

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

**Р.Г. БОЛБАКОВ, А.В. СИНИЦЫН, Г.В. ГОРБАТОВ, А.А.
АБРАМОВ**

КОМПЬЮТЕРНАЯ ГРАФИКА

Практикум по выполнению практических работ для студентов, обучающихся
по направлениям подготовки 09.03.04 «Программная инженерия» и 09.03.03
«Прикладная информатика»

Москва — 2020

УДК 004.92

ББК 30.2

Б-79

Компьютерная графика [Электронный ресурс]: Практикум / Болбаков Р.Г., Горбатов Г.В., Синицын А.В., Абрамов А.А. — М.: МИРЭА – Российский технологический университет, 2020. — 1 электрон. опт. диск (CD-ROM)

Практикум разработан в помощь студентам, выполняющим следующие практические работы: настройка интерфейса движка Unity; изучение базовых команд на языке C#, принципа работы и применение скриптов в Unity; освоение базового взаимодействия объектов на сцене; редактирование объекта Terrain; методы ввода и вывода данных через скрипт Unity; приданье объекту на сцене движения через скрипт C#; изучение свойств и принципов работы с анимацией и аниматором; работа с аудиофайлами и их проигрыванием в Unity; работа с системой частиц; свойства и принципы работы с UI элементами в Unity; создание пользовательского интерфейса для меню и изучение работы билдинга проектов.

В состав практикума входят: цель работы; задачи для достижения поставленной цели; описание выполнения работы практической работы;

Предназначено для проведения практических работ по дисциплине «Компьютерная графика» по направлениям подготовки 09.03.04 «Программная инженерия», 09.03.03 «Прикладная информатика».

Практикум издается в авторской редакции.

Авторский коллектив: Болбаков Роман Геннадьевич, Горбатов Григорий Витальевич, Синицын Анатолий Васильевич, Абрамов Артемий Алексеевич.

Рецензенты:

Холопов Владимир Анатольевич, к.т.н., доцент, заведующий кафедрой промышленной информатики, Институт информационных технологий, РТУ МИРЭА

Системные требования:

Наличие операционной системы Windows, поддерживаемой производителем.

Наличие свободного места в оперативной памяти не менее 128 Мб.

Наличие свободного места в памяти постоянного хранения (на жестком диске) не менее 30 Мб.

Наличие интерфейса ввода информации.

Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА — Российский технологический университет.

Объем: 8.38 мб

Тираж: 10

Оглавление

Практическая работа 1	4
Практическая работа 2	12
Практическая работа 3	18
Практическая работа 4	25
Практическая работа 5	31
Практическая работа 6	37
Практическая работа 7	45
Практическая работа 8	51
Практическая работа 9	58
Практическая работа 10	63
Практическая работа 11	72
Практическая работа 12	79
Практическая работа 13	89
Практическая работа 14	99
Практическая работа 15	104
Практическая работа 16	113
Список литературы.....	132

Практическая работа 1

Цель практической работы: изучить и настроить интерфейс движка Unity.

Задачи:

- Установить Unity и Unity Hub;
- Изучить основные элементы интерфейса Unity;
- Настроить пользовательский интерфейс;
- Разобрать основной функционал.

Описание выполнения работы

Перед началом работы необходимо скачать Unity и Unity Hub с официального сайта: <https://unity.com/ru>.

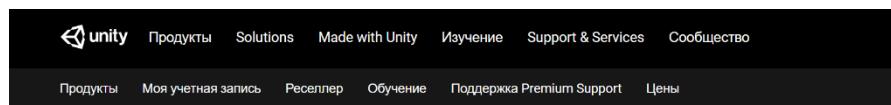
Скачиваем персональную или же студенческую версию, для студенческой нужно предварительно зарегистрироваться (Рисунок 1).

The screenshot shows the Unity website's pricing section. At the top, there are two tabs: 'Физическое лицо' (Physical person) and 'Организация' (Organization). Below these tabs, there are three main subscription options:

- Студент** (Student):
Learn the tools and workflows professionals use on the job.
Бесплатно (Free).
Sign up | Подробнее (Details).
Ограничения: Students enrolled in an accredited educational institution of legal age to consent to the collection and processing of their personal information, e.g., age 13 in the US, 16 in the EU. Must join the GitHub Student Developer Pack to be verified.
Checklist:
 - ✓ Новейшая версия базовой платформы разработки Unity
 - ✓ Unlimited access to Learn Premium
 - ✓ Five seats of Unity Teams Advanced
 - ✓ Темная тема пользовательского интерфейса
 - ✓ Диагностика реального времени в обложке
- Personal**:
Начните творить с бесплатной версией Unity.
Бесплатно (Free).
Начать | Подробнее (Details).
Ограничения: оборот или объем привлеченных инвестиций не превышает 100 тыс. \$ за последние 12 месяцев.
Checklist:
 - ✓ Новейшая версия базовой платформы разработки Unity
 - ✓ Ресурсы для начинающих и учебные материалы по Unity
- Узнать о Premium**:
Осваивайте Unity на онлайн-занятиях под руководством экспертов и по учебным материалам.
Попробовать бесплатно | Подробнее (Details).
Входит в состав подписок Unity Plus, Pro и Enterprise.

Рисунок 1 — Официальный сайт Unity. Выбор версии подписки.

Устанавливаем Unity со стандартными настройками. Теперь скачиваем Unity Hub с того же сайта: <https://unity3d.com/ru/get-unity/download> (Рисунок 2).



Загрузить Unity

Добро пожаловать! Вы оказались здесь, поскольку хотите загрузить Unity, самую популярную в мире платформу разработки для создания многоплатформенных 2D- и 3D-игр и интерактивного контента.

Перед загрузкой выберите подходящую для себя версию Unity.

[Выберите Unity + загрузку](#) [Загрузить Unity Hub](#)

[Подробнее о новом Unity Hub.](#)

Загрузите бета-версию Unity

Получите доступ к нашей новейшей функциональности раньше всех. Ваши отзывы помогут нам повысить ее качество.

[Загрузите бета-версию](#)

Рисунок 2 — Страница загрузки Unity Hub.

Устанавливаем Unity Hub и открываем его. Unity Hub сам должен обнаружить установленные версии на компьютере, но если этого не происходит, то для Unity Hub нужно указать путь к UnityEditor через вкладку Install -> Locate и указываем путь к эдитору, который установлен на PC.

Теперь во вкладке Projects создаем новый проект через кнопку NEW, Template выбираем 3D, даем произвольное наименование проекта и нажимает кнопку Create (Рисунок 3).

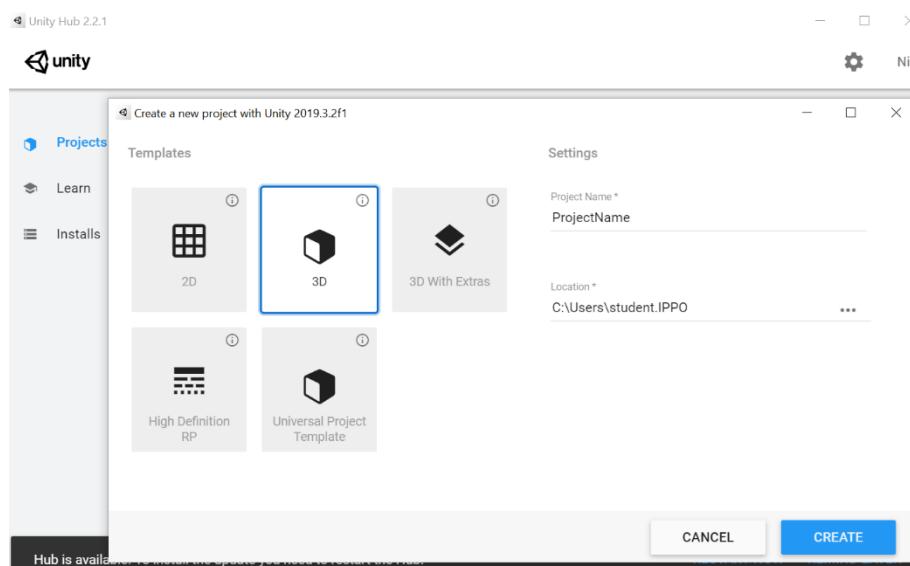


Рисунок 3 — Окно создания проекта через Unity Hub.

После создания чистого проекта, откроется сам редактор Unity (Рисунок 4).

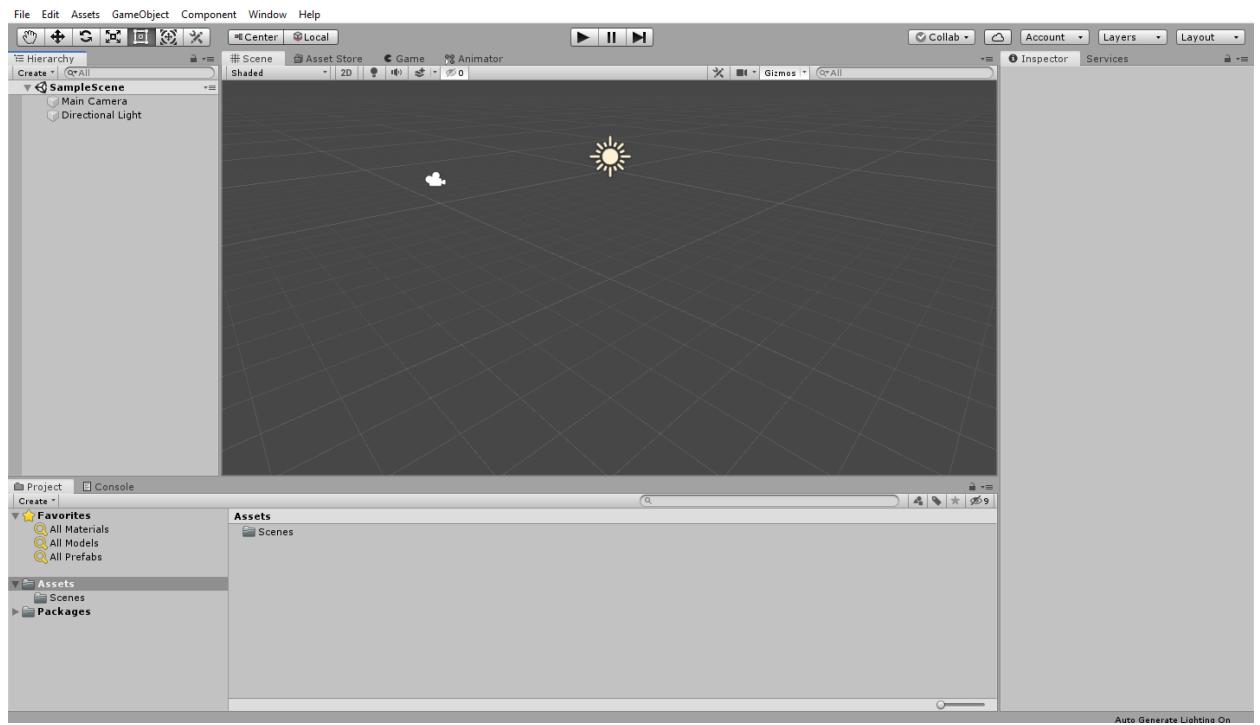


Рисунок 4 — Окно редактора Unity 2019.2.17f1.

Сразу стоит пройтись по основным элементам интерфейса и их функционалу. Hierarchy (Иерархия) по умолчанию находится в правом углу экрана (Рисунок 5).

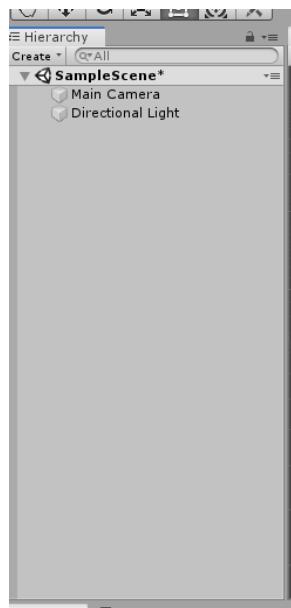


Рисунок 5 — Окно Hierarchy.

В нем отображаются сцены и все объекты в них. Эти объекты можно перемещать в иерархии, выделять, копировать, вкладывать один объект в

другой (В этом случае вложенный объект становится дочерним, и будет зависим от родительского). Стоит отметить, что сцен в проекте может быть несколько, их наполнение объектами может быть совершенно разное. Если говорить про конкретные примеры, то наиболее удачным будет смена локации в видео играх, когда персонаж уходит из одной сцены в другую, появляется экран загрузки, следующая сцена подгружается и персонаж оказывается в другом месте.

Одной из важных особенностей интерфейса редактора Unity является гибкость всех окон, размеры, местоположение и т.д., все можно подстроить под пользователя. Попробуйте переместить окно иерархии в левый нижний угол экрана и обратно, зажав левой кнопкой мыши на надписи Hierarchy.

В чистом проекте не все окна открыты сразу, чтобы просмотреть их все, нужно нажать кнопку Window (Рисунок 6), появится весь список окон, которые можно в дальнейшем открыть и добавить в пользовательский интерфейс.

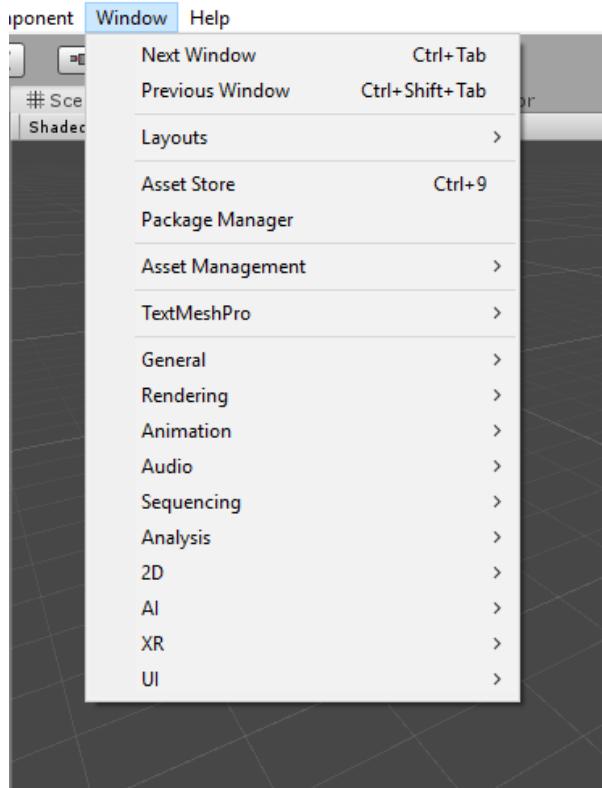


Рисунок 6 — Вкладка Window в редакторе Unity.

Над окном иерархии находится панель с инструментами для работы с объектами: перемещение объекта, поворот объекта, масштабирования объекта и т.д. (Рисунок 7)



Рисунок 7 — Панель инструментов.

Правее от панели инструментов находятся переключатели системы координат: глобальная и локальная. Глобальная система координат двигает объект относительно всей сцены. Локальная система координат относительно собственного поворота вокруг своей оси.

Под переключателями находится 4 окна, которые можно переключать (Рисунок 8):

- Scene – отображаемая сцена;
- Asset Store – встроенный в Unity браузер сайта Unity Asset Store. В нем можно скачивать различные файлы для разработки, например, 3D-модели, музыку, скрипты, материалы. Все файлы находятся в свободном доступе и их можно спокойной использовать в своих проектах, но не все они бесплатные;
- Game – здесь будет отображаться сцена при ее запуске;
- Animator – механизм, задающий логику и последовательность анимационных клипов.

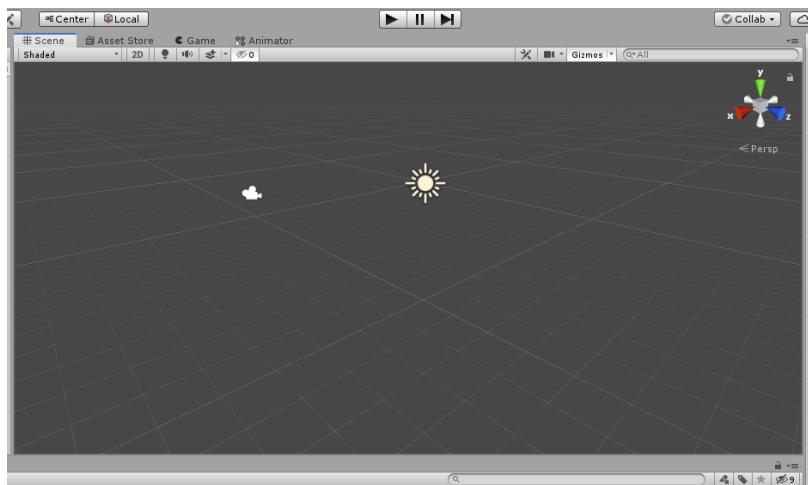


Рисунок 8 — Центральная часть редактора Unity.

Над окном со сценой находятся кнопки, с помощью которых можно запустить или остановить игру (Рисунок 9).



Рисунок 9 — Кнопка запуска сцены.

В правом углу экрана находится Inspector, здесь отображаются все компоненты и их свойства, прикрепленные к объекту (Рисунок 10).

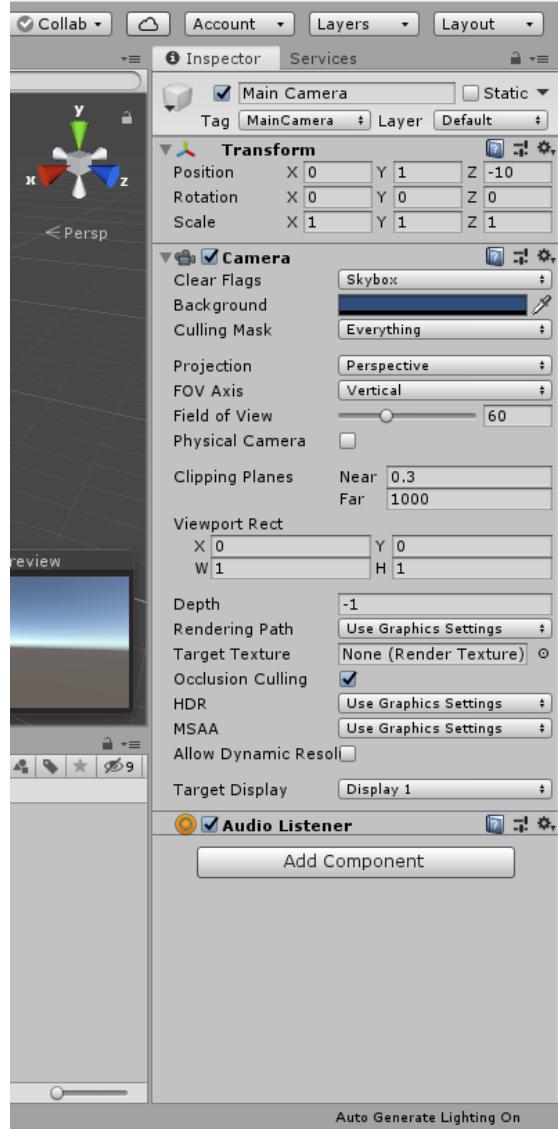


Рисунок 10 — Отображение компонентов Main Camera в Inspector’е.

В нижней части экрана расположены окна Project и Console (Рисунок 11). Окно Project предназначено для эффективной работы с файлами проекта, именно там будут отображаться все файлы, которые разработчик добавил в папку Assets. Файлы из Asset Store автоматически попадают в папку Assets.

В окне Console будут отображаться все системные сообщения: предупреждения, ошибки, дебаги.

