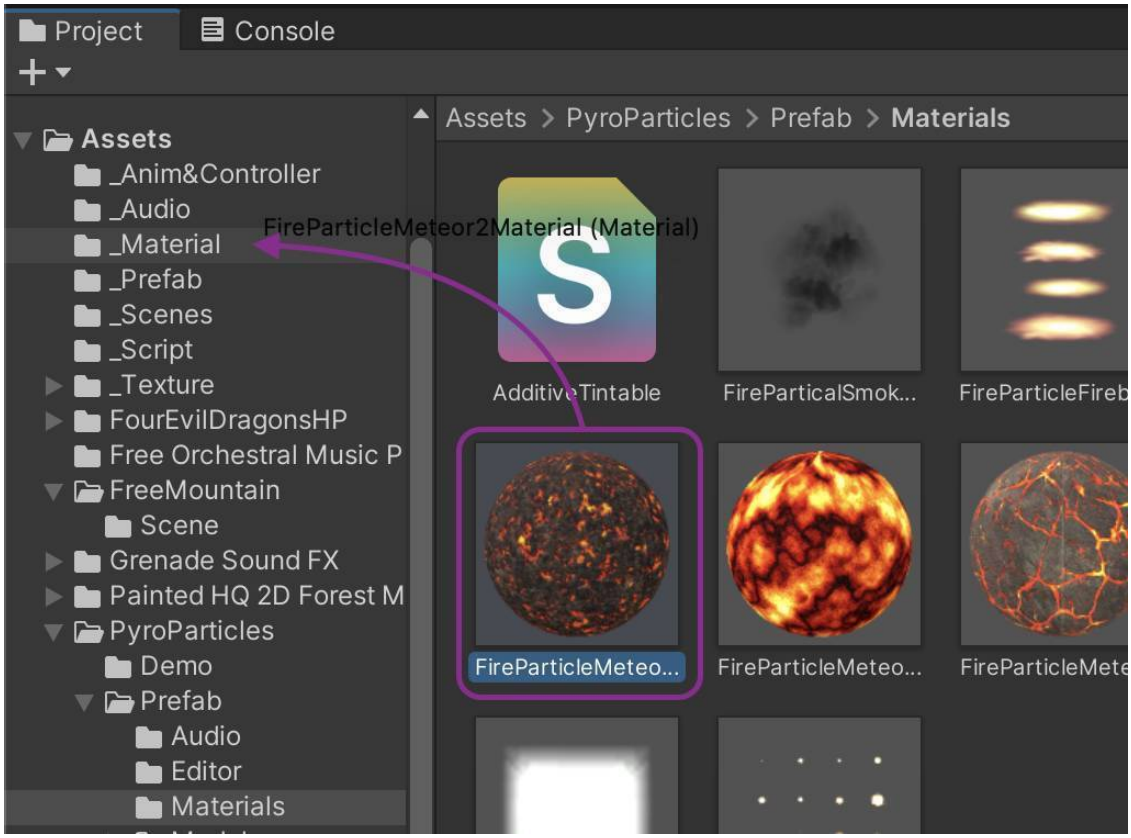
5. Перенос текстур гор из папки Assets/FreeMountain/Textures в папку Assets/_Texture:



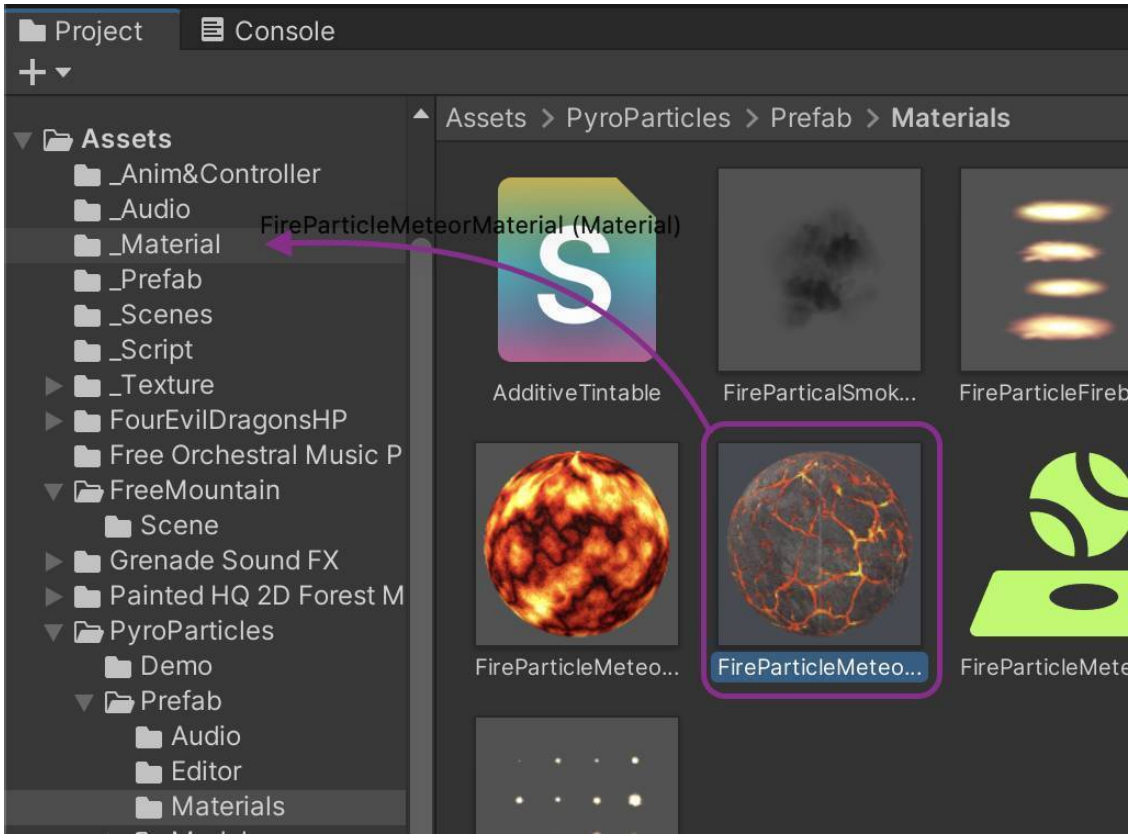
6. Перенос материала горы в папку:



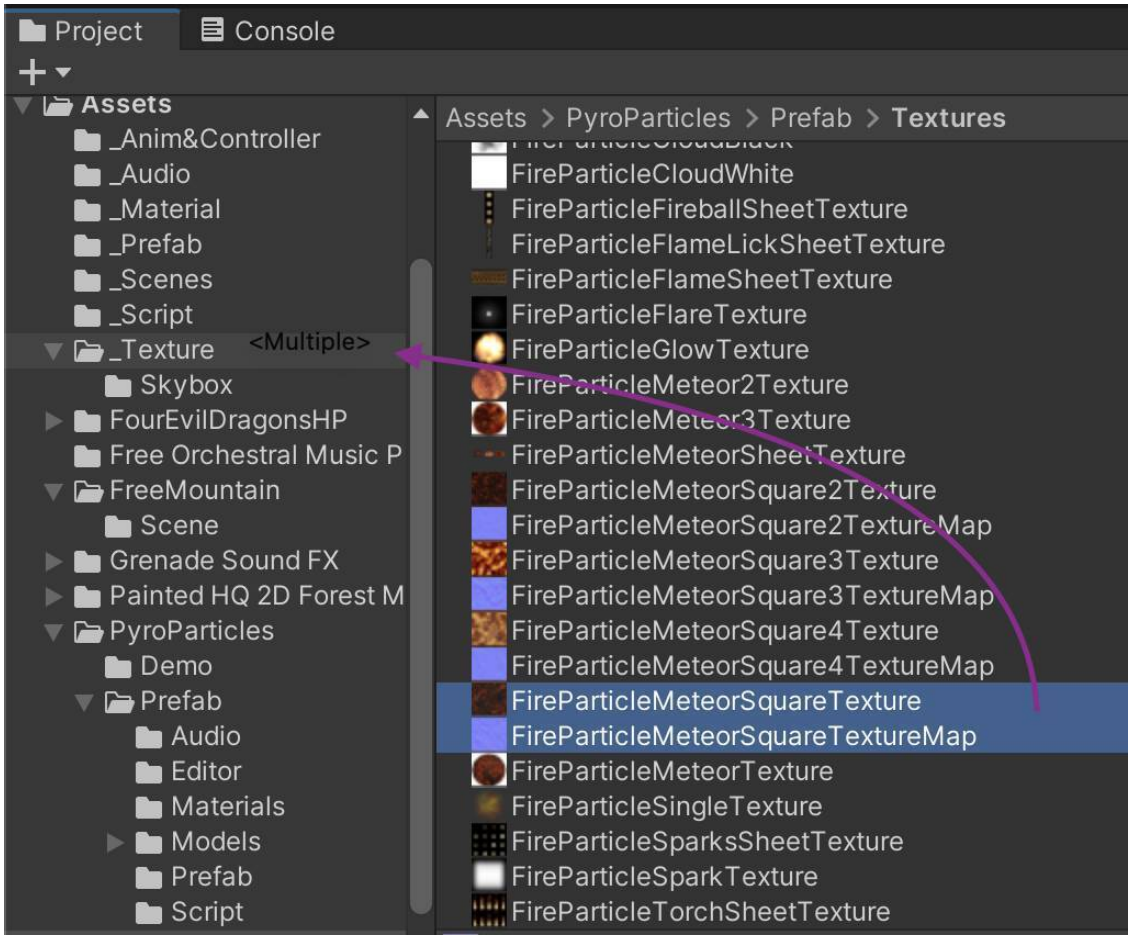
7. Перенос материала драконьего яйца:



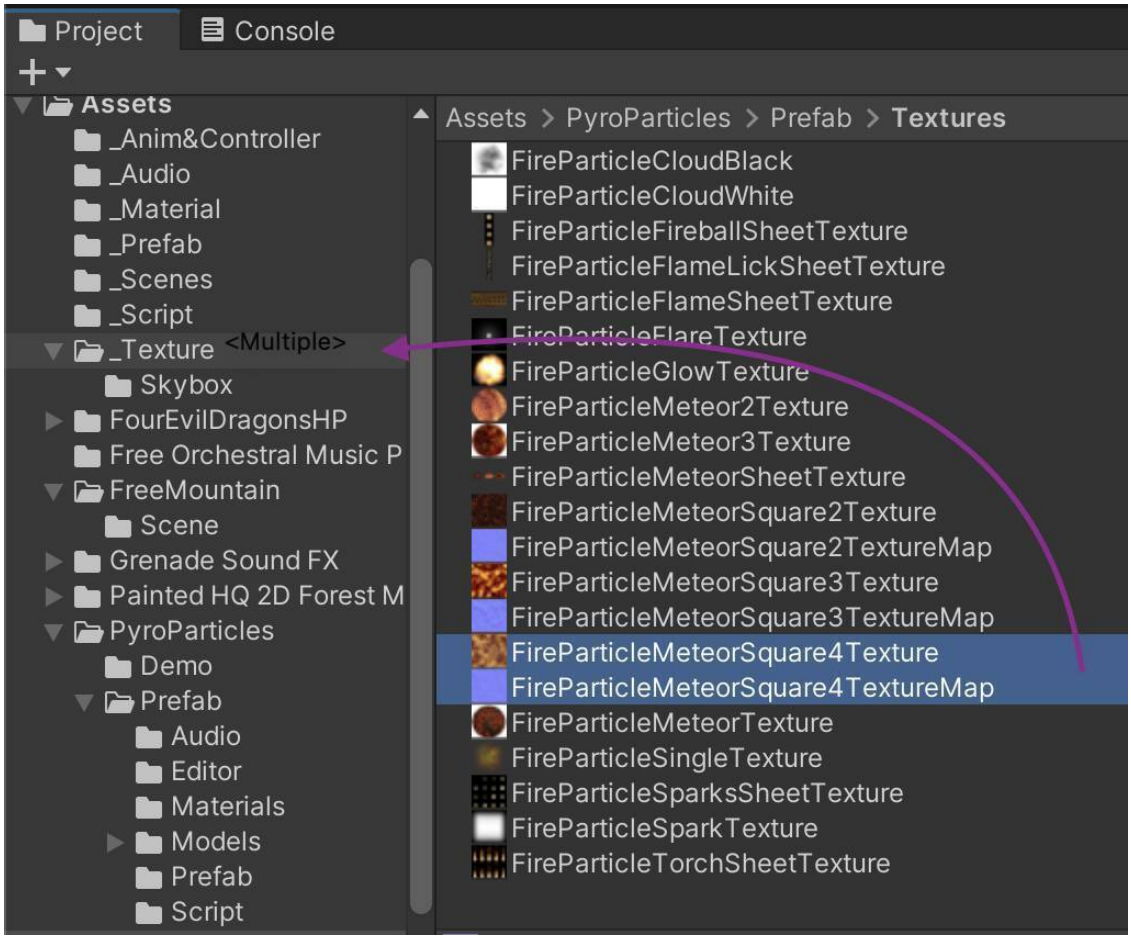
8. Перенос материала плоскости земли:



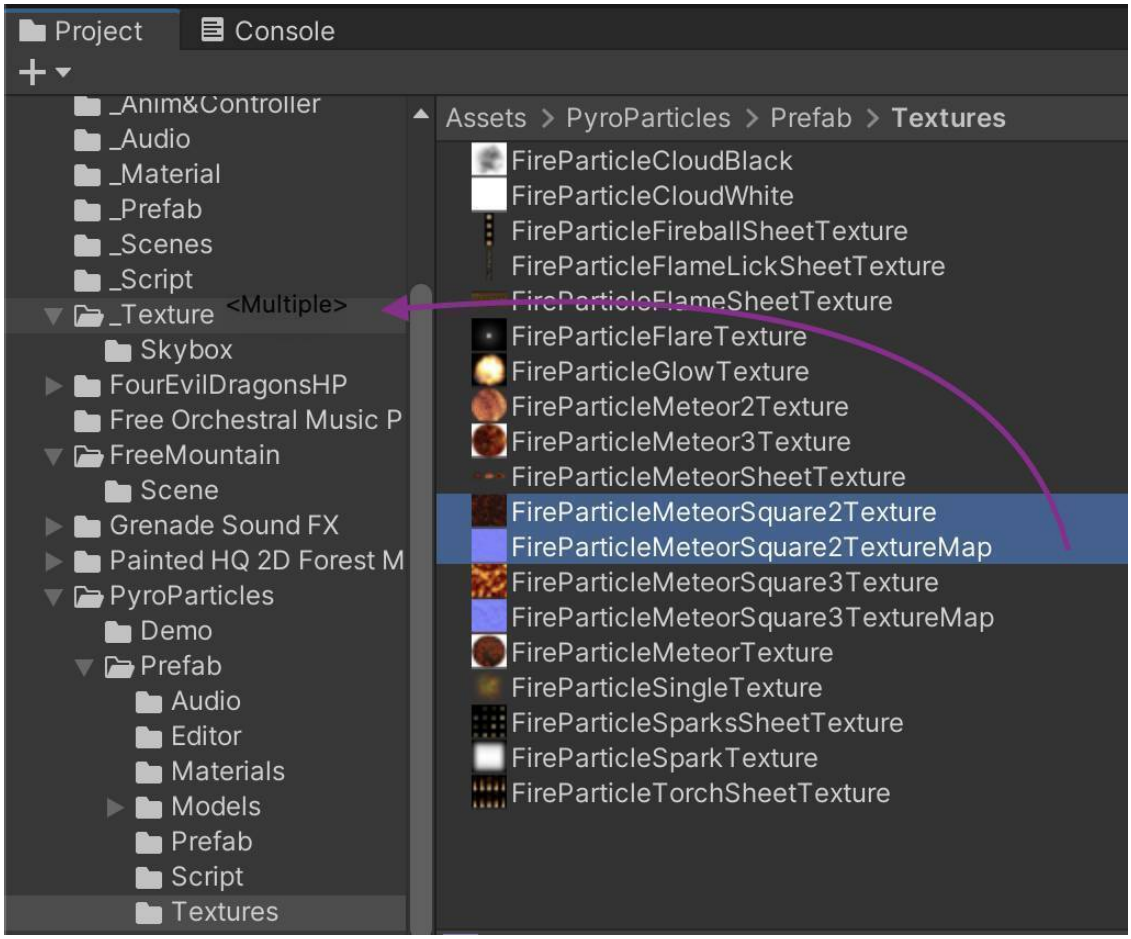
9. Текстуры метеорита:



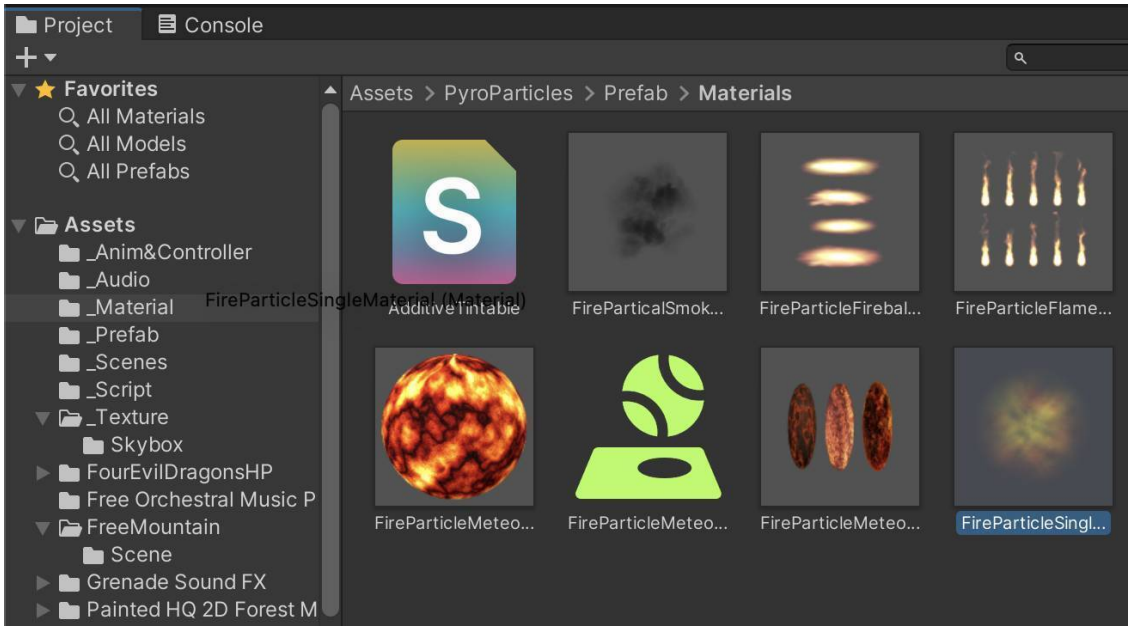
10. Текстура FireParticle, переносим в папку _Texture:



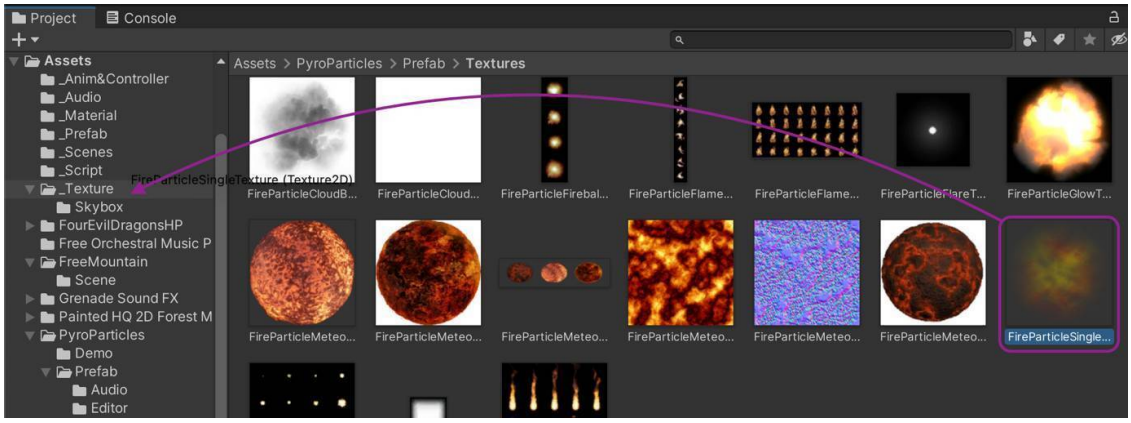
11. Перенос материала текстуры метеорита в папку _Texture:



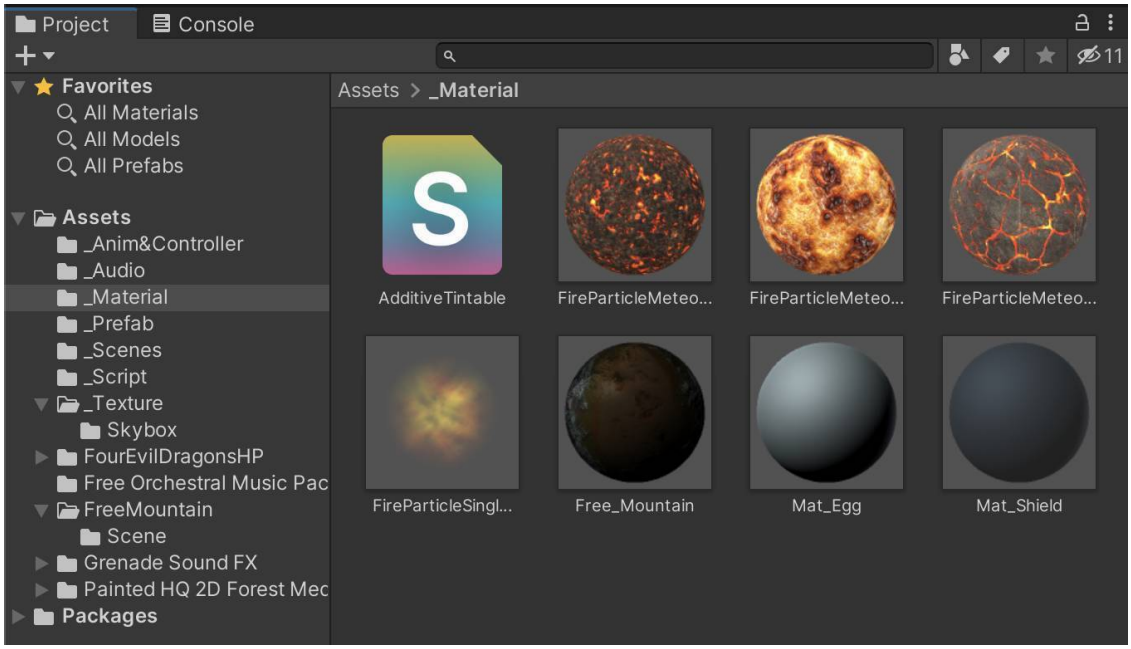
12. Материал взрыва драконьего яйца в папку _Material:



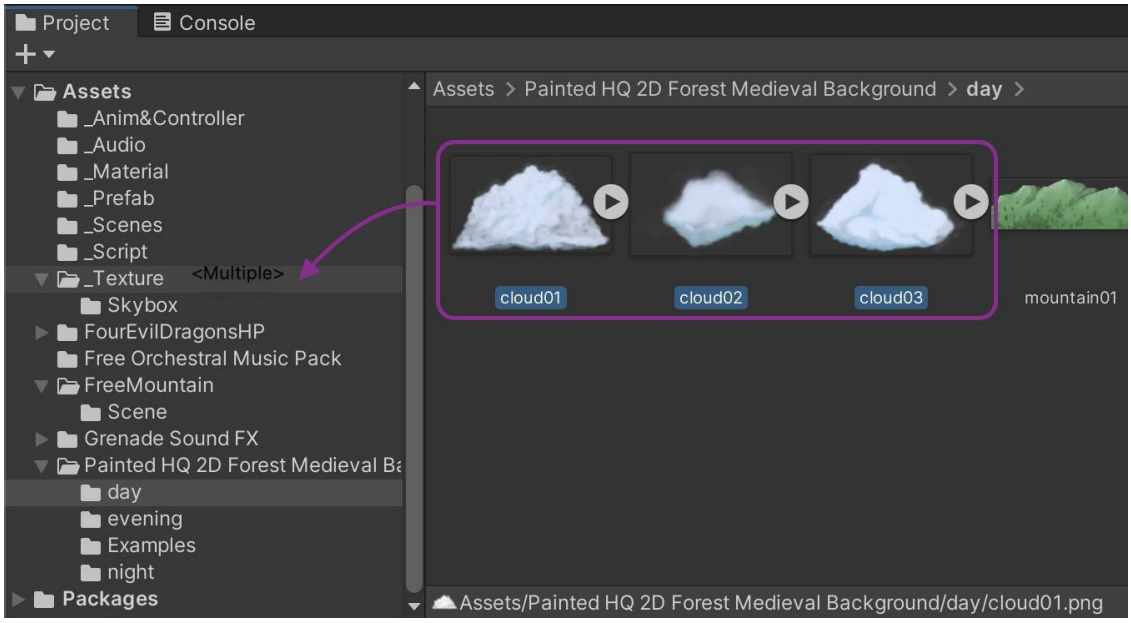
13. Перенос материала Fire Particle single:



14. Также следует перенести в папку _Material из папки Materials шейдер с именем AdditiveTintable.shader. Таким образом, содержимое папки _Material будет иметь следующий вид:



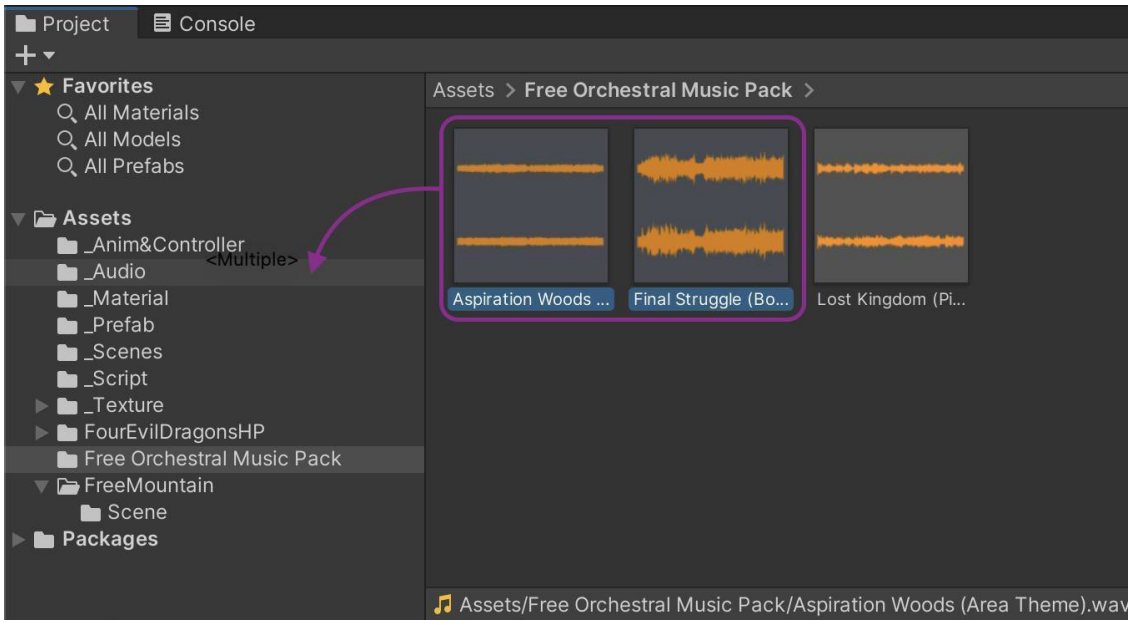
15. После этого вы можете целиком удалить папку PyroParticles с ассет-паком.
16. Аналогично перенесите текстуры с облаками из следующего ассет-пака:



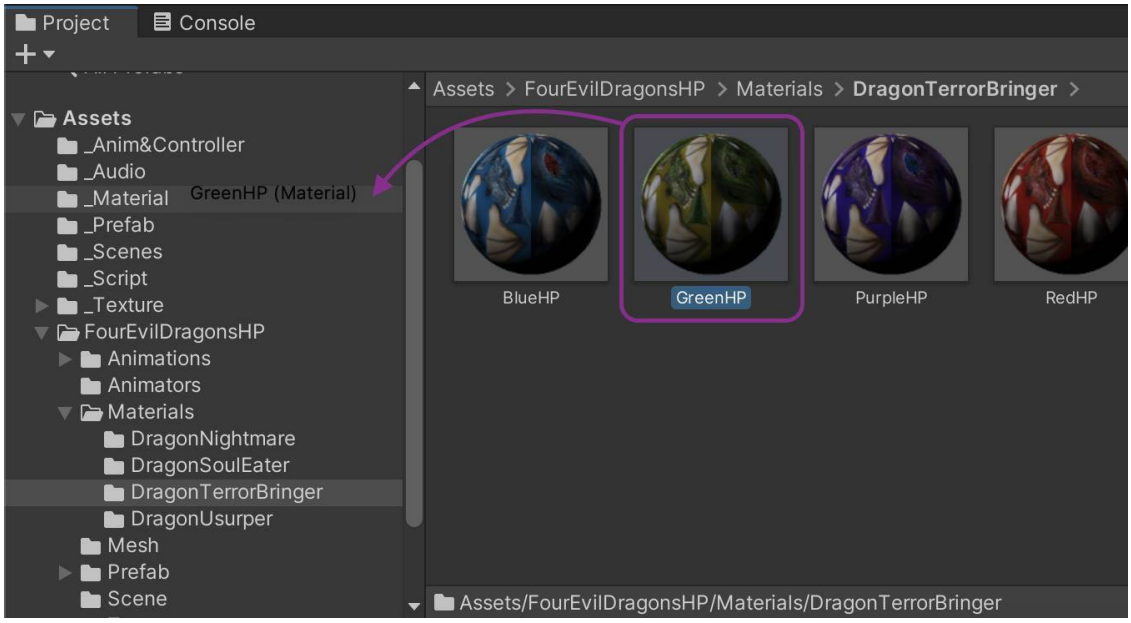
17. После этого можете удалить папку Painted HQ 2D Forest Medieval с другими текстурами облаков.

18. Также можно удалить папку Grenade Sound FX. Аудиофайлы из этого ассет-пака мы копировали в процессе.

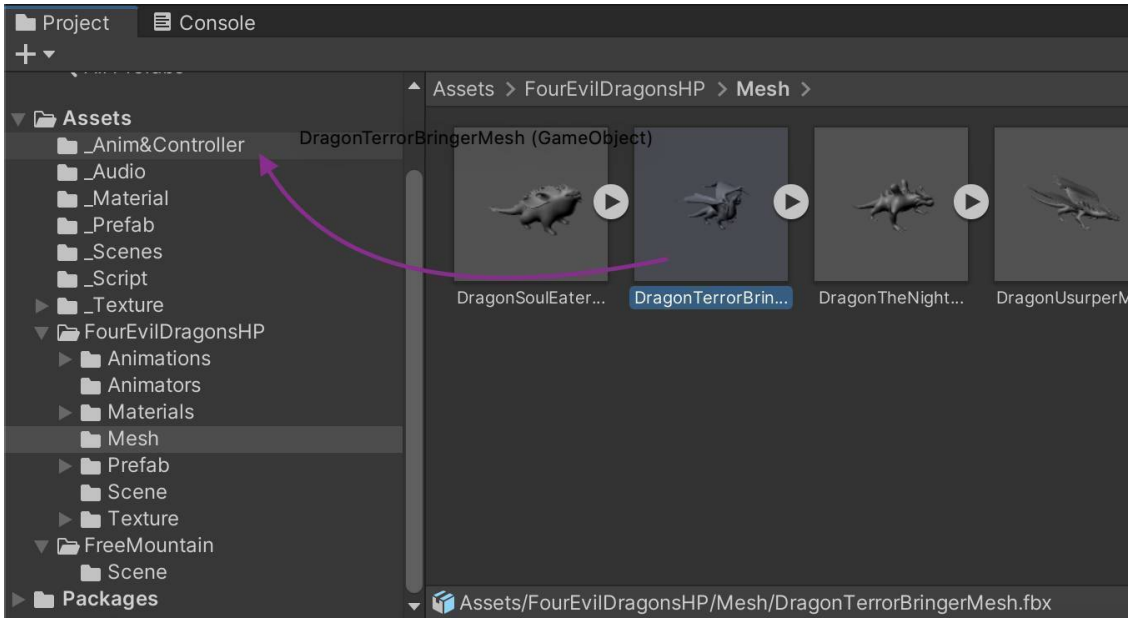
19. Два аудиофайла из папки Free Orchestral Music Pack также следует перенести в папку Audio. После этого можно удалить папку Free Orchestral Music Pack с оставшимся одним незадействованным файлом.



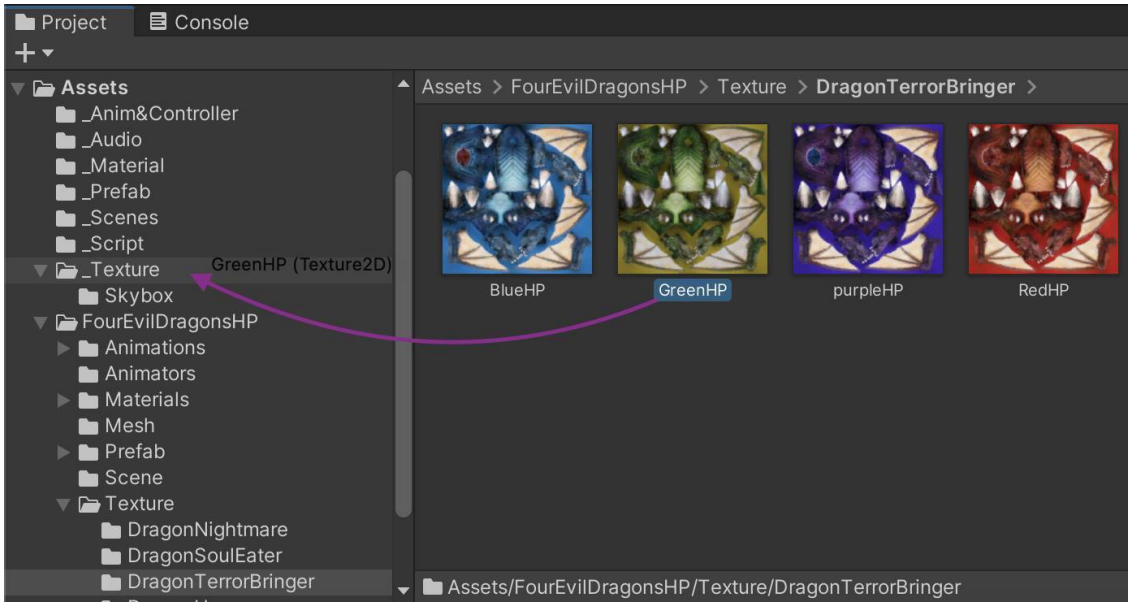
20. Переносим материал дракона GreenHP в папку проекта с именем _Material:



21. Перенос сетки (Mesh) дракона из папки Mesh в папку _Anim&Controller:

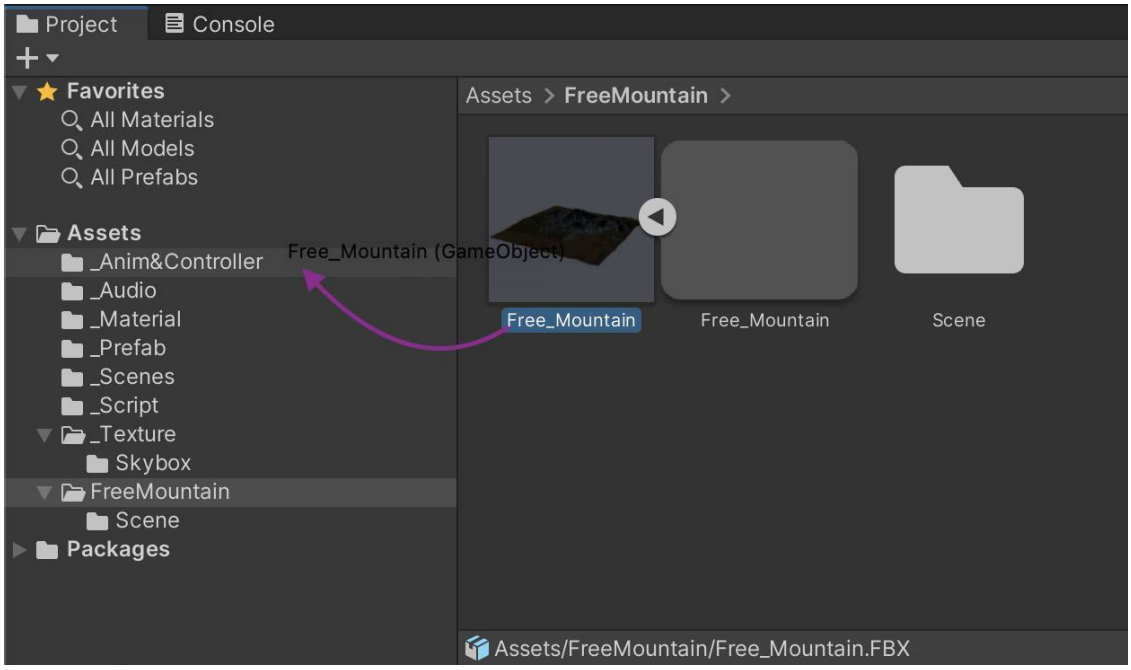


22. Переносит текстуру дракона в папку _Texture:



23. После этого можно удалить скачанный ассет-пак с драконами (папку с именем FourEvilDragonsHP).

24. Останется переместить Free_Mountain.FBX.

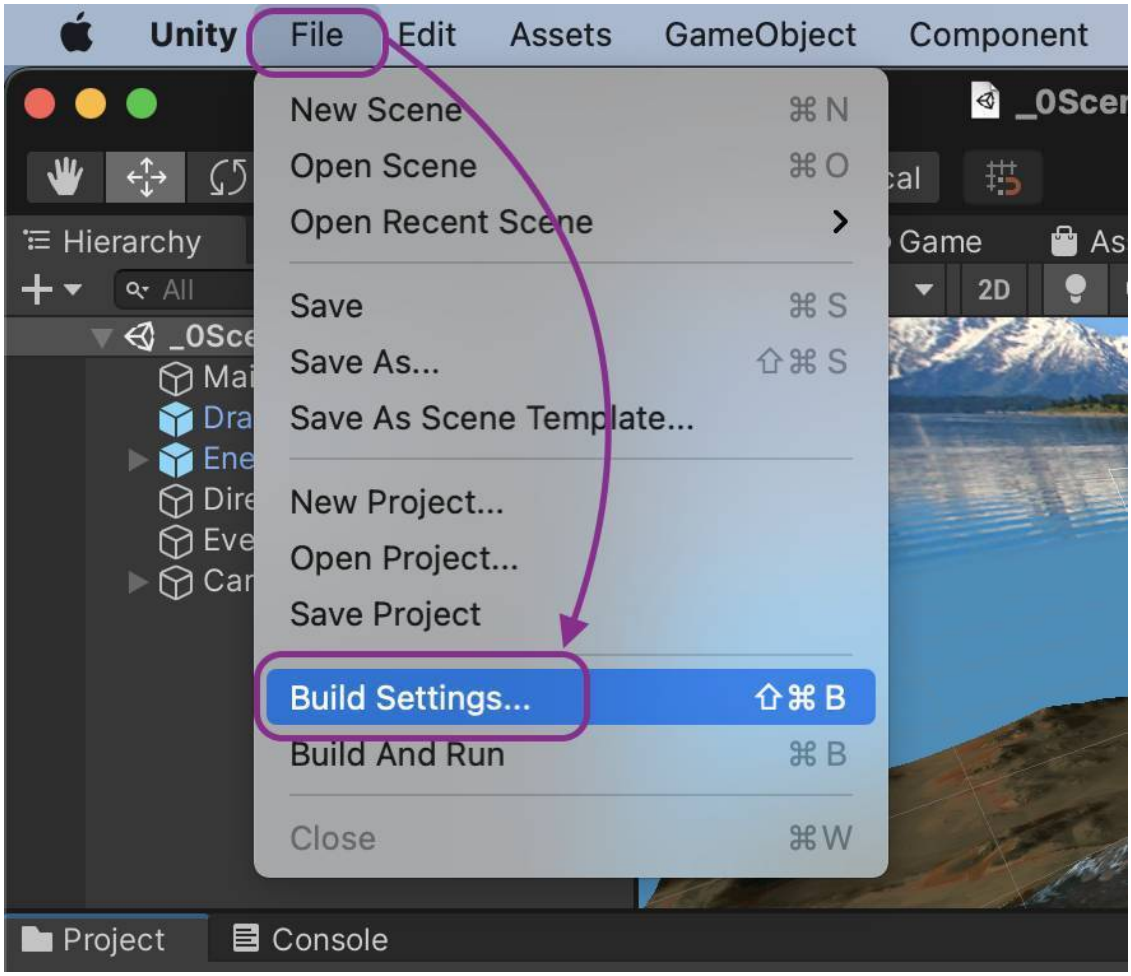


25. Удалим папку FreeMountain.

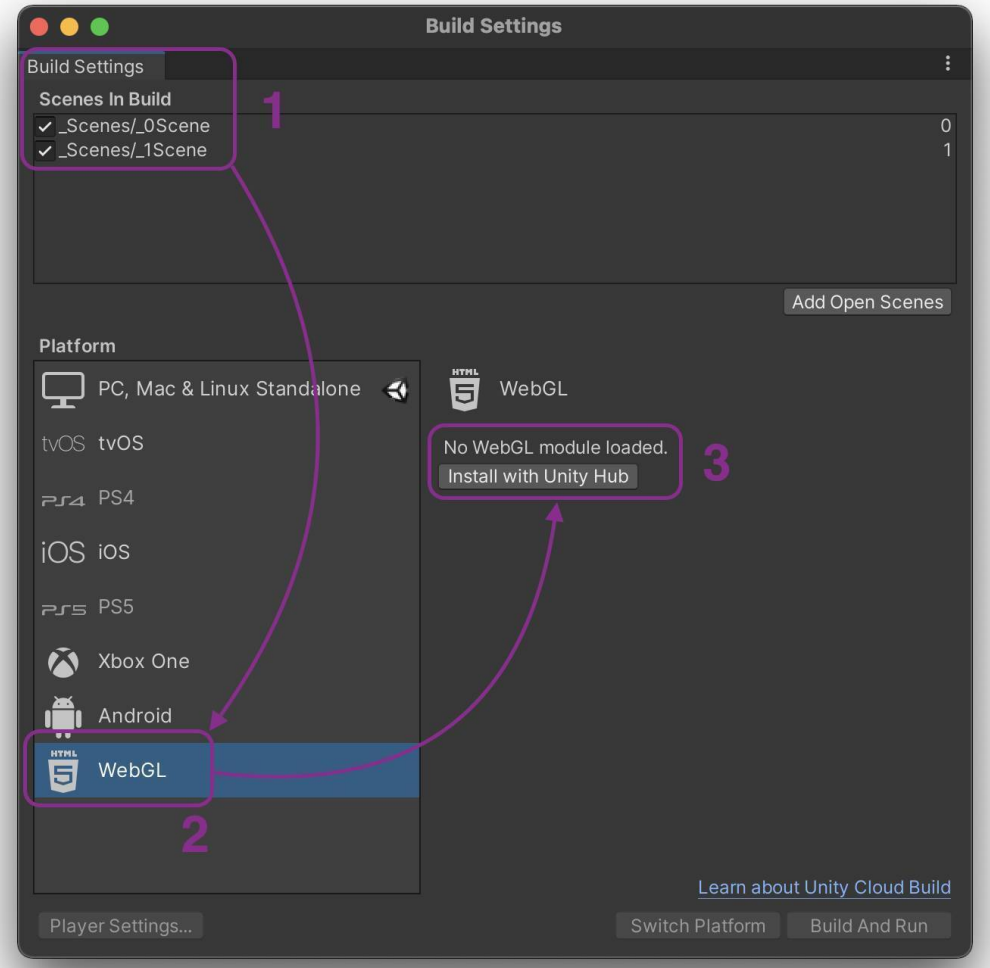
26. После этого в проекте должны остаться только те файлы, которые будут участвовать непосредственно в сборке проекта.