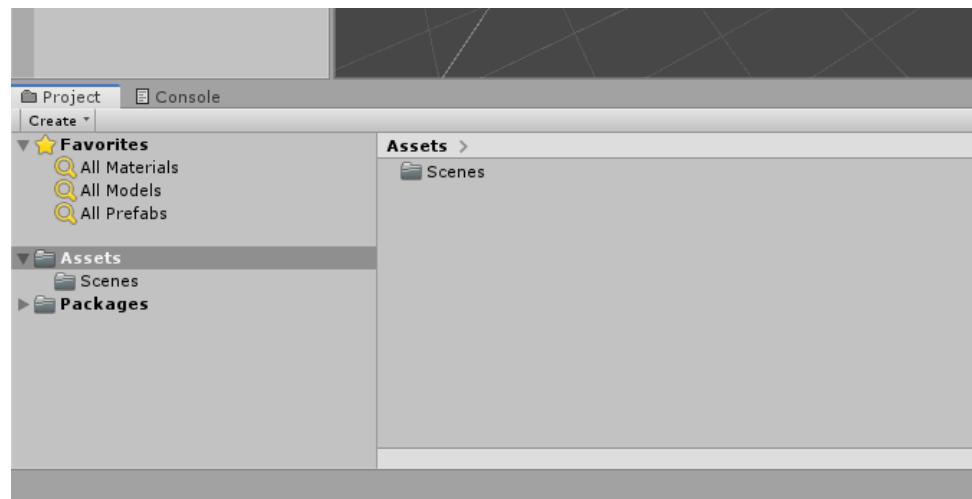


Рисунок 11 — Окно Project в редакторе Unity.

Практическая работа 2

Цель практической работы: изучить объекты и их настройки на сцене.

Задачи:

- Создать различные 3D объекты;
- Изучить их параметры;
- Импортировать 3D модель из 3ds MAX;
- Создать и наложить материал.

Описание выполнения работы

Создаем новый проект, чтобы создать новые объекты, необходимо в окне иерархии с помощью правой кнопки мыши вызвать контекстное меню, выбрать из него 3D Object и нажать на Cube, потом на Sphere и на Capsule (Рисунок 1).

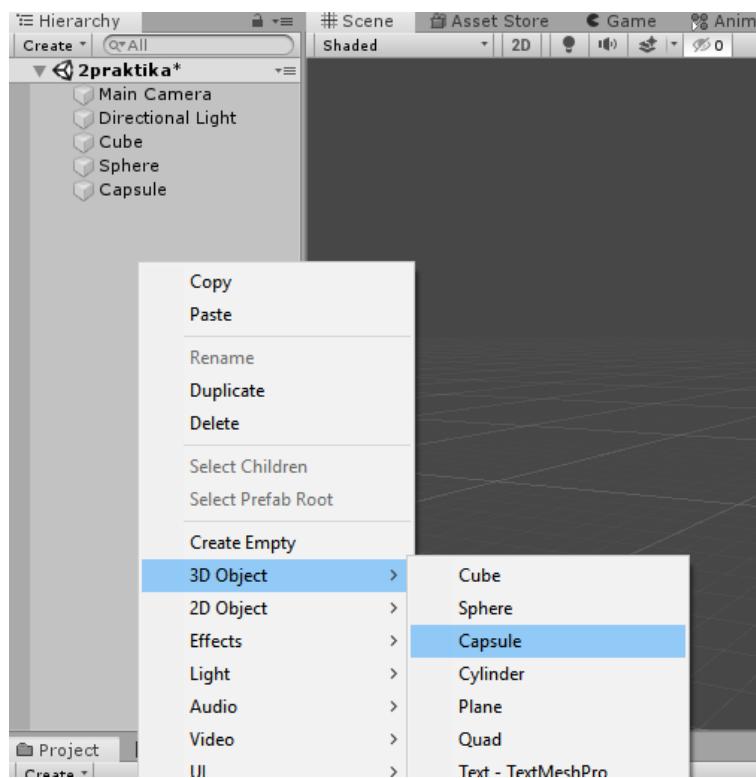


Рисунок 12 — Создание нового 3D объекта.

Все эти объекты появились в одной точке, чтобы их передвинуть, необходимо выделить объект, нажав на него левой кнопкой мыши, нажать клавишу W и перетянуть за одну из стрелочек (Рисунок 2).

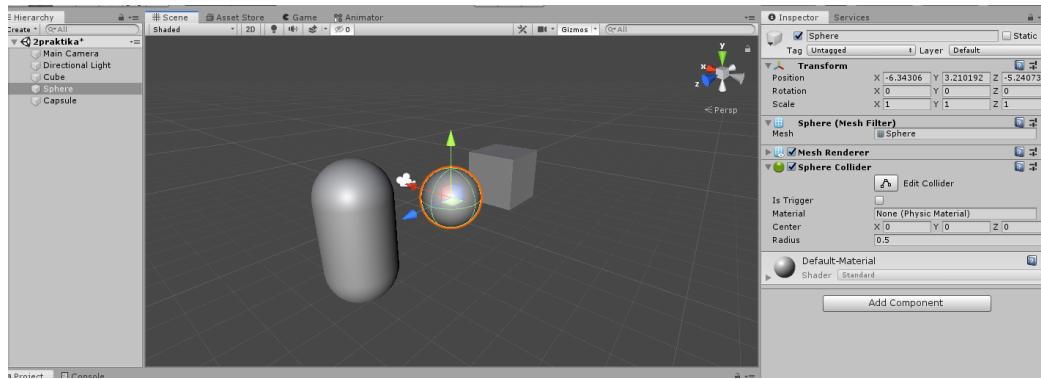


Рисунок 2 — Передвижение объекта на сцене.

Справа, в окне инспектора, можно заметить, что при движении объекта меняется параметр Position в компоненте Transform. Как видно на рисунке 2, у компонента transform есть три параметра:

Position — показывает координаты объекта относительно мировой системы или относительно локальной системы, если объект является дочерним.

Rotation — Отображает радиус поворота объекта по трем осям.

Scale — Масштаб объекта по трем осям.

Теперь необходимо создать новый произвольный объект в среде 3DS MAX и импортировать его в Unity.

После завершения моделирования необходимо задать UV координаты текстуры, поэтому на модель нужно применить модификатор UVW Map, и задать подходящие настройки для объекта (Рисунок 3).

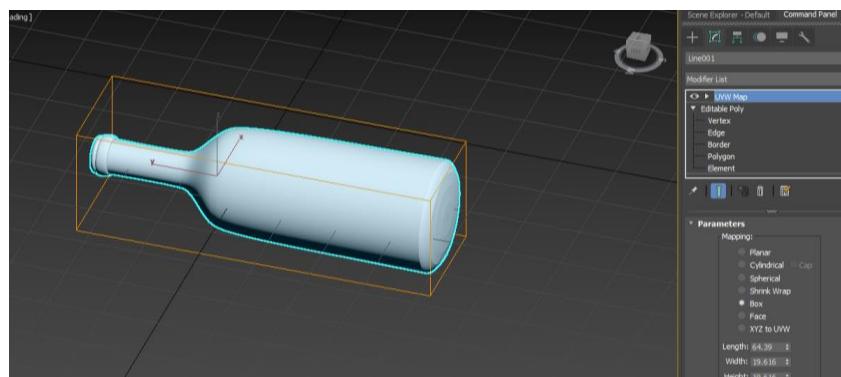


Рисунок 3 — Применение модификатора UVWMap к объекту.

Заходим в вкладку File и нажимаем Export – Export... (Рисунок 4).

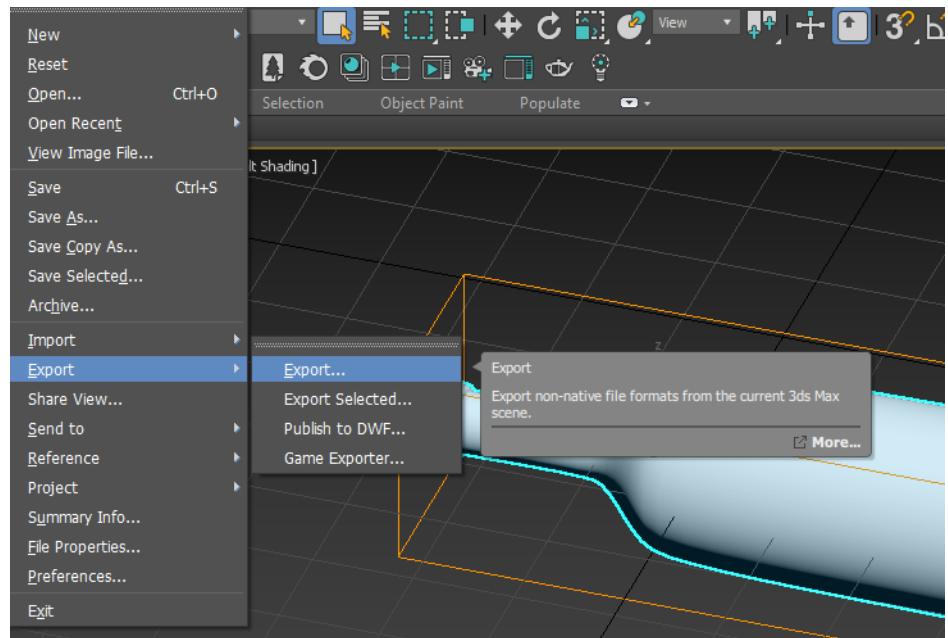


Рисунок 4 — Экспорт файла из 3DS MAX

В окне сохранения выбираем формат fbx, новое наименование и сохраняем (Рисунок 5).

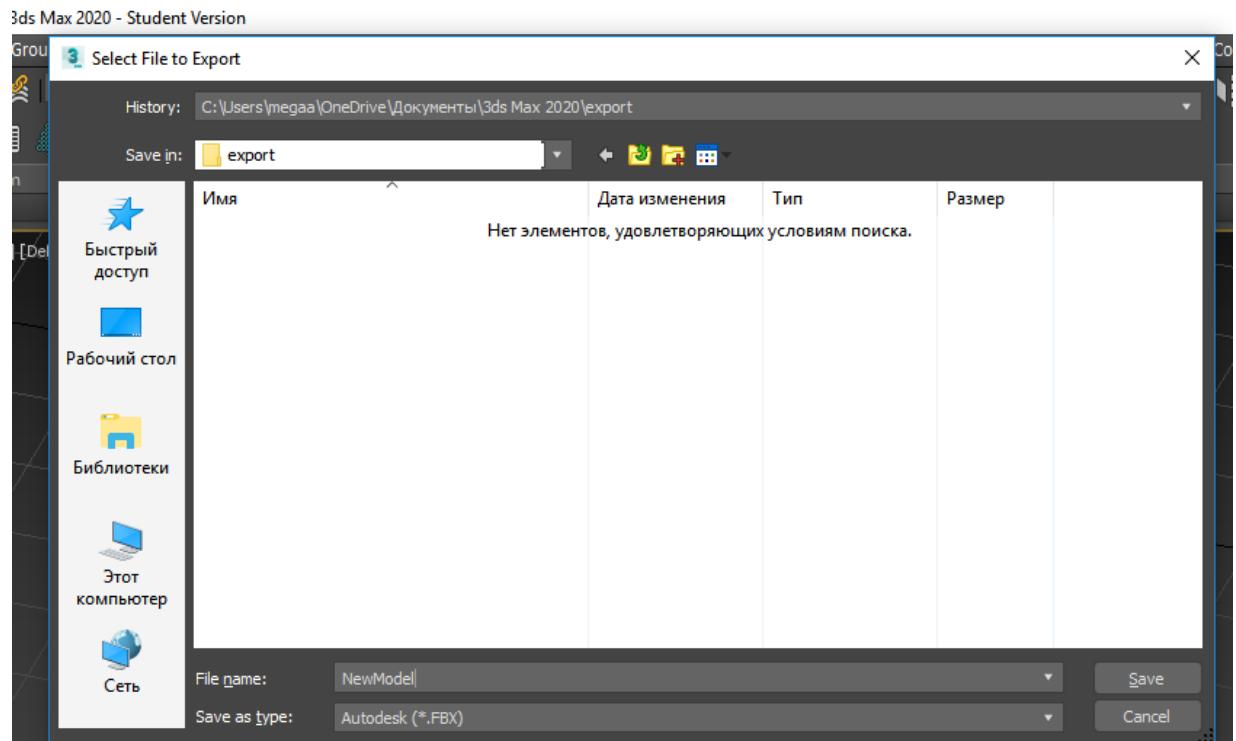


Рисунок 5 — Сохранение файла fbx

Сохраненный fbx-файл просто переносим в окно Assets в Unity и автоматически создается Prefab. С зажатой мышкой выносим prefab на сцену и объект появляется на сцене. Могут быть проблемы с масштабом объекта и

позицией, поэтому следует скопировать координаты position из созданных примитивных объектов в добавленный, а также подкорректировать значение Scale до оптимального (Рисунок 6).

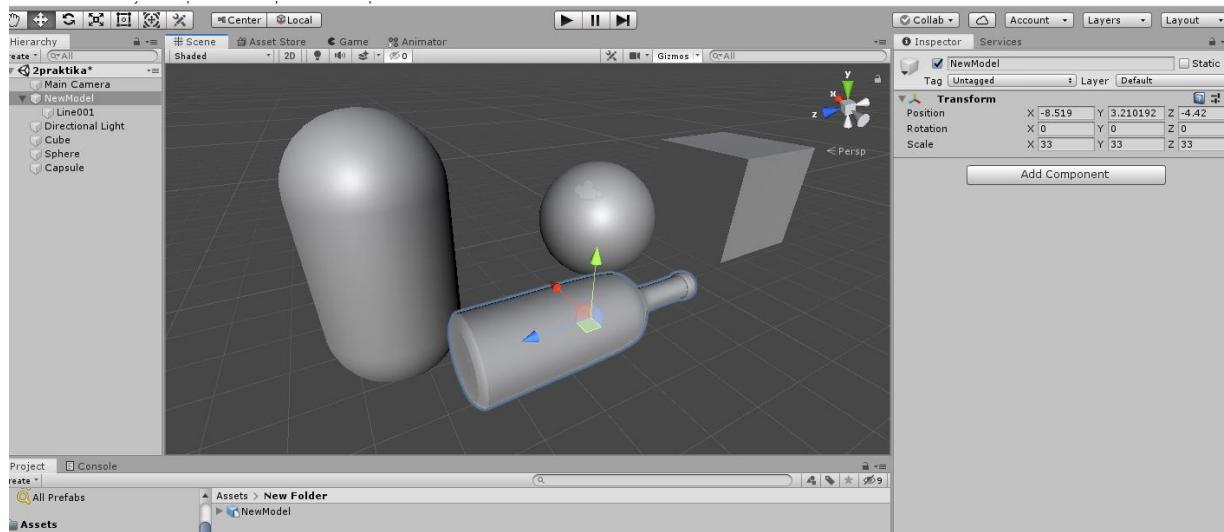


Рисунок 6 — Добавление модели на сцену.

Теперь создадим материалы для объектов, для этого в окне Assets нажимаем правую кнопку мыши и добавляем материал с названием BottleGlass (Рисунок 7).

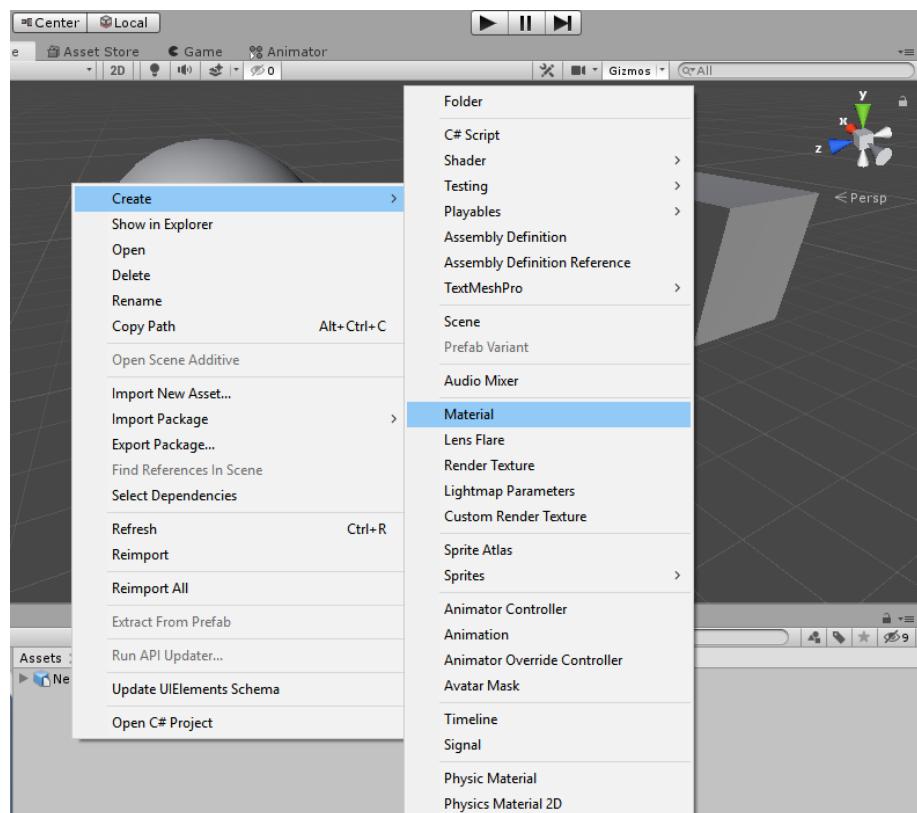


Рисунок 7 — Создание нового материала.

При его выделении он отобразиться в инспекторе, здесь есть множество параметров, но в этом случае, нужно будет создать стекло, для этого необходимо переключить Rendering Mode на Transparent, что задаст прозрачность материалу. Далее определим цвет и выберем подходящий на вкус. При выборе цвета есть четвертый ползунок, это альфа канал, он отвечает за видимость, снизим его до значения 0.3 (Рисунок 8).

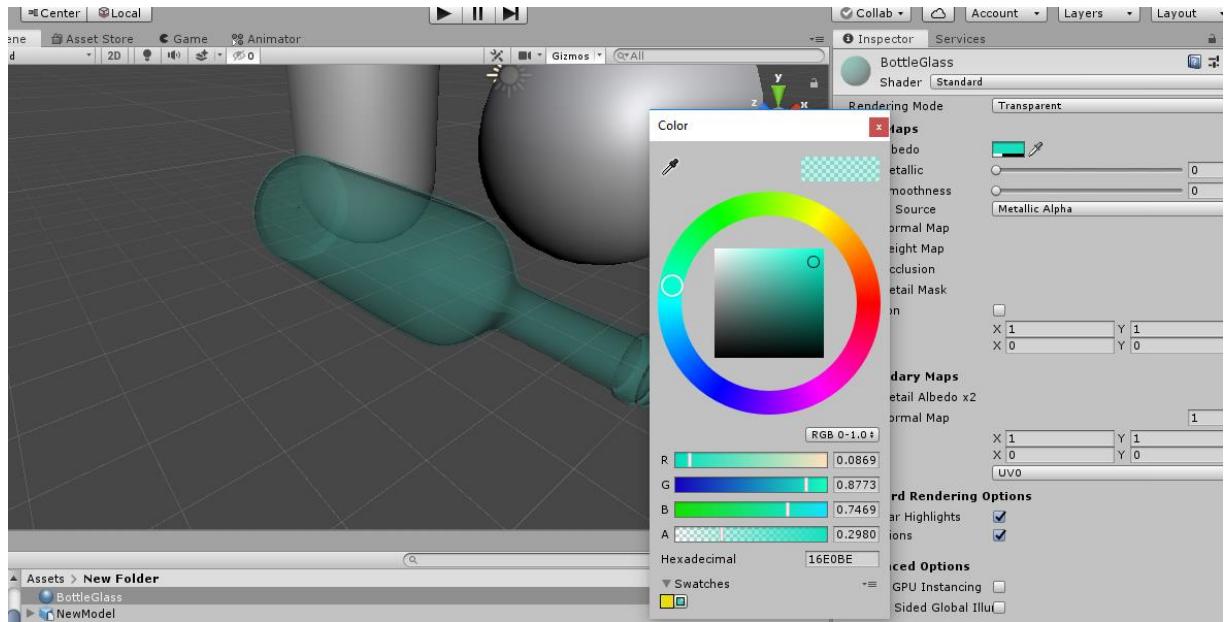


Рисунок 8 — Выбор цвета и альфа канала материала.

Параметр Metallic предполагает, что объект у нас будет отражать как металл, здесь он ни к чему, а вот параметр Smoothness, показывает, что материал гладкий и хорошо отражает, выставим ему значение до 1 (Рисунок 9).

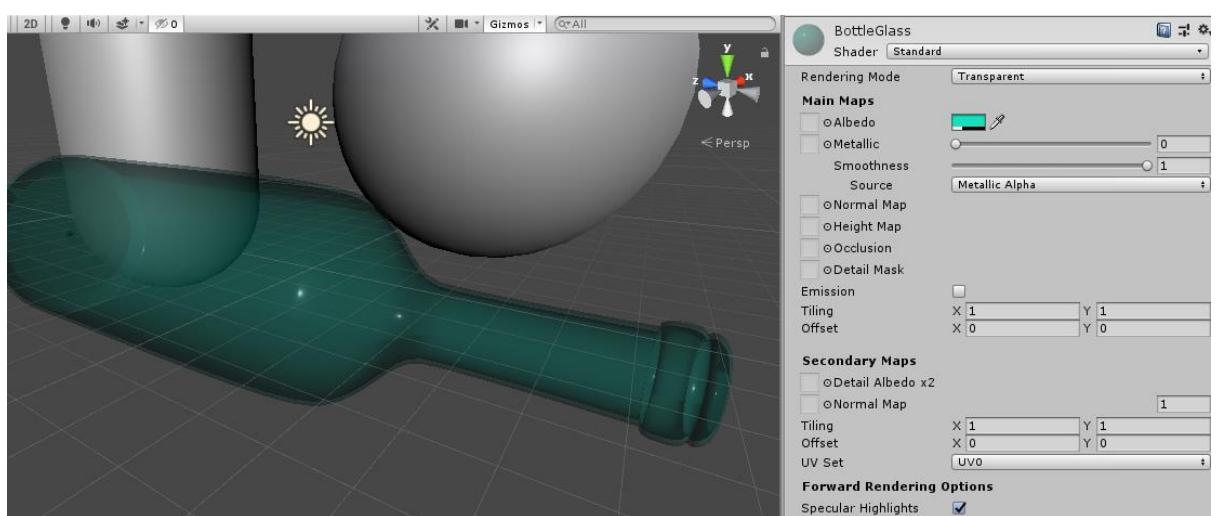


Рисунок 9 — Изменение значения Smoothness.

Стекло готово, теперь берем текстуру дерева из открытых источников и перетаскиваем в проект. Создаем новый материал под названием Wood и нажимаем на него. У материала есть такие параметры как Maps, они отображают некоторые изображения на объекте. В карту Albedo необходимо поставить карту текстуры дерева, просто перетаскиваем ее в окошко рядом с надписью и применяем этот материал на объект просто перетаскиванием его на куб (Рисунок 10).

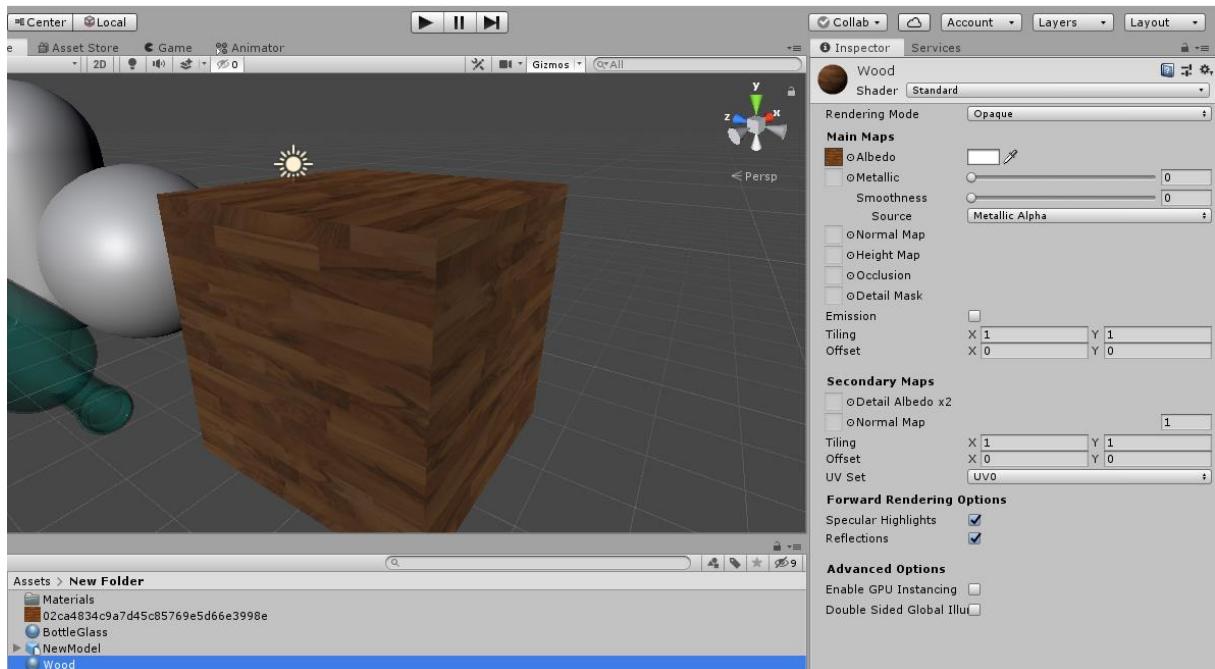


Рисунок 130 — Нанесение текстуры на материал.

Задаем материалу 0.5 Smoothness. Остался еще один важный параметр Tiling, это параметр, отвечающий за повторение изображение на материале. Для примера можно сравнить как выглядит материал при значениях 5/5 и 0.1/0.1. Таким образом можно редактировать текстуры. Найдите наилучшее значение для имеющейся текстуры и покажите преподавателю.

Практическая работа 3

Цель практической работы: изучить базовые команды на языке C#, принцип работы и применение скриптов в Unity.

Задачи:

- Изучить переменные, что из себя представляют и что они хранят;
- Изучить методы, Start(), Update();
- Написать скрипт для объекта;
- Применить скрипт к объекту;
- Показать преподавателю.

Описание выполнения работы

Переменные – это адресуемая область памяти. Адрес к этой памяти можно использовать для хранения в ней каких-либо данных. Саму переменную можно разложить на 3 составляющие: уровень доступа, тип данных и название переменной (Рисунок 1).

```
public string RandomWord;  
private GameObject RandomObject;
```

Рисунок 14 — Пример разных переменных.

Создадим C# скрипт (ПКМ в окне Assets, Create => C# Script) и откроем его в Microsoft VisualStudio, для этого нажимаем правой кнопкой мыши по скрипту в папке Assets и выбираем в выпадающем меню кнопку Open или просто нажимаем два раза на скрипт (Рисунок 2, 3). Unity начнет подгружать Microsoft VisualStudio и через некоторое время откроет сам редактор. По умолчанию Класс, который присутствует в скрипте называется тем именем, которое дается ему первоначально.

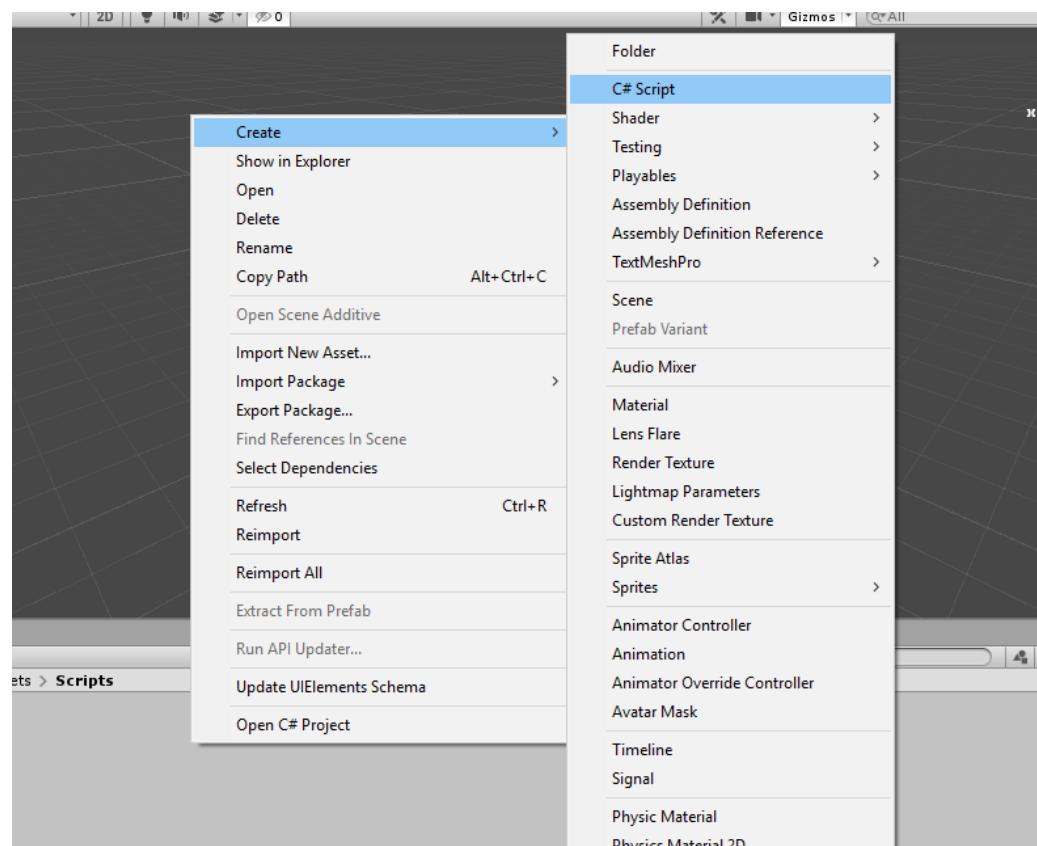


Рисунок 15 — Создание C# скрипта в Unity.

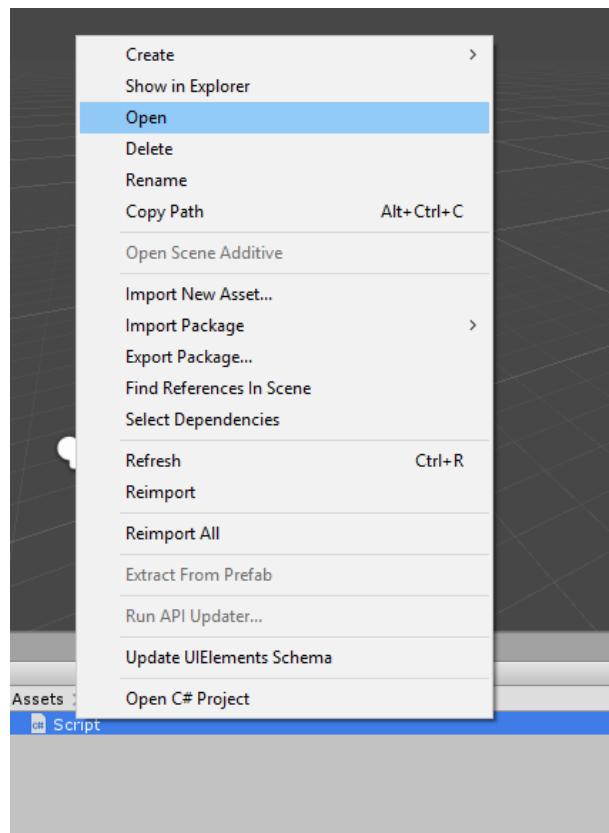


Рисунок 16 — Открытие скрипта в редакторе.

Создадим несколько переменных с уровнем доступа Public, одна будет типа float, другая типа GameObject. Сохраним изменения (Ctrl+S), вернемся в окно Unity, создадим произвольную 3D-фигуру, нажмем на новый объект и перетащим этот скрипт в окно инспектора справа, либо нажмем на Add component и введем в поиск название нашего скрипта (Рисунок 4).

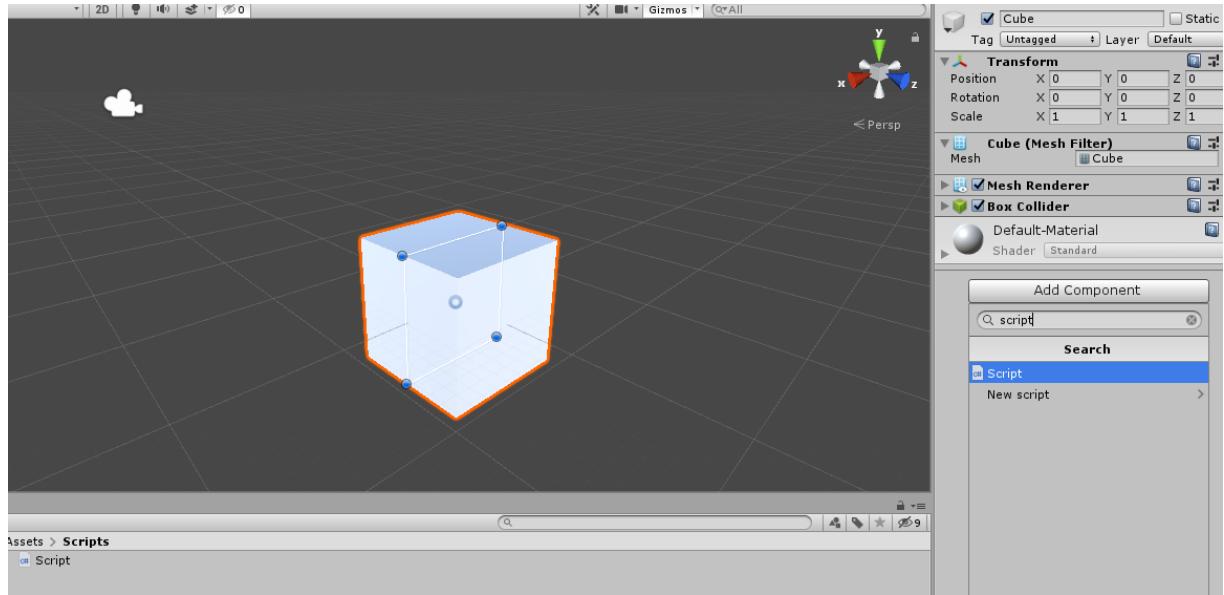


Рисунок 17 — Добавление созданного скрипта к объекту.

Если все сделано правильно, то в инспекторе будут отображаться переменные, которые мы написали в скрипте (Рисунок 5).

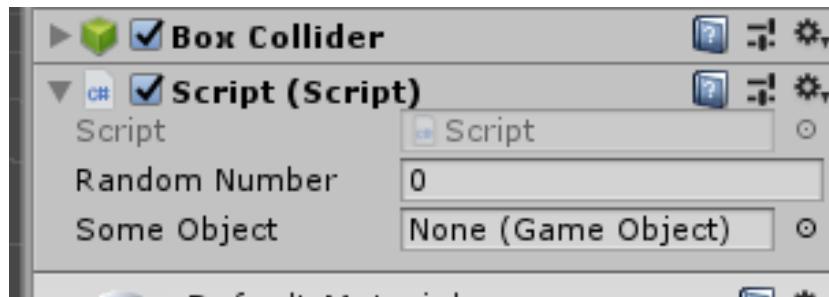


Рисунок 18 — Скрипт на объекте с редактируемыми переменными из инспектора.

Теперь нужно задать этим переменным то, что они будут хранить, первая переменная будет хранить число, а вторая любой объект со сцены. Для первой введем любое число, а для второй создадим второй 3D-объект и перетащим его в графу, где у нас должна храниться ссылка на объект (Рисунок 6).

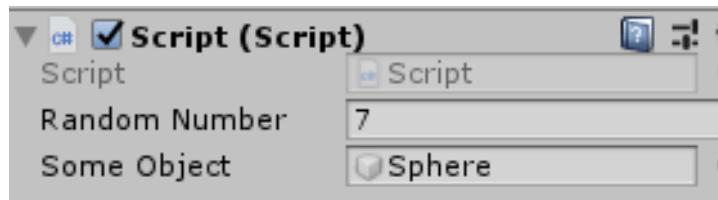


Рисунок 6 — Пример подставленных значений переменных.

Теперь, ради эксперимента, вернемся в скрипт и поменяем `public float` на `private float`, сохраним и посмотрим, что теперь будет отображаться в инспекторе на объекте. Теперь там отображается только ссылка на объект, числа нет. Public переменные дают доступ вне скрипта, такой уровень доступа можно применить, если есть задача назначить переменную через инспектор или получить доступ к переменной из другого скрипта. Private переменная скрыта от внешнего воздействия и ее можно использовать в рамках этого класса и зависящих от него. Хоть и переменная с уровнем доступа скрыта, мы все равно можем присвоить ей значение, добавим в скрипте к этой переменной такой код: `private float RandomNumber = 7;`

Теперь напишем в уже созданном методе `Start()` такую строчку:
`Debug.Log(RandomNumber);`

Эта команда используется для вывода значения переменных, такой инструмент полезен на этапе тестирования, чтобы знать промежуточные значения.

Сохраняем код, и запускаем игру, чтобы проверить, что переменная хранит число. Возвращаемся в Unity из вкладки Project переходим в Console и в этой вкладке должна появиться такая надпись (Рисунок 7). Значит все сделано правильно.

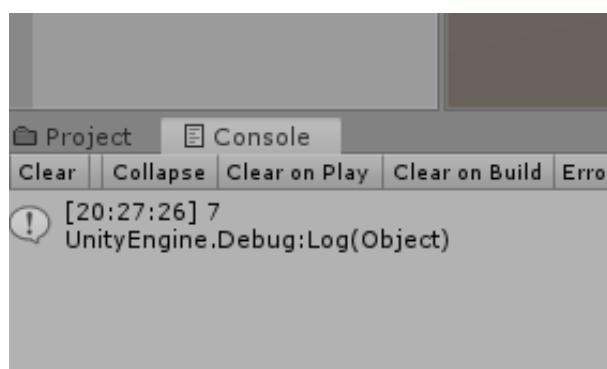
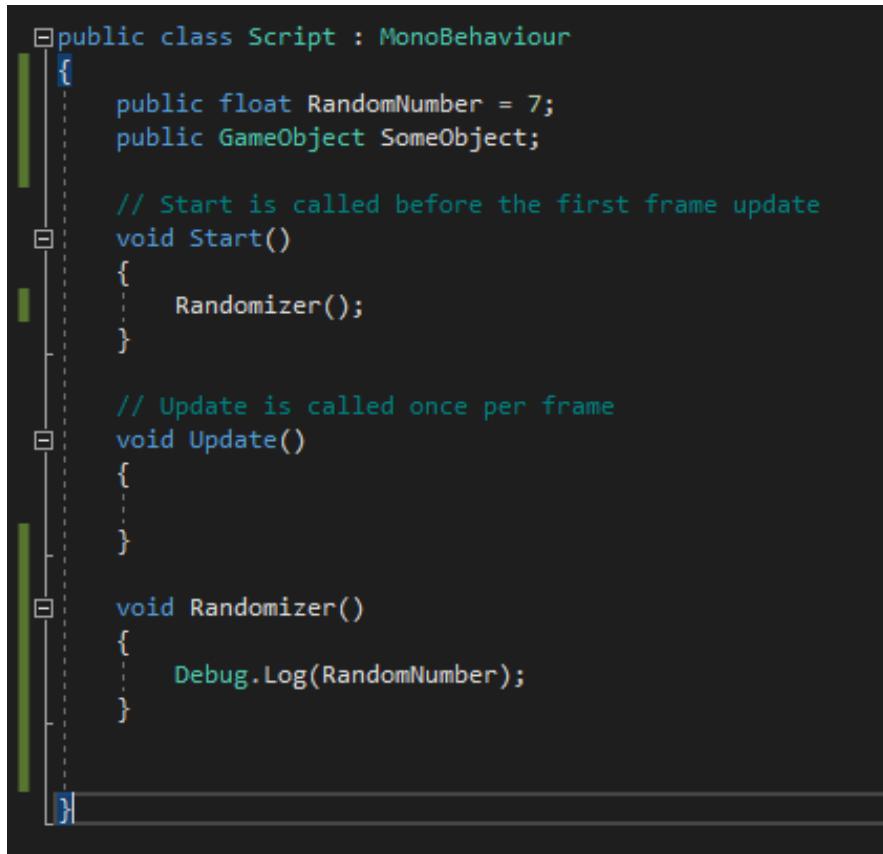


Рисунок 19 — Вывод значения в консоль.