7.2 Сборка проекта и выгрузка игры на сайт

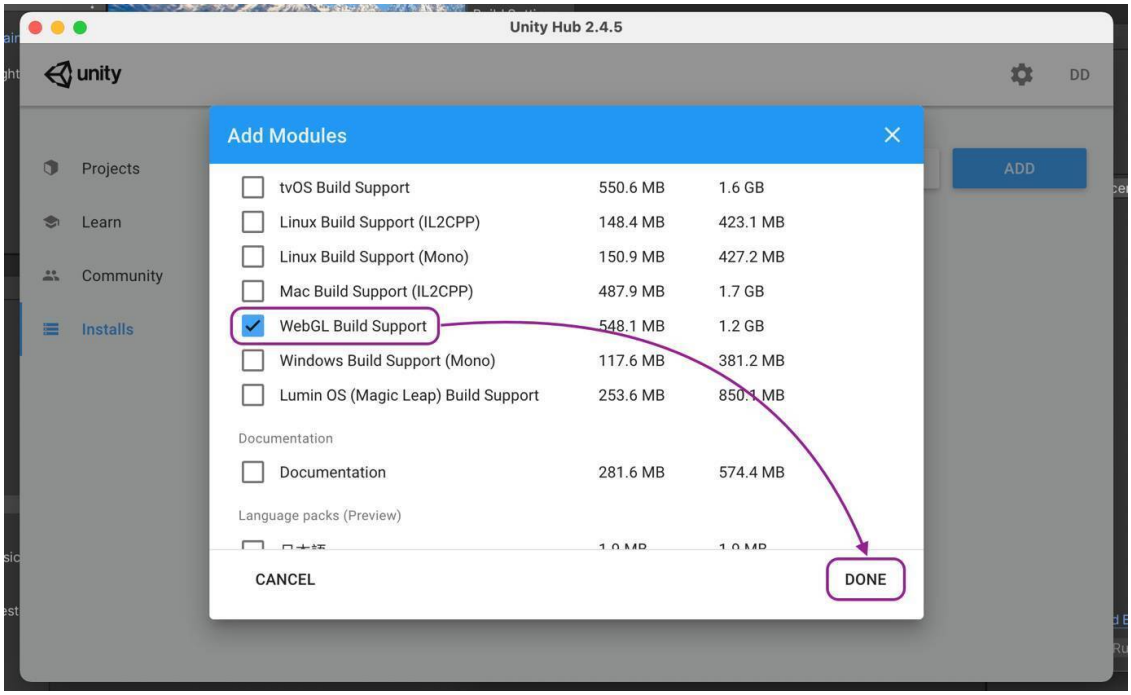
1. Чтобы перейти к сборке проекта, в верхнем меню выберите File – Build Settings.



2. Убедитесь, что в верхней части Build Settings загружено две сцены (мы их уже добавляли ранее чтобы настроить работу игрового меню). Далее в левой части того же Build Settings выберите платформу, под которую следует настроить сборку проекта – WebGL. Необходимые файлы для сборки проекта под web-версия мы не устанавливали, поэтому давайте сделаем это сейчас. Нажмите кнопку Install with Unity Hub:

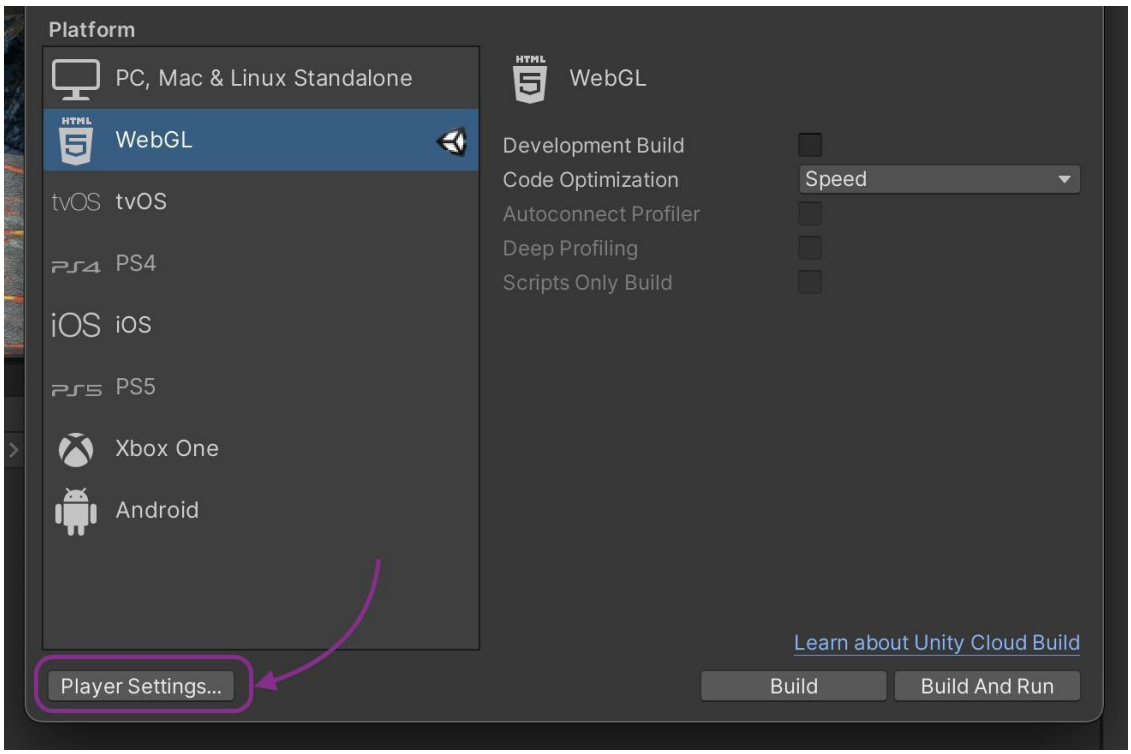


3. После этого откроется Unity Hub, в котором автоматически будет выбран модуль с именем WebGL Build Support для установки. Далее нажмите кнопку Done:



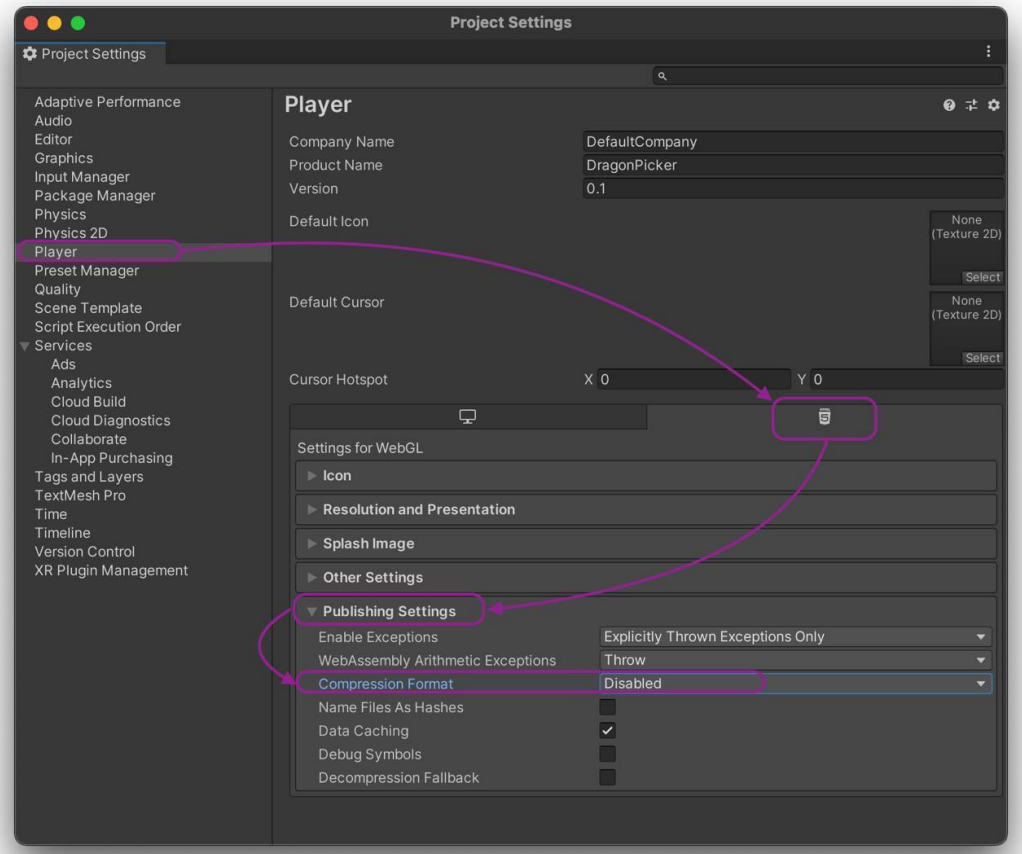
4. После того как загрузка и установка завершится, в Build Setting станет активна кнопка Switch Platform. Рекомендуется закрыть проект Unity и открыть его заново.

5. Снова откройте File – Build Settings, при выборе сборки под WebGL станет доступна кнопка Player Settings, нажмите ее:

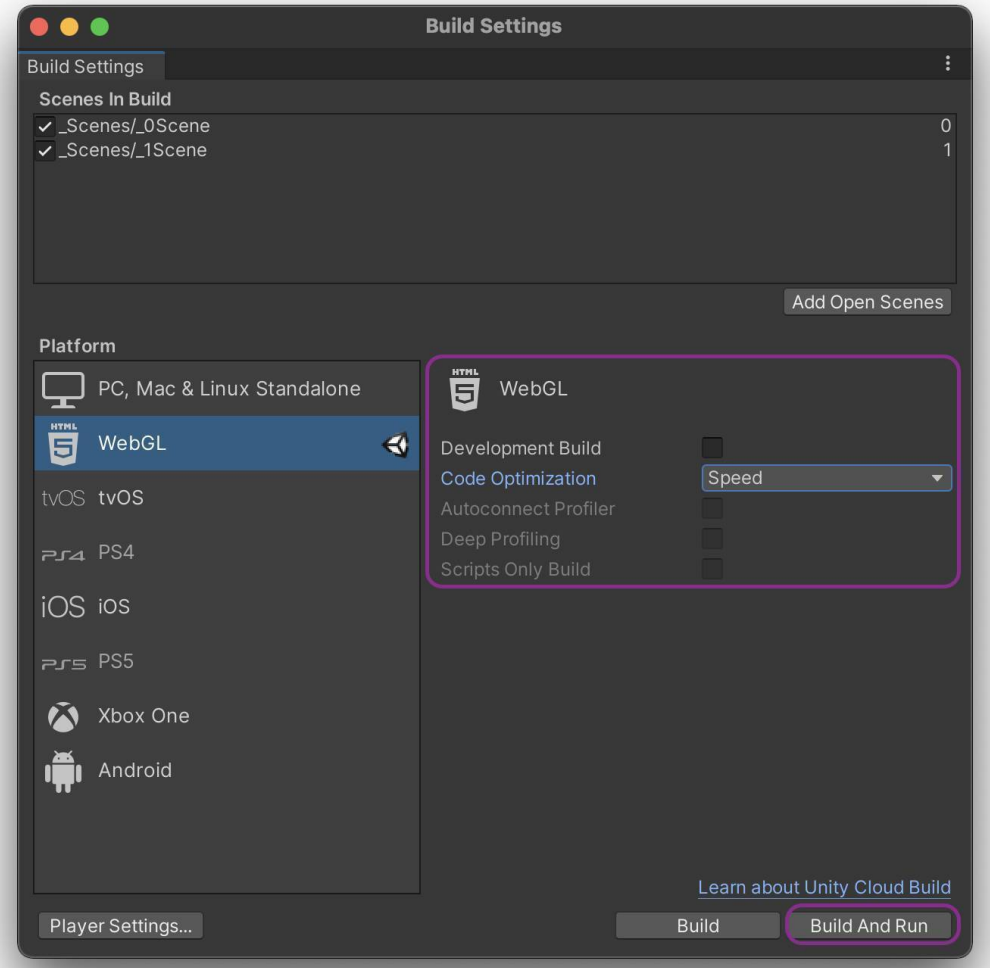


6. В настройках Player – Publishing Settings – Compression Format, установите значение Disable. Это требуется для того, чтобы мы могли корректно выгрузить сборку на портал simmer.io, т. к. на момент написания практикума портал не поддерживал сжатие в проектах,

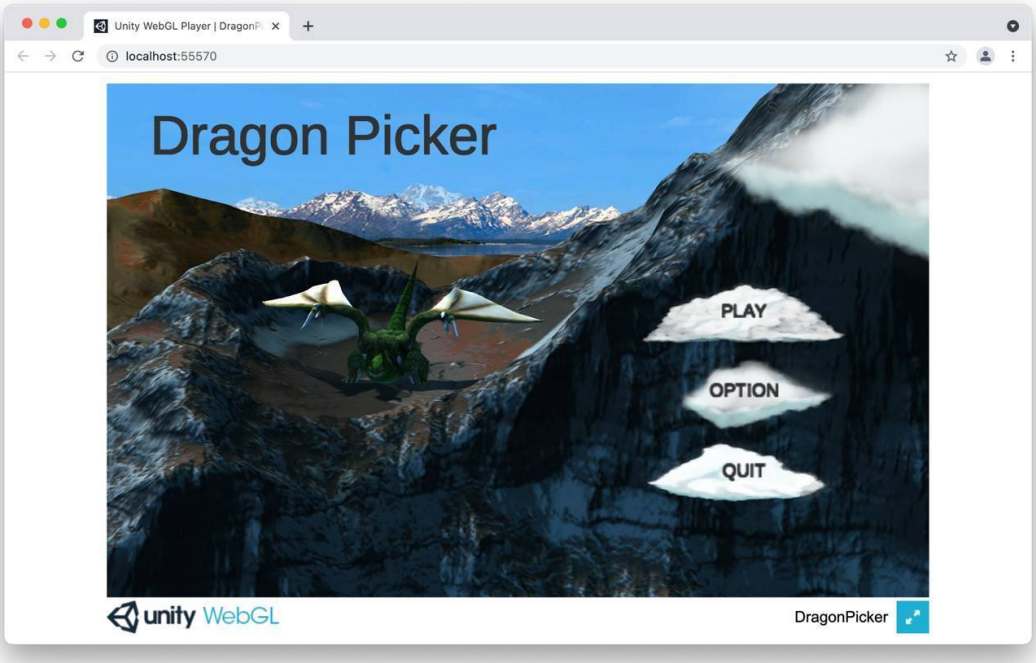
созданных в версии 2020 Unity. Стоит также поменять Company Name, так как некоторые версии Unity просто не дадут получить билд проекта с дефолтным названием компании.



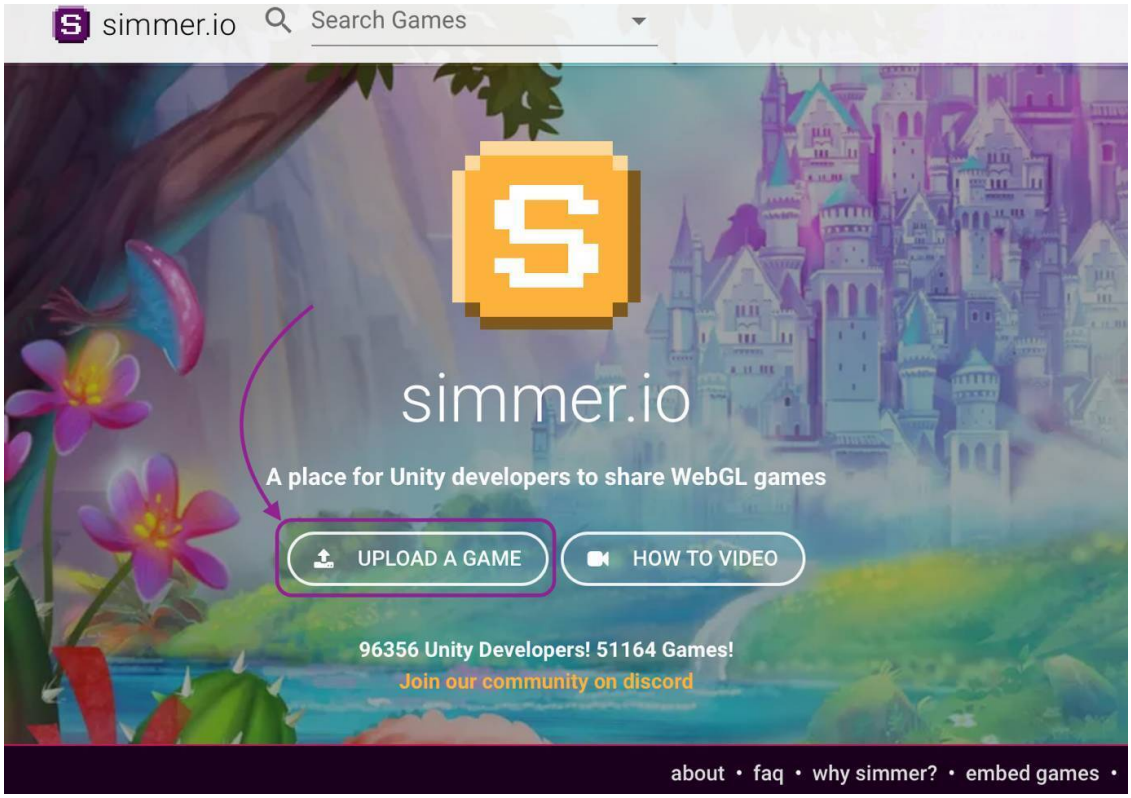
7. Закройте окно с настройками. После этого при выборе сборки под WebGL станет доступна кнопка Build And Run.