Методы – это функция, которую выполняет класс или объект. Например, есть класс Человек, у него есть функция ходить, функция дышать, функция брать и т.д. другими словами методы нужны, чтобы осуществлять какие-то действия, над данными, над объектами и т.д. У методов есть тоже уровни доступа и типы данных, но это будет на следующих занятиях. Сейчас мы будем использовать тип данных void, такой тип данных не возвращает значение, он просто выполняет функцию. Создадим метод Randomizer, напишем туда уже созданную ранее строку и вызовем его в методе Start()(Рисунок 8). Если все выполнено правильно, в консоли в Unity будет выведено тоже, что и на рисунке 7.



```
public class Script : MonoBehaviour
{
    public float RandomNumber = 7;
    public GameObject SomeObject;

    // Start is called before the first frame update
    void Start()
    {
        Randomizer();
    }

    // Update is called once per frame
    void Update()
    {

    }

    void Randomizer()
    {
        Debug.Log(RandomNumber);
    }
}
```

Рисунок 20 — вызов метода Randomizer в методе Start.

Базовые методы Start() и Update() создаются по умолчанию, при создании нового скрипта. Это не значит, что их всегда нужно использовать. Start() реализует написанное в нем тело функции единожды, с запуском игры или активацией скрипта. В то время как Update() вызывается с каждым

кадром, фактически постоянно. Если мы вызовем наш метод рандомайзер в Update(), то мы получим кучу сообщений в консоль.

Добавим в метод команду генерации случайных значений, в скобках укажем диапазон значений, которые будут генерировать и присвоим его в переменную RandomNumber (Рисунок 9), а вызов метода переместим в Update(). Теперь запустим игру и проверим консоль, появится большое количество сообщений, количество которых будет возрастать с каждым кадром и у всех сообщений значение переменной будет разное, потому как оно будет меняться каждый раз, когда обновляется кадр.

```
public class Script : MonoBehaviour
{
    public float RandomNumber = 7;
    public GameObject SomeObject;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        Randomizer();
    }

    void Randomizer()
    {
        RandomNumber = Random.Range(1, 100);
        Debug.Log(RandomNumber);
    }
}
```

Рисунок 21 — Присвоение переменной RandomNumber случайного числа каждый кадр.

Добавим в метод условие, сократим диапазон чисел со 100 до 10 и напишем следующий код: (Рисунок 10).

Теперь каждый раз, когда RandomNumber будет равен 3, в консоль будет выводиться сообщение “Проверка условия”

```
public class Script : MonoBehaviour
{
    public float RandomNumber = 7;
    public GameObject SomeObject;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Randomizer();
    }

    void Randomizer()
    {
        RandomNumber = Random.Range(1, 10);
        Debug.Log(RandomNumber);

        if (RandomNumber == 3)
        {
            Debug.Log("Проверка условия");
        }
    }
}
```

Рисунок 22 — Проверка значения на выполнение условия.

Продемонстрировать работу преподавателю.

Практическая работа 4

Цель практической работы: освоить базовое взаимодействие объектов на сцене.

Задачи:

- Изучить компонент триггер;
- Изучить компонент твердое тело (rigid body);
- Изучить компонент Mesh;
- Освоить методы коллизии и триггера;
- Применить полученные знания на сцене;
- Показать преподавателю.

Описание выполнения работы

В Unity мы работаем с 3D и 2D объектами, объекты как правило должны взаимодействовать друг с другом, притягиваться, отталкиваться, уничтожаться и т.д. Разработчики Unity позаботились о физике в движке, поэтому у нас уже есть все необходимые инструменты для полета мысли разработчика.

Collider – это компонент создающий вокруг объекта зону воздействия на другие объекты. Они бывают разные, как 2D, так и 3D, совершенно разных форм. Collider можно сделать пропускающим через себя объекты поставив в инспекторе галочку Is Trigger. Таким образом он станет триггером и будет реагировать на разные события через скрипт.

RigidBody – это компонент, который эмитирует твердое тело и добавляет физику объекту. Он подвержен столкновениям и гравитации. Еще стоит отметить, что благодаря этому компоненту можно через скрипт вмешиваться в физику объекта, добавлять ему дополнительные силы.

Mesh – это отдельный элемент, который представляет из себя сумму, граней, точек, сторон и полигонов объекта, определяющая и задающая объем.

Создадим новый объект на сцене, допустим куб, в нем автоматически добавится Mesh и BoxCollider. Создадим к нему еще один объект, сферу и поместим ее над кубом. Добавляем еще один объект Plane, это будет поверхностью, которая будет не подвержена силе притяжения. Добавим к фигурам компонент RigidBody через кнопку Add Component (Рисунок 1).

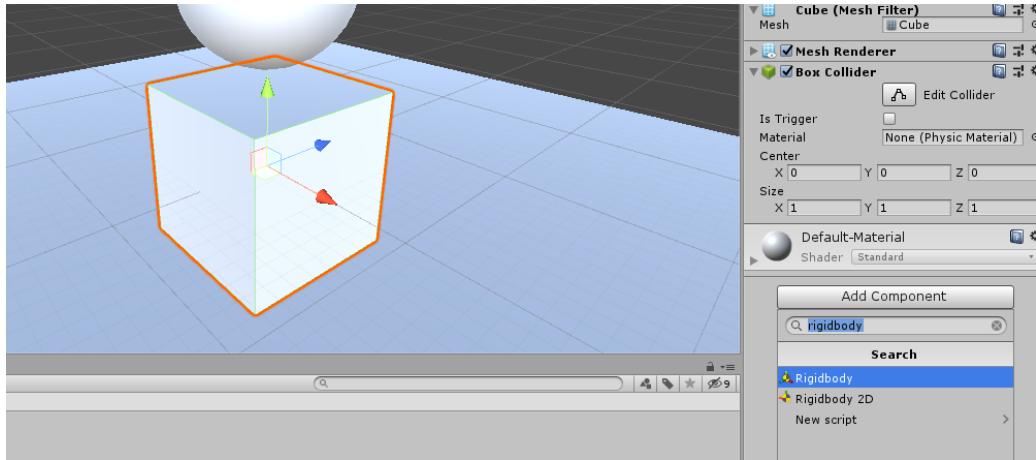


Рисунок 23 — Добавление компонента Rigidbody.

Запустим сцену и проанализируем процесс падения объектов на “пол”.

Теперь сделаем эту сферу похожей на резиновый мячик, который будет отскакивать от поверхностей. Создадим Physic Material, дадим ему название (Рисунок 2) и зайдем в его настройки в инспекторе. Зададим ему в параметре Bounciness 1 и поставим Bounce Combine на значение Maximum. Добавим новый материал на Collider (Рисунок 3).

Запускаем сцену.

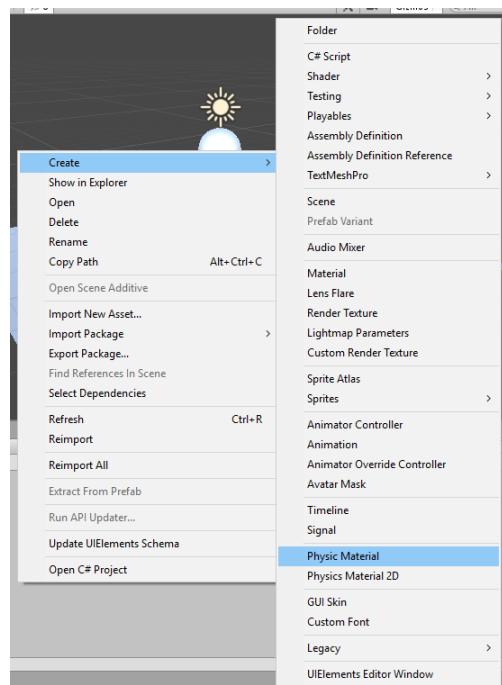


Рисунок 24 — Создание нового Physic Material.

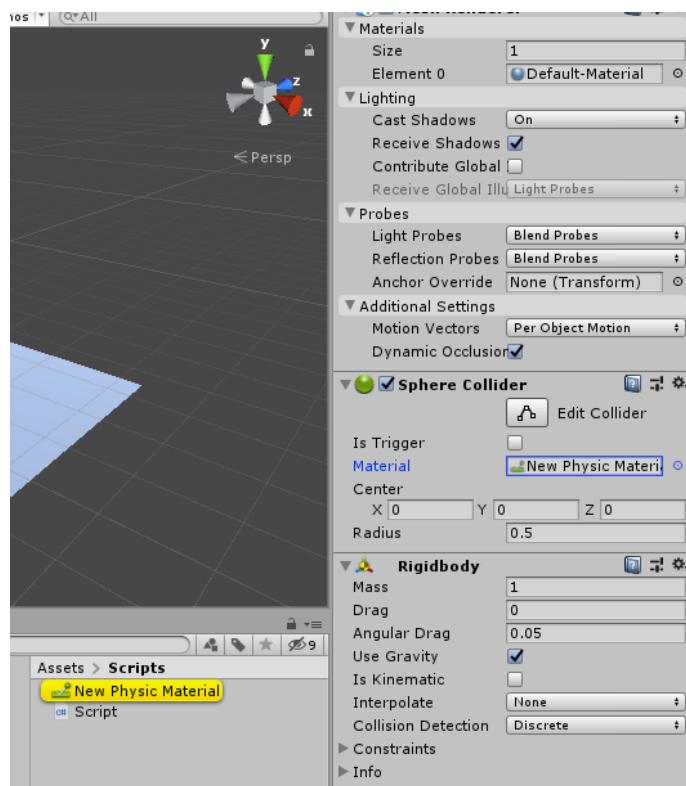


Рисунок 25 — Помещение нового материала в компонент Collider.

Создадим новый C# скрипт, откроем эдитор и напишем функцию, которая будет вызываться при столкновении. (Рисунок 4).

```
private void OnCollisionEnter(Collision collision)
{
}
```

Рисунок 26 — Пример написания функции OnCollisionEnter.

В теле метода необходимо написать тип Destroy() и указать ему в скобках, что нужно уничтожить, а именно самого себя, носитель скрипта. Это по умолчанию переменная gameObject (Рисунок 5).

```
private void OnCollisionEnter(Collision collision)
{
    Destroy(gameObject);
}
```

Рисунок 27 — Команда для самоуничтожения объекта.

Добавим этот скрипт объекту Сфера в инспекторе и запустим сцену. Когда сфера соприкасается с любым Collider'ом, она сразу же удаляется.

Теперь изменим метод в скрипте на другой, который будет вызываться при входе в объект, а не при соприкосновении (Рисунок 6).

```
private void OnTriggerEnter(Collider other)
{
    Destroy(gameObject);
}
```

Рисунок 28 — Пример написания функции OnCollisionEnter.

После запуска сцены видно, что метод не вызывается, шар лежит на кубе. Все потому что он лежит на его грани и не входит в сам объект, чтобы сделать объект проходимым, остановим плеер и у компонента Collider шара поставим галочку на пункте Is Trigger. Запустим еще раз сцену, теперь объекты работают правильно.

Теперь задача сделать так, чтобы все объекты, которые попадут под в сферу удалились. Для этого поменяем переменную, которая будет приниматься типом Destroy. Т.к. Метод триггера принимает чужой Collider, который встречается ему на пути. Collider хранится в локальной переменной other, таким образом, можно получить доступ к стороннему объекту через его же Collider. Как было описано ранее, по умолчанию объект записывается в своей переменной gameObject. Таким образом, получится строка: Destroy(other.gameObject).

Запускаем сцену, сфера удаляет все объекты на своем пути.

Усложним задачу, нам нужно удалять только определенные объекты, для этого в Unity есть механизм тегов.

Добавим тег на куб (Рисунок 7).

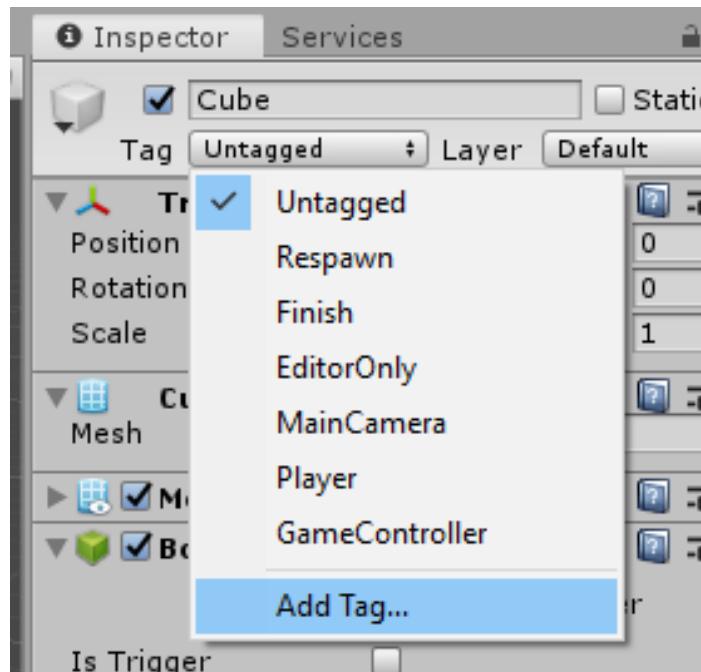


Рисунок 29 — Список Тегов, которые можно добавить на объект.

В появившемся окне нажимаем на плюс, вводим наименование тега “Cube” и сохраняем. Возвращаемся в инспектор куба, опять нажимаем на надпись Untagged и выбираем новый тег из списка.

Возвращаемся в скрипт и добавляем условие, согласно которому объект при входе в любой Collider будет проверять тег объекта (Рисунок 8, 9).

```
private void OnTriggerEnter(Collider other)
{
    if(other.tag == "Cube")
        Destroy(other.gameObject);
}
```

Рисунок 30 — Первый способ проверки тега объекта.

```
private void OnTriggerEnter(Collider other)
{
    if(other.CompareTag("Cube"))
        Destroy(other.gameObject);
}
```

Рисунок 31 — Второй способ проверки тега объекта.

На Рисунке 9 представлен еще один способ реализации, более быстрый, его рекомендуется использовать, но в конкретной задаче это не так важно.

При запуске сцены, сфера уничтожает только куб и свободно проходит через остальные объекты.

Результат продемонстрировать преподавателю.

Практическая работа 5

Цель практической работы: научиться редактировать объект Terrain и научиться связывать сцены между собой.

Задачи:

- Импортировать инструменты для редактирования Terrain;
- Изучить работу с инструментами;
- Нарисовать на Terrain биом;
- Расставить ассеты;
- Настроить освещение.

Описание выполнения работы

Terrain в отличии от Plane имеет гораздо больше возможностей редактирования. По своей сути, это окружение на сцене: горы, деревья, трава.

Перед началом работы импортируем standard assets. Это можно сделать через assets store, скачиваем Terrain Tools Sample Asset Pack и импортируем его в проект (Рисунок 1). Еще для работы на этом практическом понадобятся другие материалы для разработки: деревья, камни.

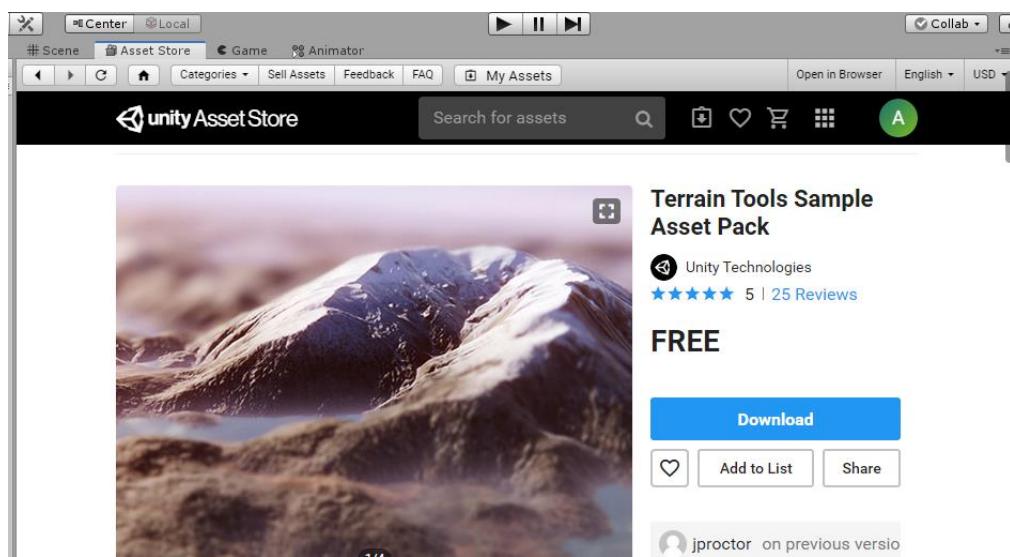


Рисунок 32 — Страница Terrain Tools в Asset Store’е.

Теперь необходимо создать сам объект Terrein через правую кнопку мыши в иерархии (Рисунок 2).

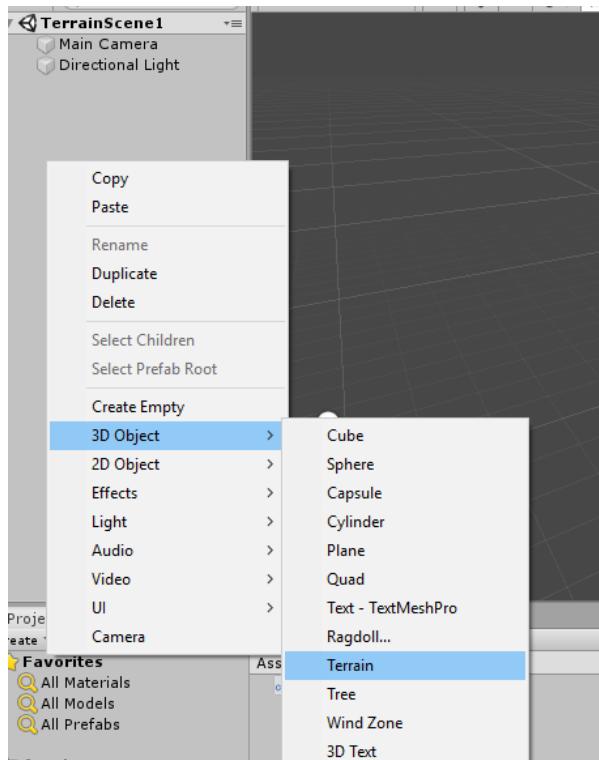


Рисунок 33 — Создание объекта Terrain.

Нажав на объект, в инспекторе будет отражено много различных опций. Сейчас нужно придать объем террейну, сделать горы и холмы. Для этого нужно зайти в инструмент Paint terrain.

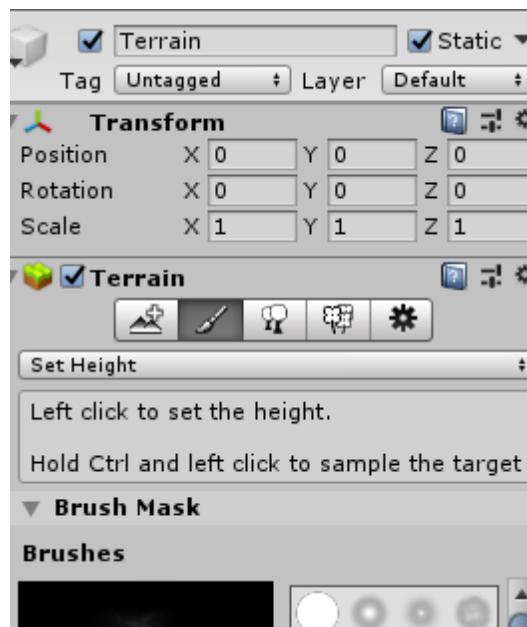


Рисунок 34 — Инструменты в Inspector'e Terrain'a.

Выбираем понравившуюся кисть, задаем ее размеры и максимальную высоту Height, до которой будет подниматься плоскость (Рисунок 4).