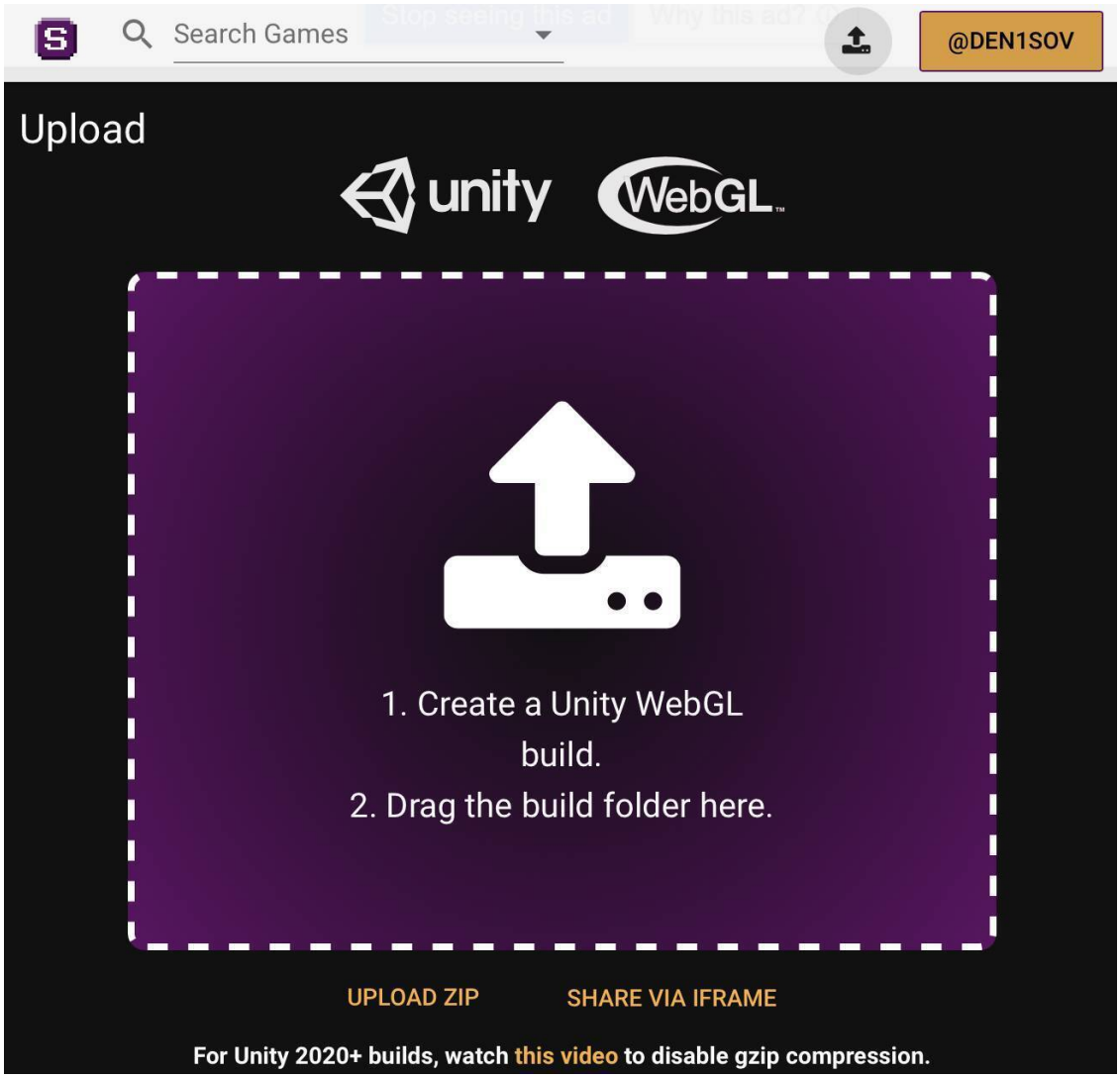
8. Вид приложения после сборки в окне браузера:



9. Теперь вы можете перейти на сайт https://simmer.io/upload по прямой ссылке, либо нажав соответствующую кнопку в шапке сайта simmer.io:

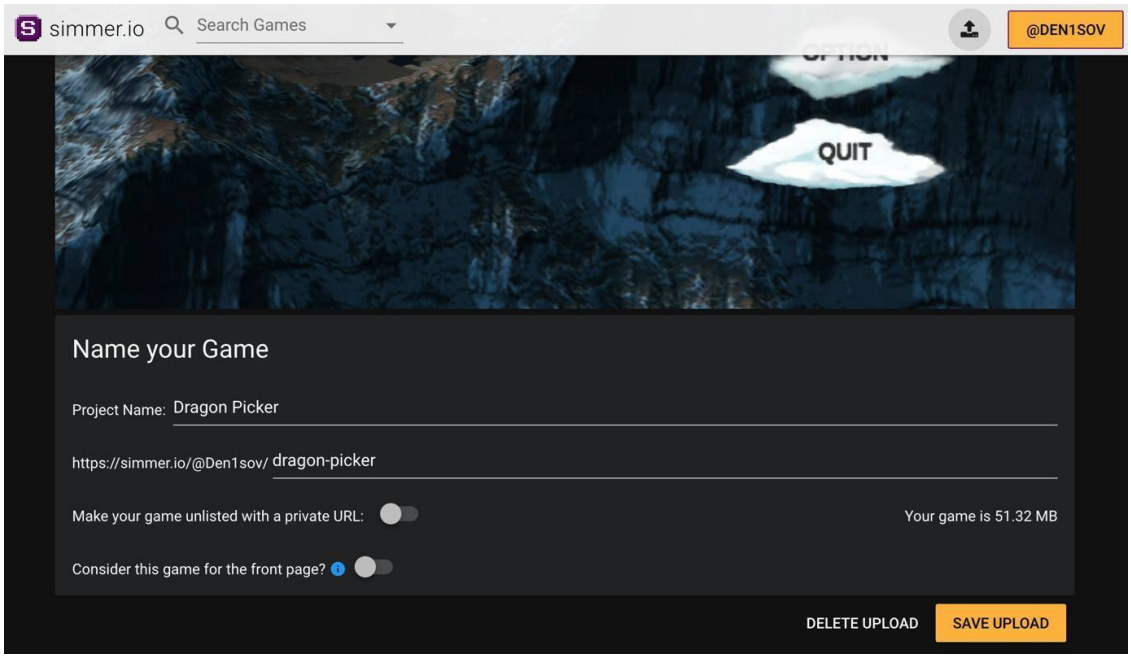


10. После того как вы пройдете стандартную процедуру регистрации на сайте (укажите почту, имя пользователя, пароль и т. д.) вам станет доступна функция загрузки приложения на сайт:



11. Как указано в подсказке, вам нужно просто перетащить папку с билдом игры (папка DragonPickerBuild).

12. После того как игра будет загружена, можно будет ввести ее название и задать постоянный адрес:



13. Нажмите кнопку Save Upload, вы перейдете на страницу описания игры. На ней можно будет оставить описание управления, прикрепить скриншоты и расставить теги. После того как завершите заполнение всех полей еще раз сохраните игру.

14. Поздравляем, теперь ваша игра доступна на портале SIMMER.io. Игра Dragon Picker была также загружена и доступна по ссылке: https://simmer.io/@Den1sov/dragon-picker.

Заключение

Компьютер – это самый удивительный инструмент, с каким я когда-либо сталкивался. Это велосипед для нашего сознания.

(c) Steve Jobs | Memory & Imagination, 1990 г.