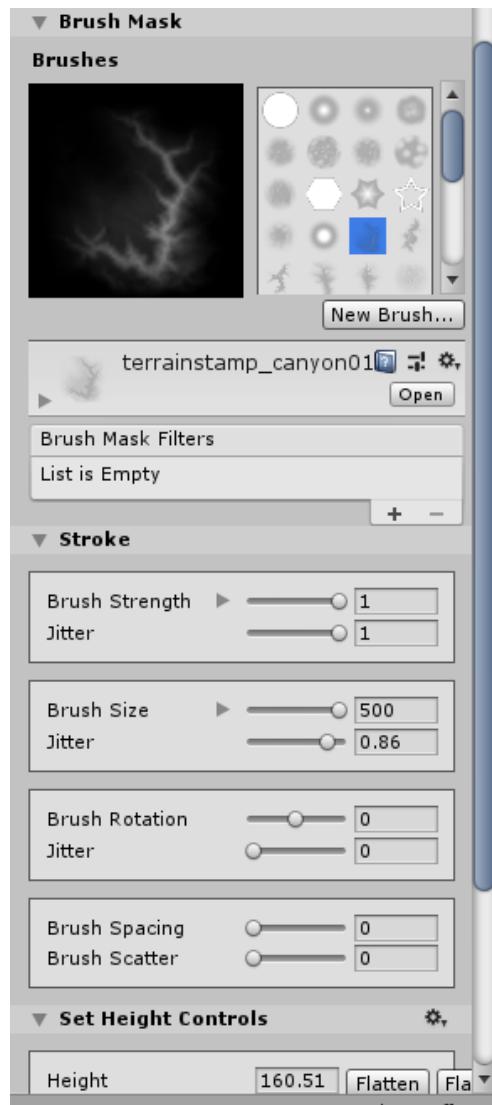


Рисунок 35 — Настройки для рисования высот и кисти.

Теперь переключаем режим на текстурирование (Рисунок 5), чтобы окрасить объект в заготовленные материалы.

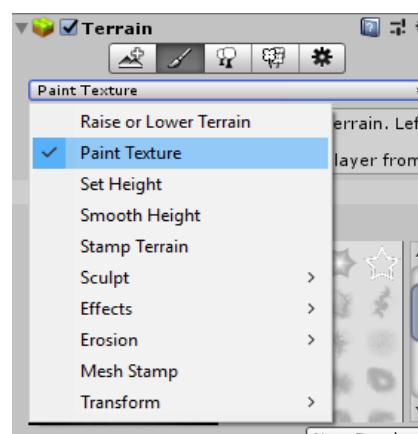


Рисунок 36 — Параметр для смены режима редактирования.

Для того, чтобы добавить новый слой текстур, нужно нажать на кнопку Add Layer и выбираем rock из ассета, который до этого импортировали в

проект (Рисунок 6). В настройках к нему (Layer Properties) указываем в параметре Tiling Settings - Size x = 10, y = 10.

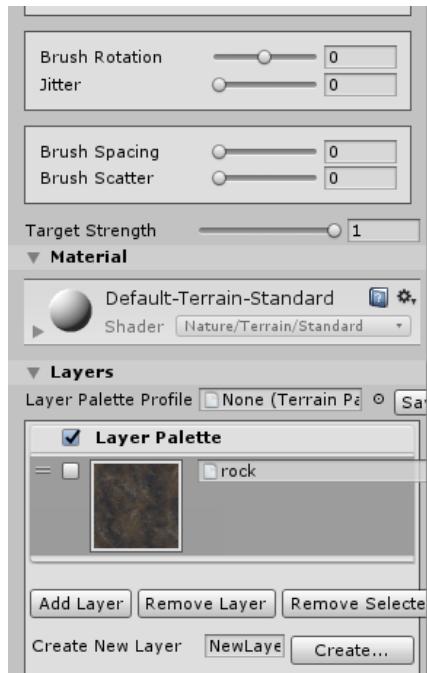


Рисунок 37 — Список слоев (Layers).

Добавляем еще парочку слоев таким же способом. Меняем размер кисти (Brush) под свое усмотрение и раскрашиваем Terrain. Стоит отметить, что не обязательно брать уже готовые слои, можно создать свой из подготовленной текстуры, через кнопку Create под списком слоев, и задать ему подходящий тайлинг.

Когда высоты построены, модель покрашена (Рисунок 7), можно приступить к наполнению сцены деревьями и травой. Для этого нужно перейти в специальные инструменты как Paint Trees (Рисунок 8).

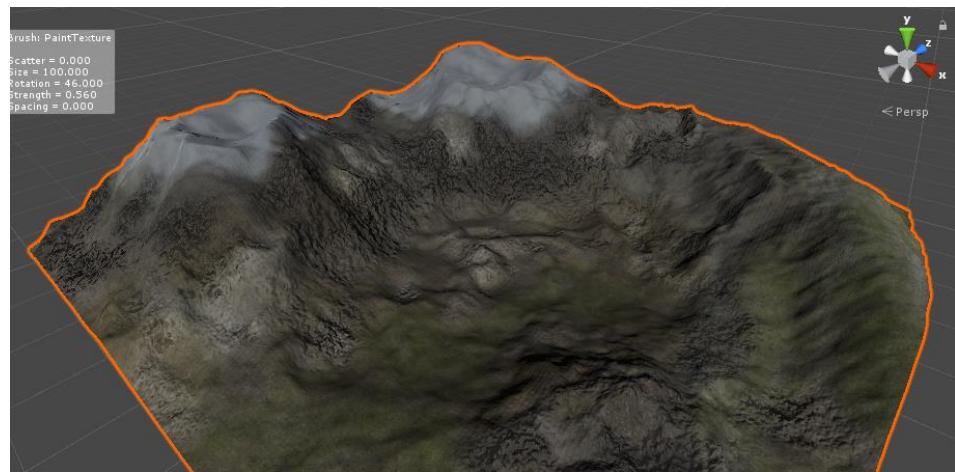


Рисунок 7 — Готовый объект к детализации.

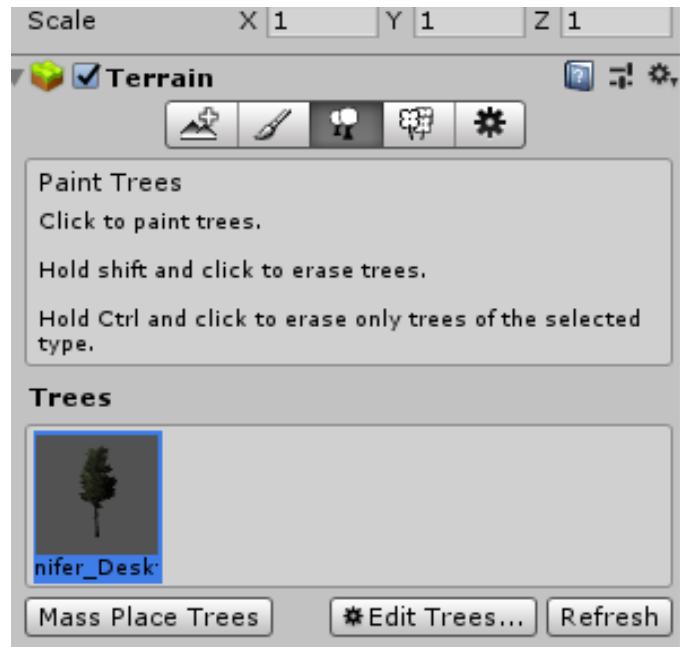


Рисунок 8 — Редактор деревьев.

Новые объекты добавляются через кнопку Edit Trees и в контекстном меню нажимаем Add Tree. Появится окно с парой параметров и напротив Tree Prefab, нажимаем на круг, откроется список с prefabами, для сортировки можно воспользоваться поиском (Рисунок 9).

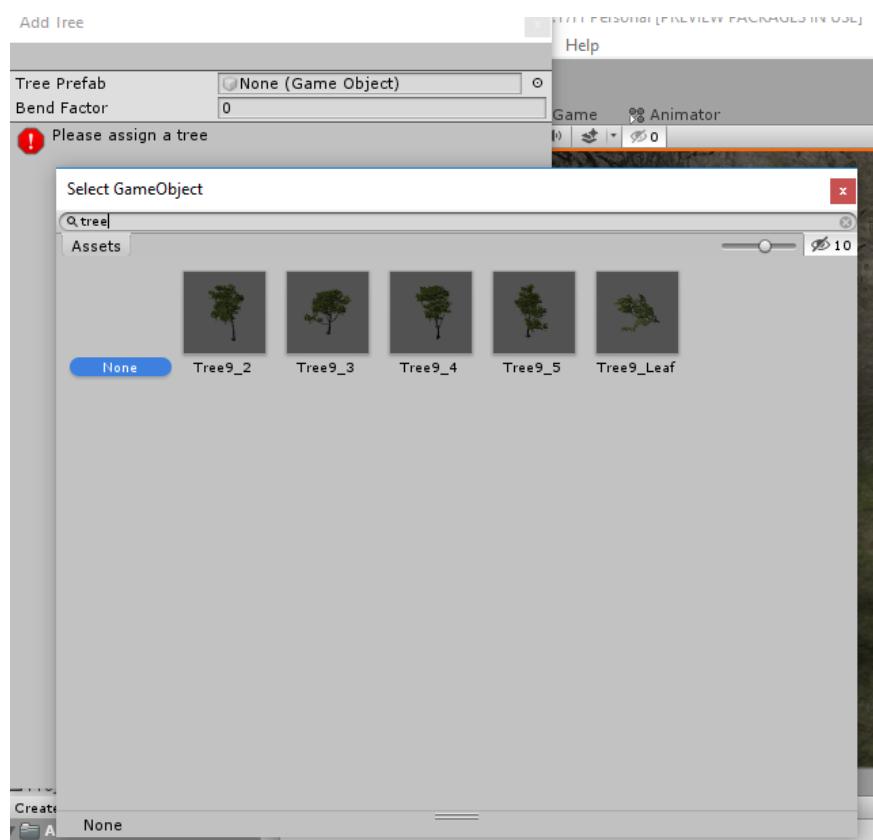


Рисунок 9 — Окно добавления нового prefabа дерева.

После того, как проработка окружения закончена, можно через камеру взглянуть на рендер сцены. Теперь настроим источник света, его цвет, яркость, по желанию, можно изменить скайбокс. Для этого устанавливаем подходящий Asset в списке Window выбираем Rendering – Lighting Settings и в параметре Skybox material заменяем на нужный (Рисунок 10).

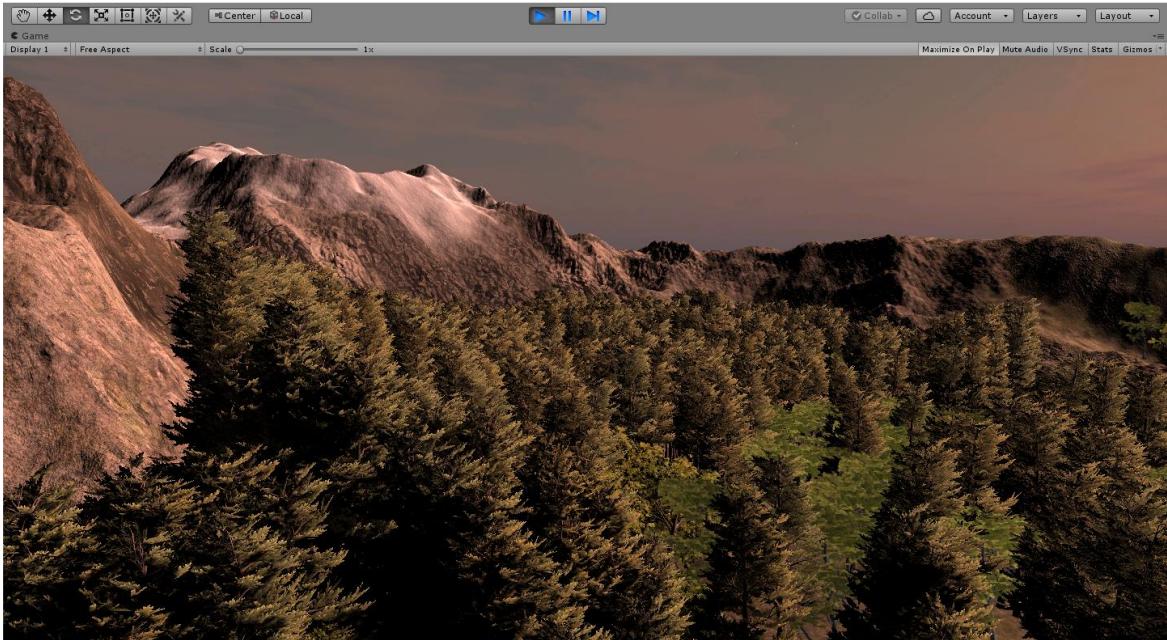


Рисунок 10 — Готовая сцена.

Помимо освещения можно поставить 1st person controller и походить по созданной сцене. Результат работы на практике продемонстрировать преподавателю.

Практическая работа 6

Цель практической работы: изучить методы ввода и вывода данных через скрипт Unity.

Задачи:

- Подготовить сцену;
- Написать 5 скриптов для взаимодействия с клавиатурой и мыши;
- Применить скрипты на объекты;
- Продемонстрировать преподавателю.

Описание выполнения работы

Для начала создадим небольшой плейн, поставим на него пять кубов в ряд, с компонентом Rigidbody и добавим на сцену First person controller (Рисунок 1).

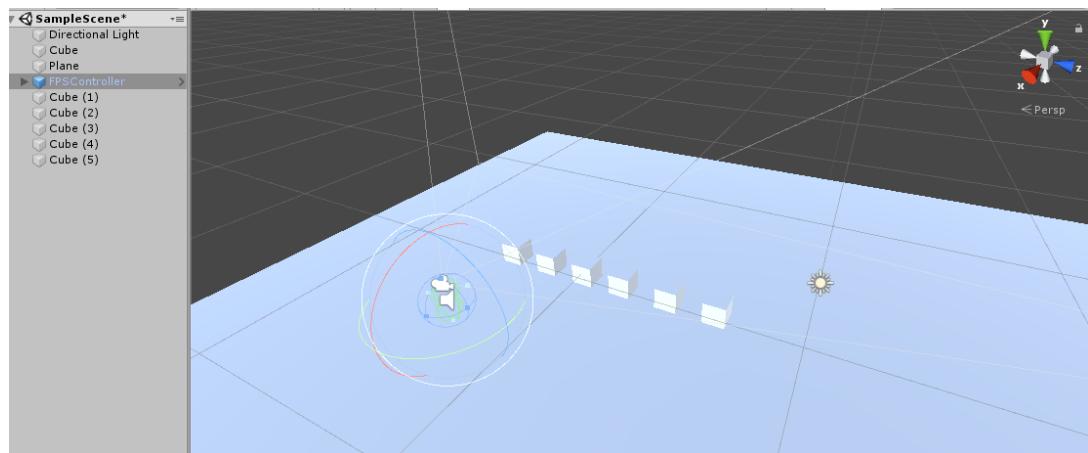


Рисунок 38 — Подготовленная сцена для практического занятия.

Создадим новый Material правой кнопкой мыши в папке, выберем любой понравившийся цвет и зададим этот материал кубам, перетащив его на объекты (Рисунок 2).

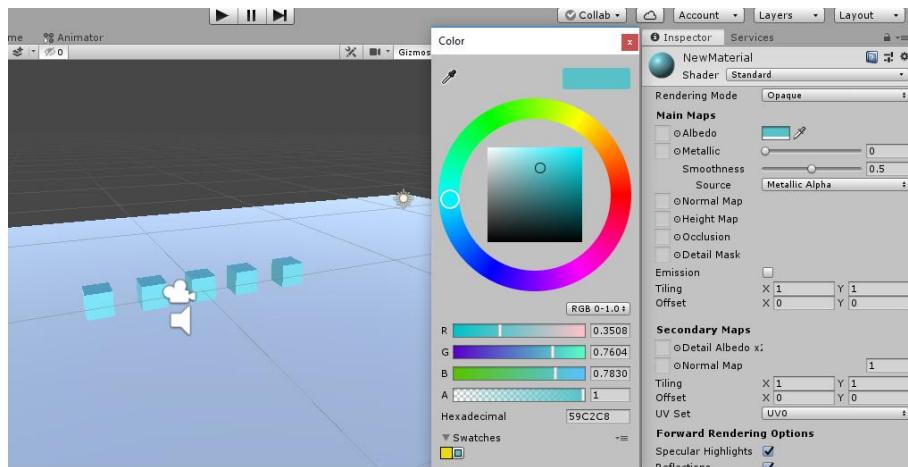


Рисунок 39 — Выбор цвета нового материала через Inspector.

Сцена подготовлена, теперь создаем новый C# скрипт Script, вешаем его на камеру и открываем его.

Задача этого скрипта откинуть объект при нажатии на клавишу R.

Для начала, нужно задать переменную, с которой будет происходить действие – public GameObject SomeGameObject;

В этом скрипте в теле метода Update() теперь напишем условие, согласно которому только при нажатии на клавишу оно выполняется (Рисунок 3).

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.R))
    {
        // do something
    }
}
```

Рисунок 40 — Пример проверки на нажатие клавиши.

Теперь, обращаемся в объекту, который будет храниться в новой переменной, нужно получиться доступ к его компоненту, к Rigidbody, в Unity это делаемся с помощью команды GetComponent<"Название компонента">(). Таким образом, можно обращаться не только к стандартным компонентам Unity, но и скриптом, которые находятся на объектах.

После того, как доступ получен, применяем метод AddForce и в нем задаем Vector3, другими словами, направление по трем осям координат, в которое полетит объект (Рисунок 4). Не стоит бояться экспериментировать со значениями в скобках.

```
public class Script : MonoBehaviour
{
    public GameObject SomeGameObject;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            SomeGameObject.GetComponent<Rigidbody>().AddForce(0,200,0);
        }
    }
}
```

Рисунок 41 — Пример применения метода AddForce к стороннему объекту.

Возвращаемся в Unity и перемещаем ссылку на объект через Inspector. При запуске сцены, при нажатии на клавишу R указанный объект будет подлетать по координате Y.

Теперь перейдем к следующему кубу. Создаем новый C# скрипт Scrip2 и перетаскиваем его на второй объект. Этим скриптом будем менять цвет второму кубу, когда персонаж зайдет в Collider и пользователь нажмет на клавишу Т. Для решения такой задачи, понадобится метод OnTriggerStay(Collider), а в нем напишем проверку на нажатие клавиши, как в Script и в нем через компонент Renderer поменяем цвет материала (Рисунок 5).

```
}
private void OnTriggerEnter(Collider other)
{
    if (Input.GetKeyDown(KeyCode.T))
    {
        GetComponent<Renderer>().material.color = Color.red;
    }
}
```

Рисунок 42 — Пример исполнения метода OnTriggerEnter, с изменением

Перед запуском сцены нужно добавить новый BoxCollider на объект, сделать его немного больше через кнопку Edit Collider и сделать его Is Trigger (Рисунок 6).

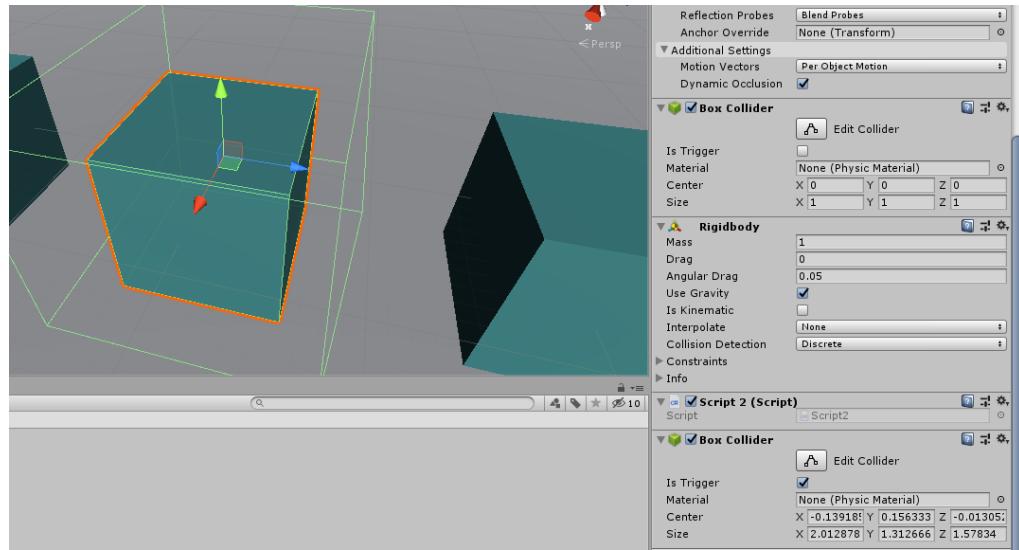


Рисунок 43 — Границы второго Box Collider’а и его настройка.

Теперь при запуске сцены, когда персонаж войдет в этот коллайдер, и будет нажата клавиша Т в этот момент, то куб окрасится в красный (Рисунок 7).

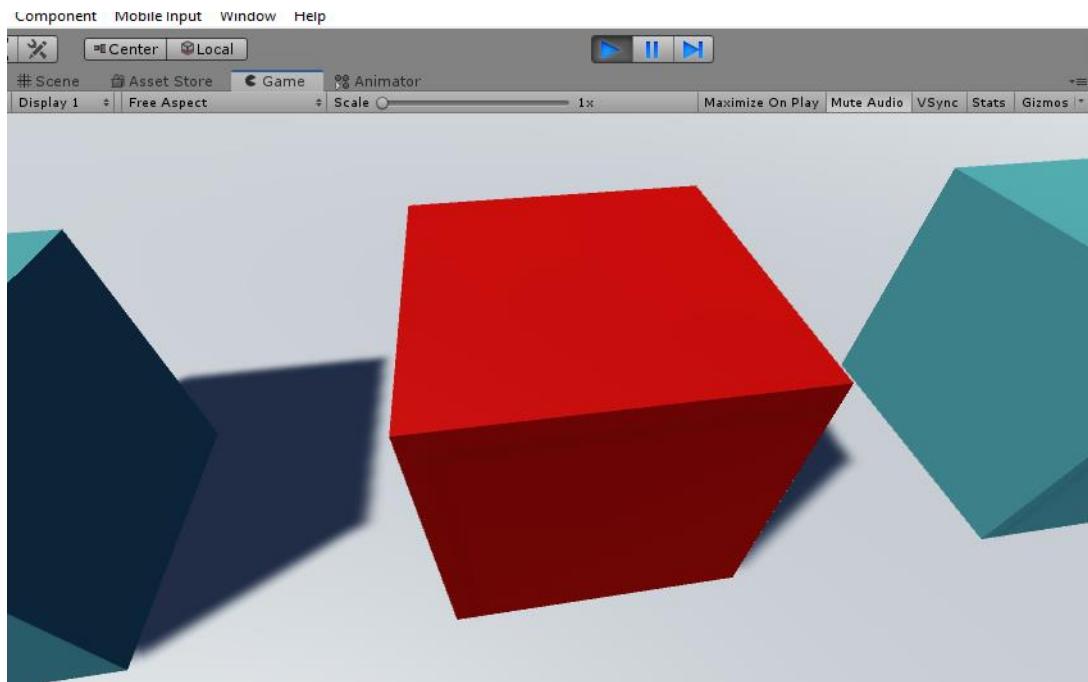
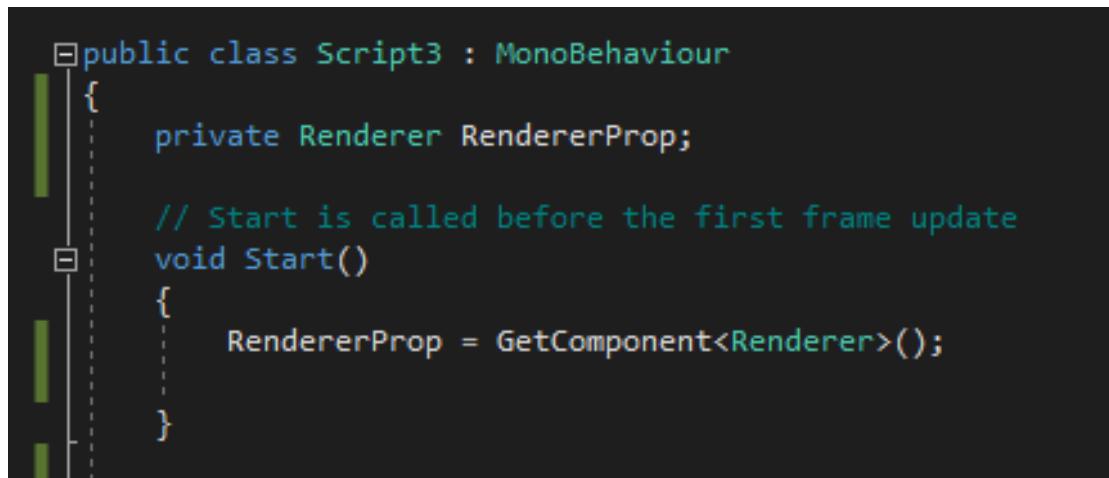


Рисунок 44 — Пример исполнения взаимодействия во время активной сцены.

Второй куб настроен, переходим к следующему. Создаем еще один C# скрипт Script3. Этот скрипт нужно положить на третий куб. С помощью этого

скрипта куб будет менять свой цвет на зеленый, когда на него наводится курсор мыши, а когда курсор выходит за границы объекта на экране, то он сменит цвет на синий.

В скрипте сразу объявим переменную типа Renderer, и присвоим ей в методе Start() соответствующий компонент, это имеет смысл, т.к. не нужно будет постоянно писать одно и тоже, когда обращение к переменной будет повторяться несколько раз, не так как было со вторым скриптом (рисунок 8).

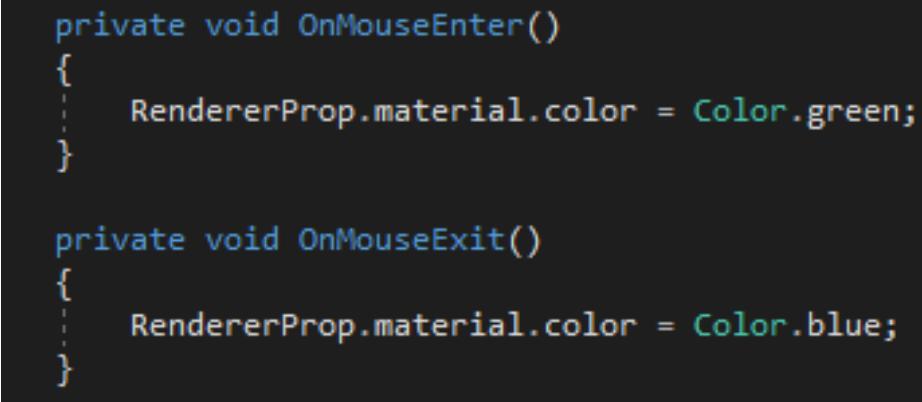


```
public class Script3 : MonoBehaviour
{
    private Renderer RendererProp;

    // Start is called before the first frame update
    void Start()
    {
        RendererProp = GetComponent<Renderer>();
    }
}
```

Рисунок 45 — Объявление и присвоение значения переменной RendererProp.

Теперь методам OnMouseEnter() и OnMouseExit() пропишем их функционал, согласно поставленной задаче (Рисунок 9).



```
private void OnMouseEnter()
{
    RendererProp.material.color = Color.green;
}

private void OnMouseExit()
{
    RendererProp.material.color = Color.blue;
}
```

Рисунок 46 — Изменение цвета материала на объекте с помощью положения курсора.

Теперь при запуске сцены, третий куб будет менять свой цвет в зависимости от того, наведен на него курсор или нет.

Четвертый куб будет выводить в консоль некоторые свои характеристики при нажатой клавише Y, а именно: имя объекта, его координаты, цвет его материала и масса в компоненте RigidBody.

Напишем метод OnMouseDown() с парой Debug'ов, которые будут получать свойства объекта, на котором скрипт. Этот метод будет вызываться каждый раз, когда пользователь будет кликать правой клавишей мыши на объект (Рисунок 10).

```
public class Script4 : MonoBehaviour
{
    private void OnMouseDown()
    {
        Debug.Log(name);
        Debug.Log(transform.position);
        Debug.Log(GetComponent<Renderer>().material.color);
        Debug.Log(GetComponent<Rigidbody>().mass);
    }
}
```

Рисунок 47 — Вывод данных через Debug.

Теперь запустим сцену, нажмем на клавишу Y и проверим консоль, если в консоль выводится примерно тоже, что и на рисунке 11, то все сделано верно.

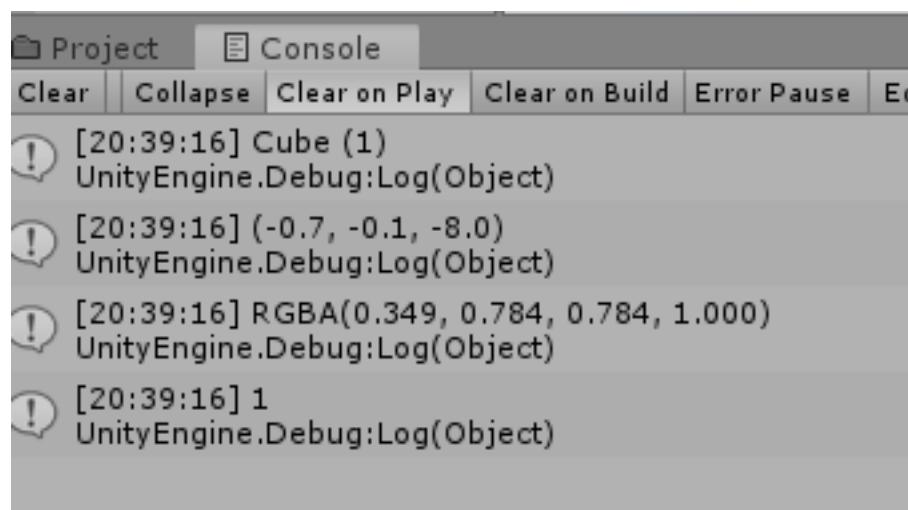


Рисунок 48 — Выведенные данные в консоль при запущенной сцене.

Для пятого куба нужен скрипт, с которым объект можно будет брать с помощью зажатия клавиши мыши. Для этого объявим две переменные MouseOffset и MouseZCoord. Еще понадобится два встроенных метода OnMouseDown() и OnMouseDrag(). В теле последнего метода и будет основная логика скрипта: transform.position = Get.mousePosition() + MouseOffset;

Для того, чтобы эта логика работала, нужно получать координаты курсора, собственно, нужно написать такой метод, который их и будет получать Get.mousePosition(). Этот метод будет не void, а Vector3, все потому что необходимо вернуть значения курсора по x, у и z, что можно сделать с помощью типа Vector3 (Рисунок 12).

```
public class Script5 : MonoBehaviour
{
    private Vector3 MouseOffset;
    private float MouseZCoord;

    //При нажатии клавиши мыши
    //происходит корректировка позиции
    void OnMouseDown()
    {
        MouseZCoord = Camera.main.WorldToScreenPoint(gameObject.transform.position).z;

        MouseOffset = gameObject.transform.position - Get.mousePosition();
    }

    //Получение координат курсора
    private Vector3 Get.mousePosition()
    {
        Vector3 mousePoint = Input.mousePosition;

        mousePoint.z = MouseZCoord;

        return Camera.main.ScreenToWorldPoint(mousePoint);
    }

    //Основная логика скрипта
    //смена позиции объекта относительно курсора
    void OnMouseDrag()
    {
        transform.position = Get.mousePosition() + MouseOffset;
    }
}
```

Рисунок 12 —Скрипт для передвижения куба вместе с курсором.

Теперь возвращаемся в Unity и запускаем сцену. Если все сделано правильно, то, когда пользователь зажимает левую клавишу мыши на пятом кубе, куб будет двигаться вместе с курсором.

Показать все взаимодействия с объектами преподавателю.

Практическая работа 7

Цель практической работы: изучить разные способы придания объекту на сцене движения через скрипт C#.

Задачи:

- Подготовить сцену;
- Написать скрипт движения к первому объекту с помощью векторного способа;
- Написать скрипт движения ко второму объекту с помощью его Rigidbody;
- Создать генератор объектов при помощи prefabов
- Продемонстрировать преподавателю.

Описание выполнения работы

Перед началом выполнения практической работы необходимо установить стандартный пак с ассетами. Из него для этого занятия потребуется First person controller.

Теперь подготовим сцену, для занятия понадобится: плейн, два шара с компонентом Rigidbody и две кубы напротив шаров с этим же компонентом (Рисунок 1).

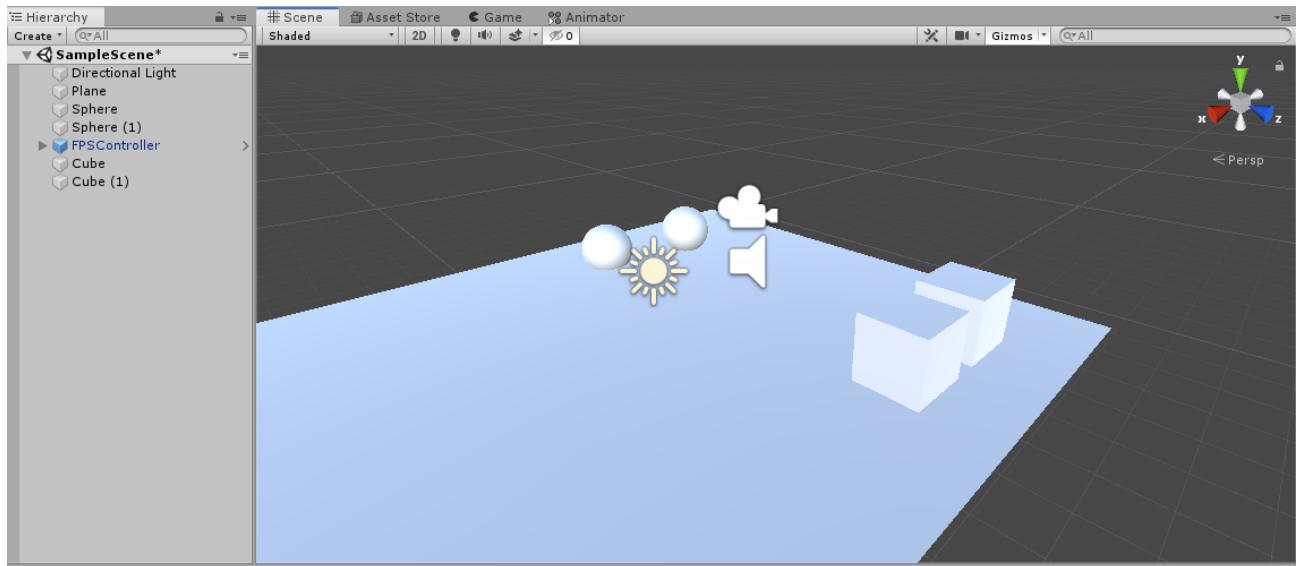


Рисунок 49 — Пример сцены для практического занятия.

В Unity движение можно реализовать далеко не одним способом, на этом практическом будет затронуто два таких, самых распространенных.

Первый способ реализуется с помощью вектора и предполагает постепенную смену значений в параметре position. Это самый примитивный способ воздействия на движение объекта через скрипт. Минус этого способа в том, что он не будет подвержен физике сцены, иногда такой метод не подходит для решения некоторых задач. Для его реализации можно использовать метод Translate(arg), в его скобках необходимо будет указать аргумент, в данном случае формулу, по которой будет работать движение этого объекта, например, transform.forward + Speed + Time.deltaTime. В замену вектору transform.forward можно использовать любой другой вектор (Рисунок 2).

```
public class Script1 : MonoBehaviour
{
    public float Speed;

    void Update()
    {
        transform.Translate(transform.forward * Time.deltaTime * Speed);
    }
}
```

Рисунок 50 — Векторный способ реализации движения.

Второй способ реализуется через компонент RigidBody объекта. У этого компонента есть такой параметр как velocity (Скорость), по умолчанию, когда компонент помещается на объект, он равен нулю, что значит, что он будет покоиться на одном месте. С помощью скрипта можно изменить этот параметр, обратившись к нему через твердое тело: rigidbody.velocity, и присвоить ему формулу на подобии: Speed + Time.deltaTime (Рисунок 3).

```
public class Script2 : MonoBehaviour
{
    public Vector3 Speed;

    void Update()
    {
        GetComponent<Rigidbody>().velocity = Speed * Time.deltaTime;
    }
}
```

Рисунок 51 — Реализация движения через Rigidbody объекта.

Необходимо создать два разных скрипта, один с помощью первого способа (Рисунок 2), другой с помощью второго (Рисунок 3), прикрепить их к разным сферам, задать значения в инспекторе.

Когда все описанные выше задачи выполнены, можно запустить плеер и посмотреть на результат. Если сферы идут в разных направлениях, не на кубы, или вообще не движутся, то необходимо провести корректировку значений в инспекторе или проверить код на наличие ошибок. Из этого эксперимента наглядно видна разница между этими двумя способами реализации движения объектов.

Следующей задачей является создать генератор, который будет создавать новые движущиеся сферы при нажатии на клавишу R. Этого можно достичь, прибегнув к созданию prefabа объекта.

Создадим новую сферу и сделаем ее prefabом. Для этого необходимо перенести объект со сцены в любую файловую директорию, например, в папку ассетс, такой файл будет отображаться голубым кубом (Рисунок 4).

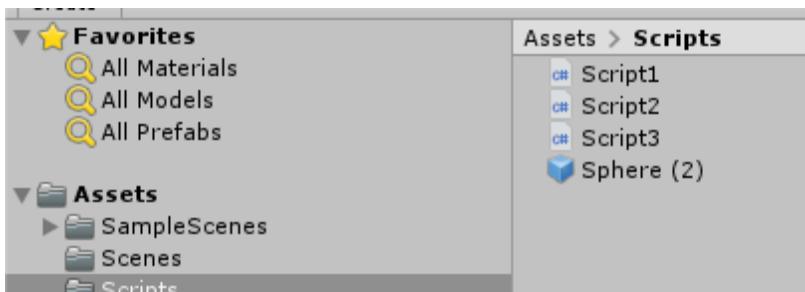


Рисунок 52 — Пример отображения prefabа в папке.

Теперь этот объект сохранен как копия вне одной сцены. Если изменить prefab, то изменения можно будет применить ко всем объектам, созданным с помощью этого prefabа. Таким образом удобно управлять

различными объектами, которые используются многократно. Чтобы не переписывать много кратно один и тот же код, prefab добавим компонент Script2 и назначим значения паблик вектору в инспекторе, чтобы открыть prefab, необходимо кликнуть на него в директории два раза, и объект появится в специальной сцене (Рисунок 5).

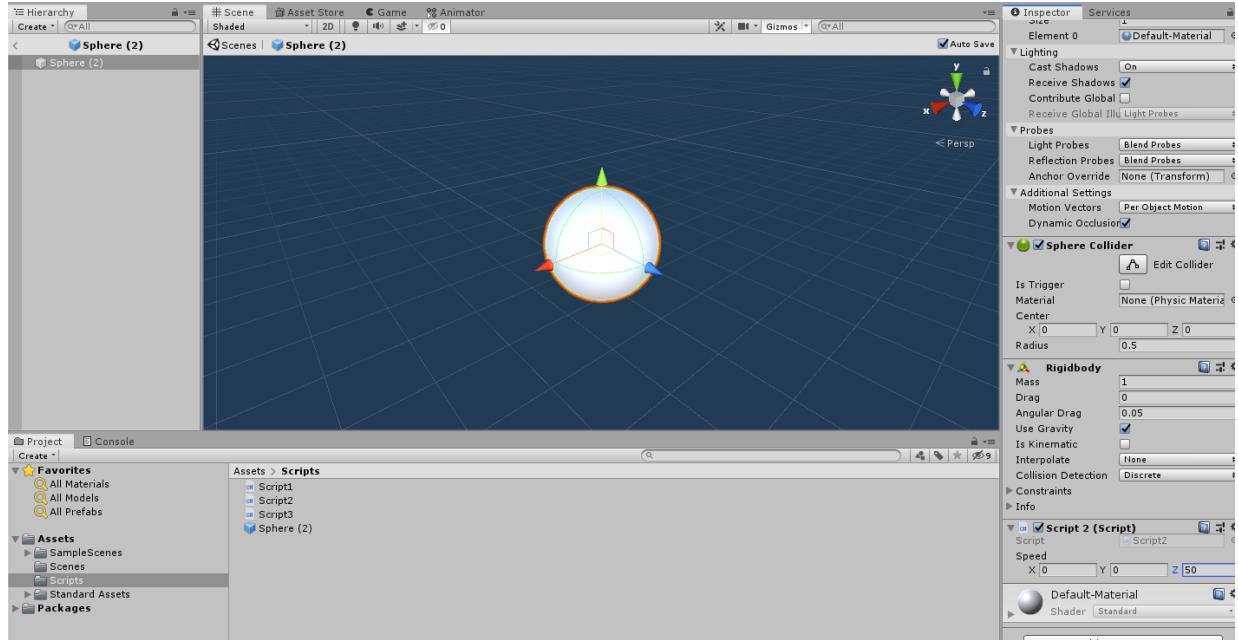


Рисунок 53 — Открытый prefab.

Теперь вернемся на основную сцену, нажав на кнопку Scenes, в правом углу от отображаемого prefaba, или на стрелку назад в иерархии объектов. Prefab готов, уже можно удалить его копию со сцены. Создаем новый объект, который и будет служить генератором, им может служить любой объект, в данном занятии им станет Пустой Объект, Empty GameObject (Рисунок 6). Из его названия понятно, что это объект без дополнительных компонентов, только со свойствами, которые имеются у всех объектов в Unity.

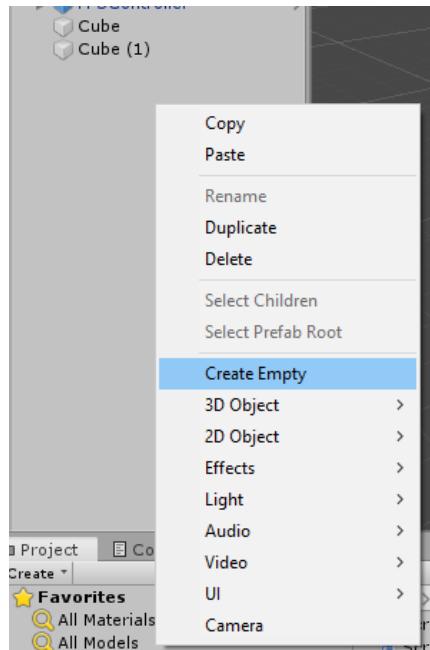


Рисунок 54 — Создание пустого объекта.

Теперь создадим новый C# скрипт, добавим его на пустой объект и напишем код, который будет создавать копии движущихся сфер. Для этого понадобится public GameObject переменная, чтобы хранить prefab и метод Instantiate, первым аргументом которого будет prefab, вторым координаты, в которых будет появляться копия и третьем параметр поворота, с которым будет появляться объект (Рисунок 7).

```
public class Script3 : MonoBehaviour
{
    public GameObject Prefab;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R))
            Instantiate(Prefab, transform.position, transform.rotation);
    }
}
```

Рисунок 55 — Реализация генератора объектов.

Осталось задать в инспекторе создаваемый prefab и проверить местоположение генератора и поставить его так, чтобы можно было наблюдать создание сфер через 1st person controller (Рисунок 8).

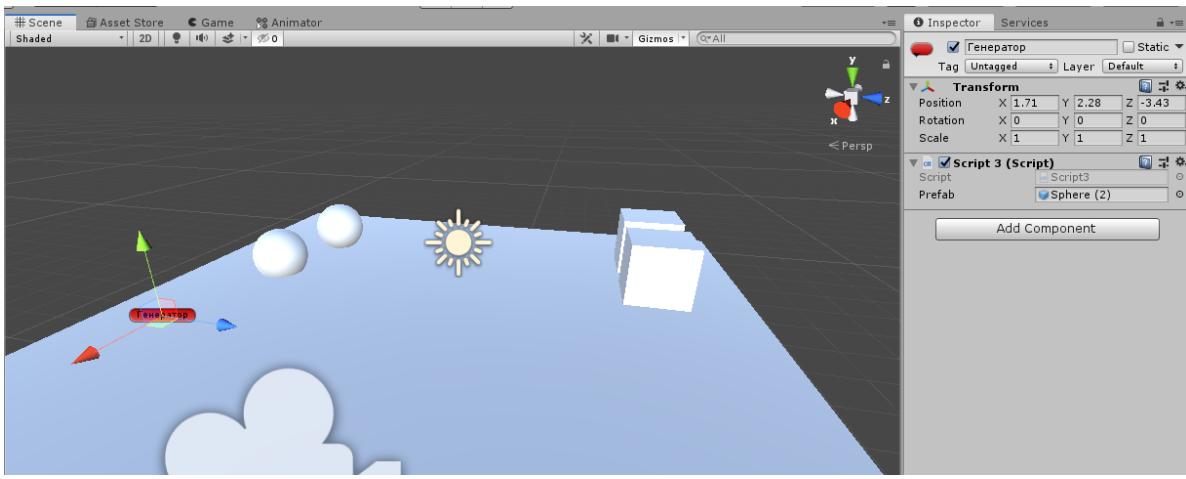


Рисунок 56 — Пример расстановки объектов на сцене.

При запущенной сцене, каждый раз, когда будет прощата клавиша R, будет создаваться клон prefaba, который будет двигаться в указанном направлении (Рисунок 9).

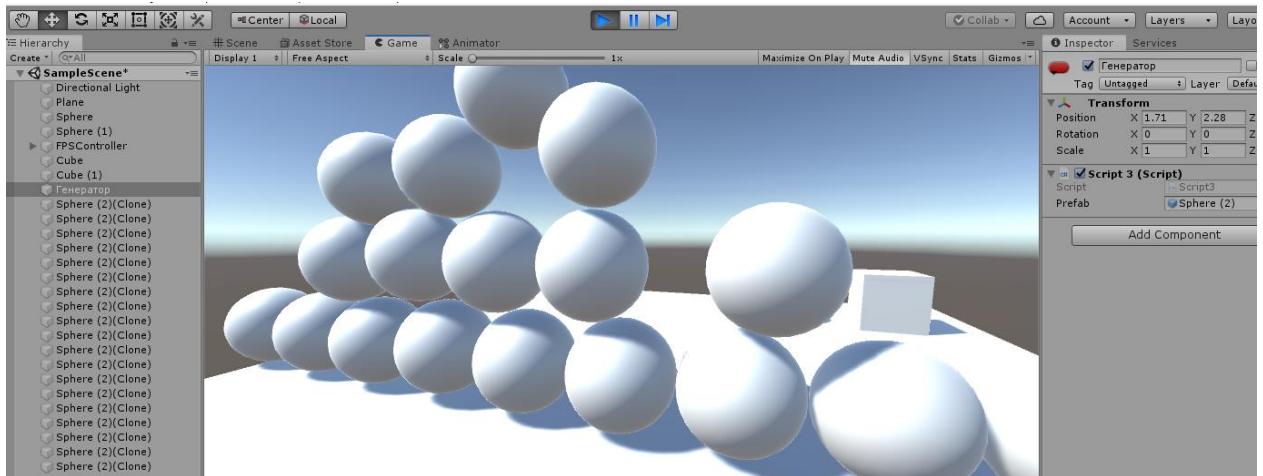


Рисунок 57 — Пример финальной сцены.

Результат работы продемонстрировать преподавателю.

Практическая работа 8

Цель практической работы: изучить свойства и принципы работы с анимацией и аниматором.

Задачи:

- Подготовить сцену;
- Записать клип движения сферы;
- Записать клип взлета сферы;
- Написать скрипт переключения анимации через Trigger;
- Продемонстрировать преподавателю.

Описание выполнения работы

Перед началом практической работы нужно подготовить сцену к этому занятию. Для этой работы достаточно будет создать plane и на него поместить сферу. Еще для этого занятия понадобится такое окно как Animation, его можно поставить открыть с помощью вкладки Window и закрепить окно в любое удобное место (Рисунок 1).

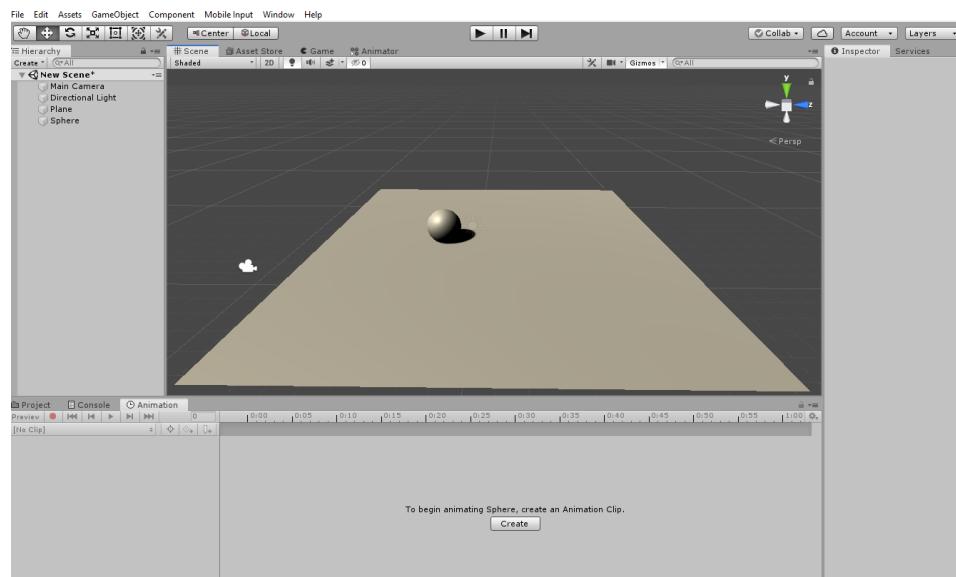


Рисунок 58 — Готовая сцена к работе.

Теперь для объекта сферы необходимо добавить компонент animator. Теперь можно создать анимационный клип, для этого нужно нажать кнопку Create в окне animation (Рисунок 2). Сохраним файл клипа в папке и перейдем к записи клипа.

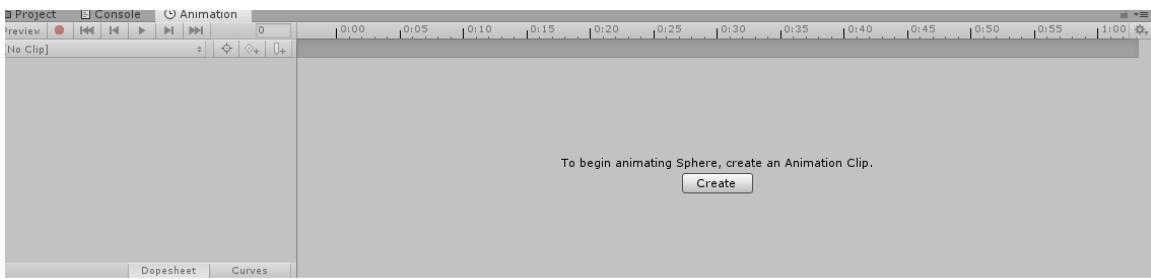


Рисунок 59 — Окно Animation.

Для того, чтобы начать запись, нужно нажать на красную кнопку, тогда будет доступна запись по ключевому кадру. Например, есть перемещать объект, менять ему координаты, и все время переключать временные отрезки всего клипа. Следует колесиком мышки увеличить интервал клипа примерно до 10-20 секунд, изначально там отображаются мили секунды.

Когда режим записи будет активирован, временной отрезок будет подсвечен красным (Рисунок 3).

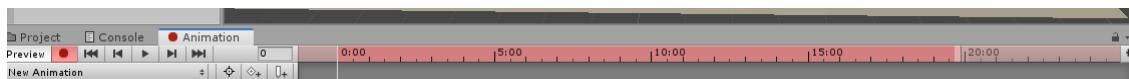


Рисунок 60 — Начало записи анимационного клипа.

Необходимо сделать клип, в котором сфера пройдет по всем углам платформы и вернется обратно на ту же позицию.

После начала записи, передвигаем на первый временной диапазон, например, на 5 секунд и на этой точке передвинем саму сферу в другой угол, подставится специальный маркер на это время (Рисунок 4).

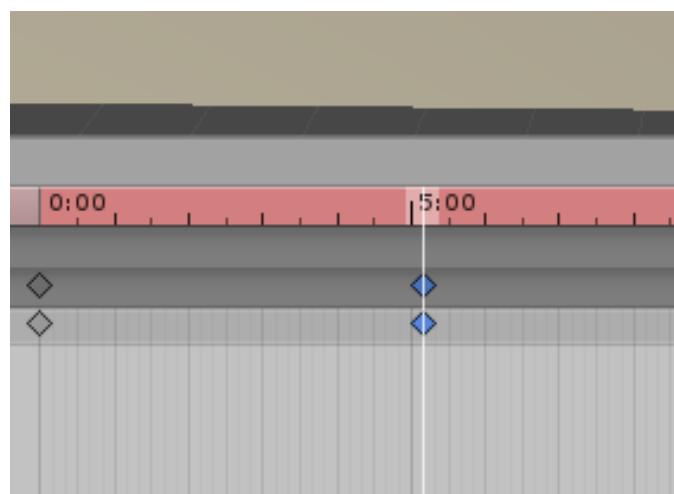


Рисунок 61 — Анимационный ключ.

Аналогичные действия проделать с остальными углами, а для возвращения в изначальную позицию можно просто кликнуть на самый первый ключ (0:00 по времени) и копировать его в конец сочетанием клавиш CTRL+C/CTRL+V. Помимо копирования их можно еще двигать на всем временном диапазоне (Рисунок 5).

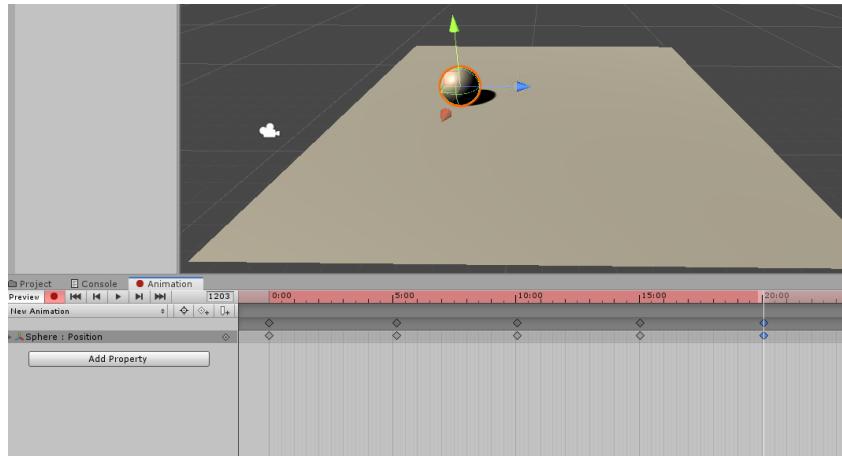


Рисунок 62 — Все ключи на всем временном диапазоне клипа.

Теперь можно закончить запись, для проверки можно промотать клип на начало и запустить.

После того, как клип был сделан, можно создать следующий, его можно создать из обычной панели создания через правую кнопку мыши в директории и дадим ему наименование “Jump”. В этом клипе объект будет подниматься по координате Y.

Перед тем, как открыть его, нужно перейти во вкладку Animator (она расположена рядом со вкладкой Game по умолчанию). После клика на сферу нужно переставить текущий анимационный клип (Рисунок 6).

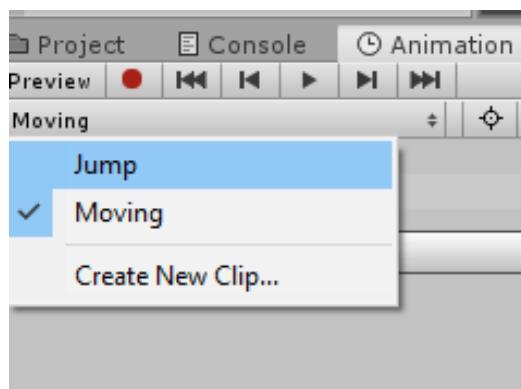


Рисунок 63 — Переключение активных клипов.

Откроем новый клип в Animation и начнем запись. Таким же образом создаем ключи для высоты на десятой секунде и на двадцатой копируем самый первый ключ.

После того, как два клипа сделано, можно приступить к созданию переходов между ними, для этого нужно открыть окно аниматора, оно вызывается через вкладку window. Окно Animator по умолчанию открыто и находится у окна Game (Рисунок 7).

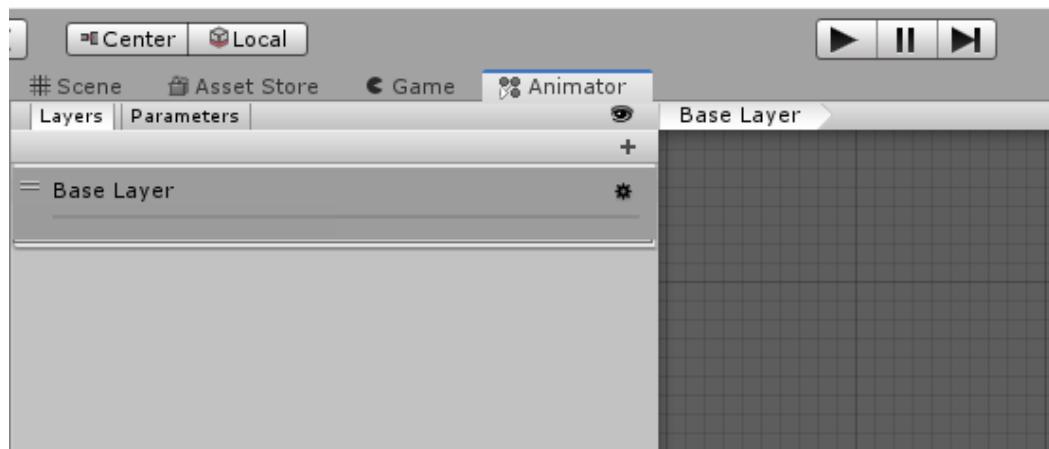


Рисунок 64 — Окно Animator.

Теперь необходимо перетащить в окно файлы созданных ранее клипов и расположить их согласно логике на Рисунке 8.

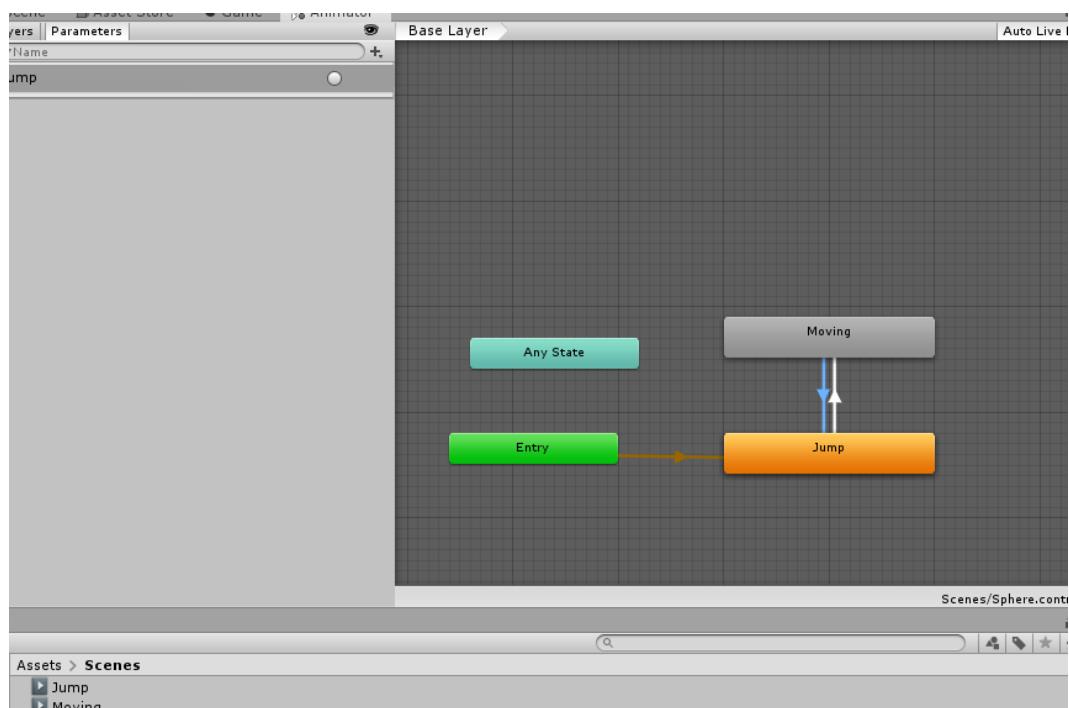


Рисунок 65 — Логика построения переходов между клипами.

Состояние Entry – первичное, оно указывает, какой клип будет проигрываться первым и будет считаться начальным.

Any State – состояние, которое может вызываться в любой момент, например, когда персонаж идет и ему нужно прыгнуть посреди анимации шага, клип шага прерывается, и проигрывается анимация прыжка.

Стоит обратить внимание на стрелки, это переходы (Transition) между клипами. Для их создания нужно нажать на клип правой кнопкой мыши и выбрать в меню Make transition.

Теперь необходимо настроить переходы. Для этого нажимаем на созданные стрелки и меняем настройки (Settings) через Inspector (Рисунок 9).

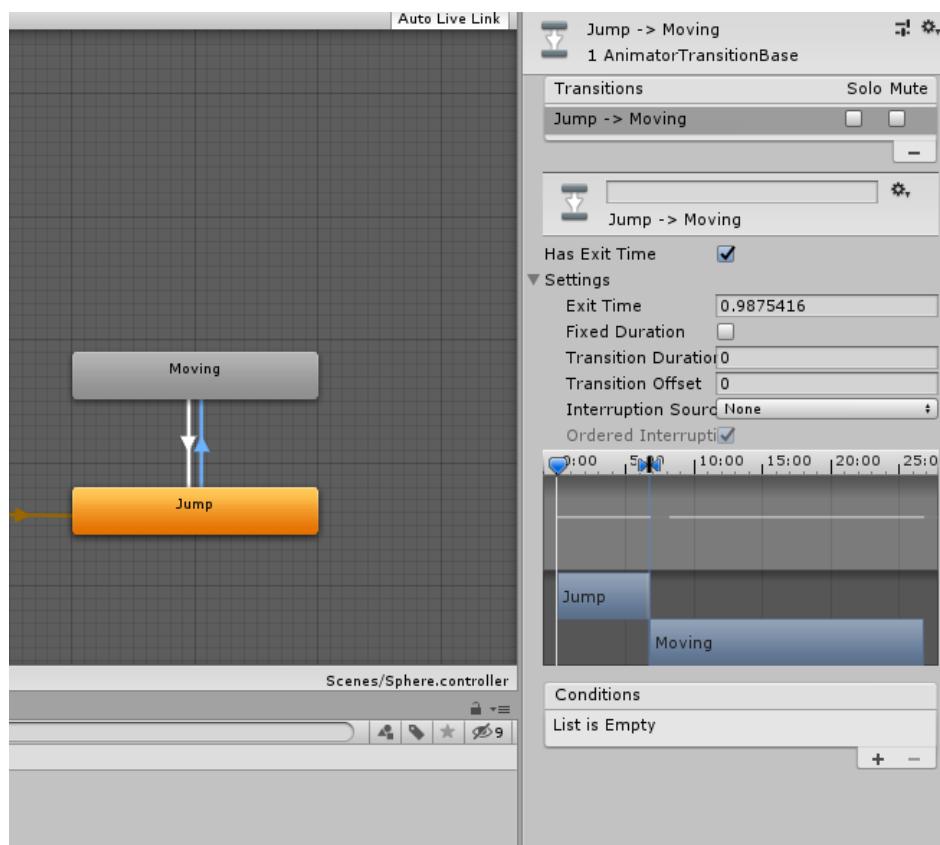


Рисунок 66 — Параметры перехода.

После всей проделанной работы можно проверить правильность работы последовательности анимации. При запуске сцены, сфера сначала подпрыгивает, а потом начинает кататься по поверхности, возвращается на изначальную точку и проделывает тот же самый цикл.

Если на этом этапе все выполнено правильно, то можно усложнить задачу и добавить к этой логике вмешательство с помощью скрипта. Создаем новый C# скрипт AnimationScript. В нем напишем код, который будет менять параметр JumpTrigger при нажатии клавиши E на клавиатуре (Рисунок 10).

```
public class AnimationScript : MonoBehaviour
{
    Animator anim;

    void Start()
    {
        anim = GetComponent<Animator>();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            anim.SetTrigger("JumpTrigger");
        }
    }
}
```

Рисунок 67 — Код скрипта управления анимации.

Сохраняем скрипт и добавляем его на сферу, ту самую, где и находится компонент Animator.

Теперь осталось создать этот самый параметр JumpTrigger. Для этого в окне аниматора заходим во вкладку Parameters, нажимаем на плюс рядом с поиском по названию, выбираем тип параметра Trigger, и даем ему наименование JumpTrigger (Рисунок 11).

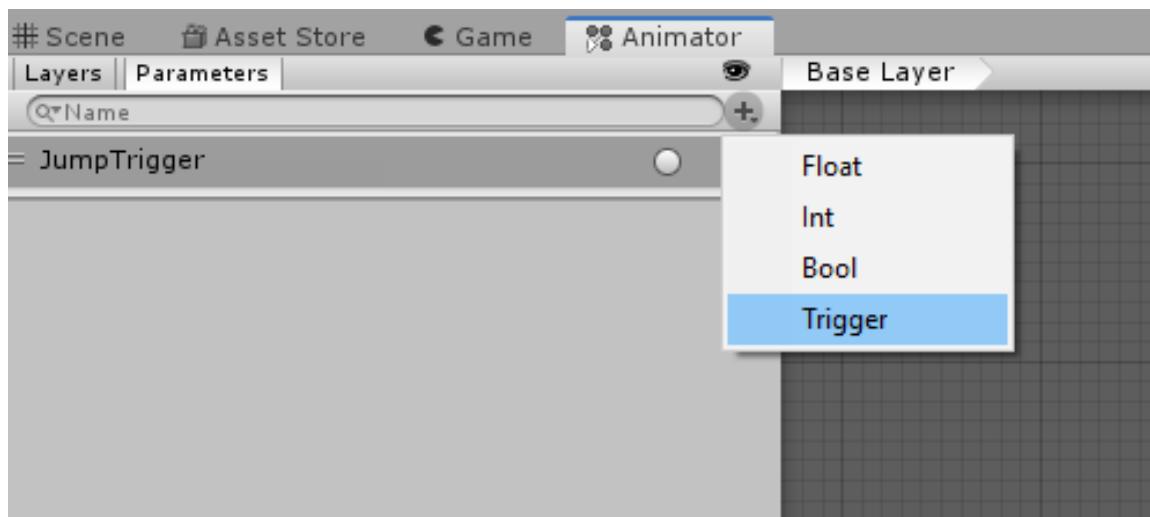


Рисунок 68 — Добавление параметра Trigger.

Для того, чтобы триггер сработал, его нужно установить на какой-нибудь из переходов. В данном случае можно поставить триггер на переход из Jump в Moving. Для этого нужно нажать на значок плюса в разделе Condition (Рисунок 12).

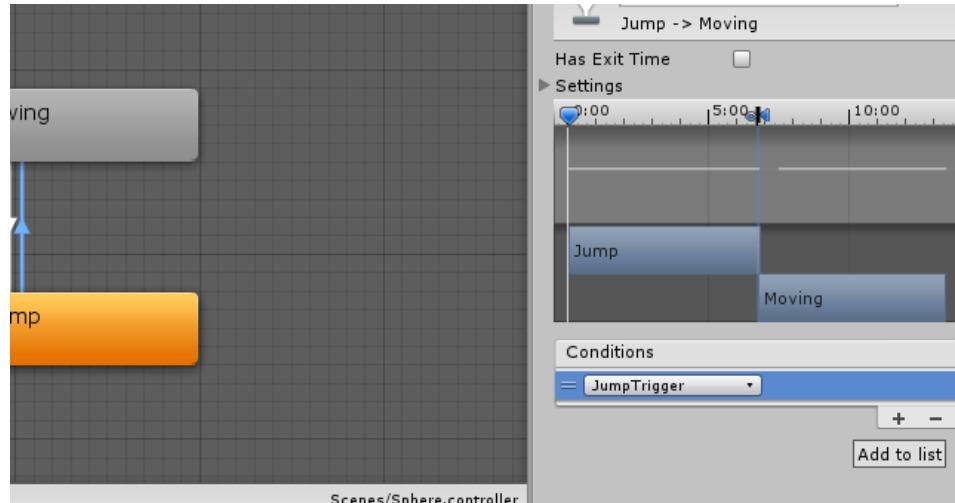


Рисунок 69 — Установка триггера на один из переходов.

Стоит отметить, что в этом переходе нужно убрать галочку с параметра Has Exit Time. Помимо этого, для быстроты проигрывания анимации можно ускорить клипы. Для этого нужно нажать на клип в окне аниматора и в инспекторе поменять параметр Speed, например, на значение 3 (Рисунок 13).

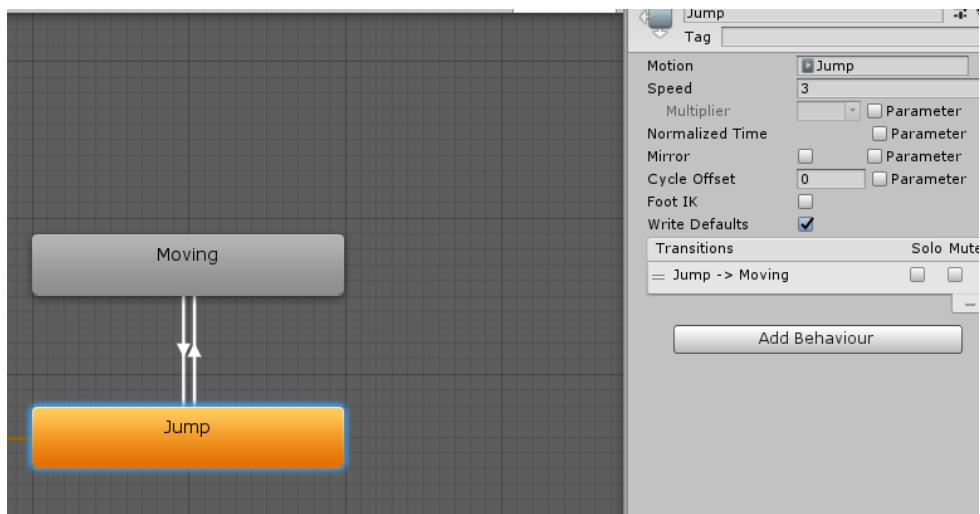


Рисунок 70 — Параметры анимационного клипа.

Запускаем сцену и проверяем правильность сделанной работы.
Показываем преподавателю.

Практическая работа 9

Цель практической работы: изучить свойства и принципы работы с аудиофайлами и их проигрыванием в Unity.

Задачи:

- Подготовить сцену;
- Добавить проигрывание фоновой музыки;
- Добавить звуковой эффект падения на твердую поверхность;
- Продемонстрировать преподавателю.

Описание выполнения работы

Перед началом практической работы нужно подготовить сцену к этому занятию. Понадобится установить из Asset Store Standart Assets, и еще понадобится фоновая музыка. В Asset Store нужно будет найти background music и импортировать в проект понравившийся бесплатный пак. На сцене установить Plane и создать куб и пару сфер (Рисунок 1). Добавим на все объекты компонент RigidBody. Для последней сферы добавим новый Physics Material, с выставленным значением Bounciness на максимум.

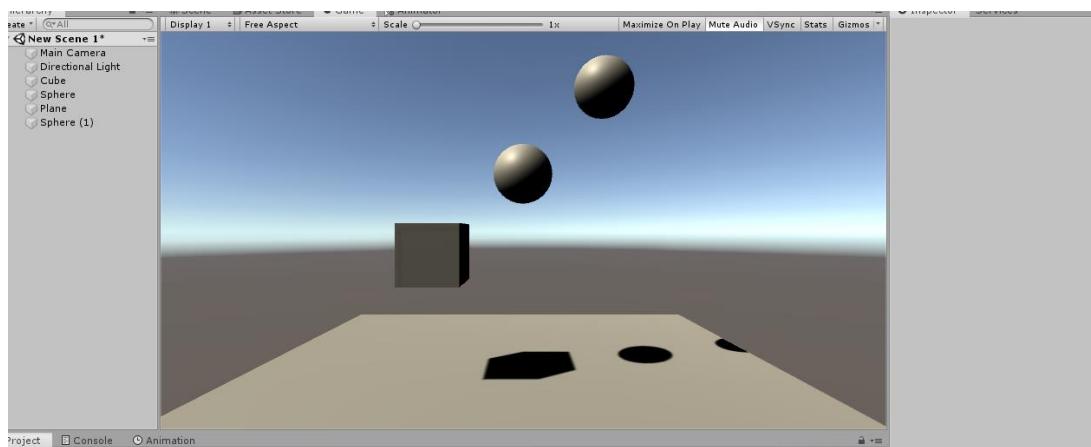


Рисунок 71 — Пример подготовленной сцены.

Сцена подготовлена, теперь можно приступить к созданию AudioSource объекта. AudioSource отвечает за проигрывание аудиофайлов, и все настройки делаются через него. В отличии от аниматора, источник аудио не является компонентом, а создается как полноценный объект на сцене. Через меню создания в иерархии добавляем новый объект (Рисунок 2).

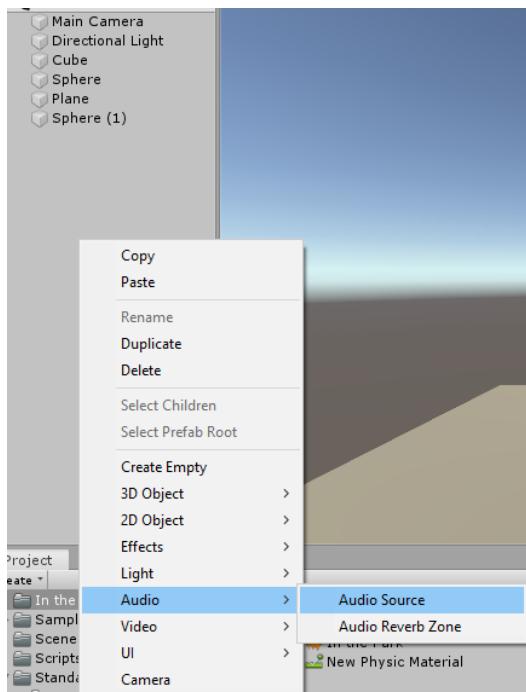


Рисунок 72 — Создание объекта AudioSource.

Этот источник звука наименуем как BackGroundMusic, она будет проигрываться фоном на протяжении всего сеанса. Следующим шагом добавим Audio Mixer через директорию Assets (Рисунок 3). Это специальный файл, позволяющий удобно группировать аудио и выставлять интересные иерархии проигрывания, что очень пригодится в больших проектах, в этой же работе, Audio Mixer будет рассматриваться как пример.

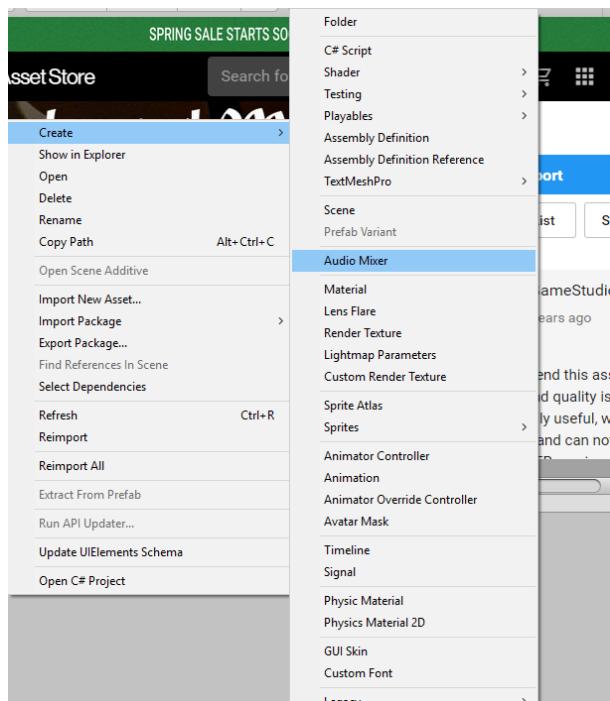


Рисунок 73 — Создание файла Audio Mixer.

Откроем Audio Mixer и с помощью иконки плюсика рядом с разделом Groups добавим две группы звуков (Рисунок 4).

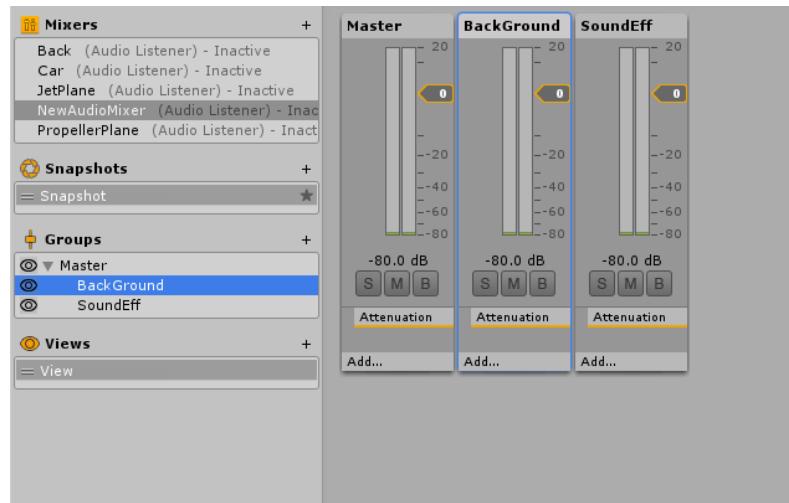


Рисунок 74 — Окно панели Audio Mixer'a.

Теперь вернемся к сцене и перейдем на объект BackGroundMusic и откроем его параметры в Inspector'e (Рисунок 5)

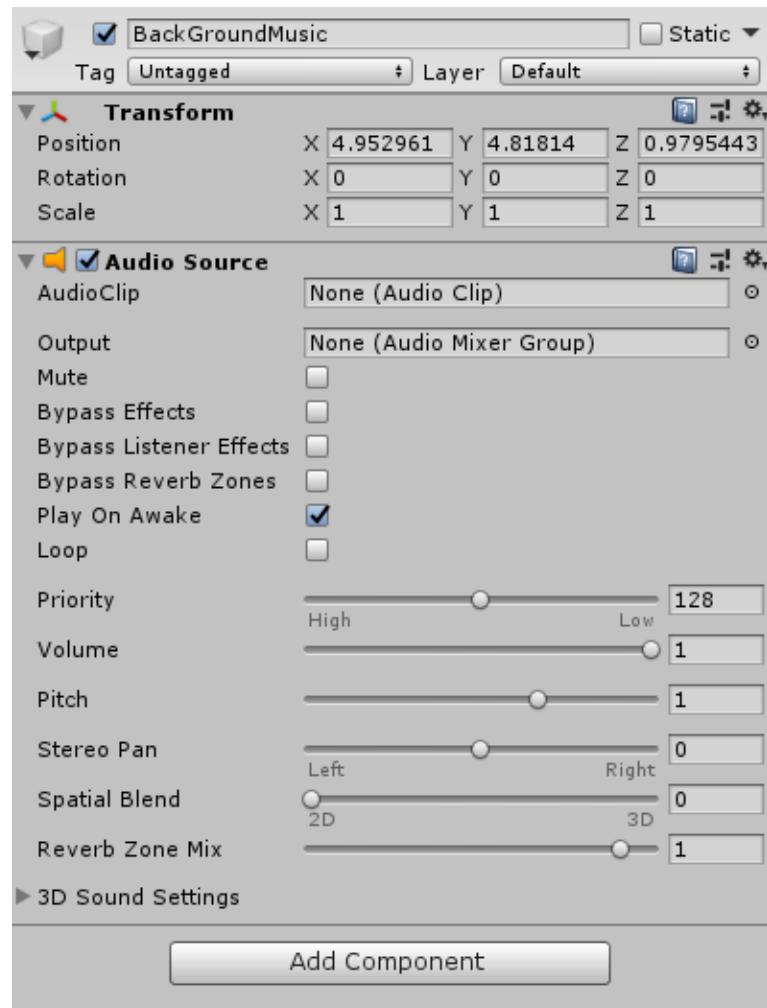


Рисунок 75 — Окно Inspector'a объекта Audio Source.

В переменной AudioClip будет храниться ссылка на тот файл, который этот источник звука будет проигрывать, через кружок справа выберем нужный файл для фоновой музыки (Рисунок 6).

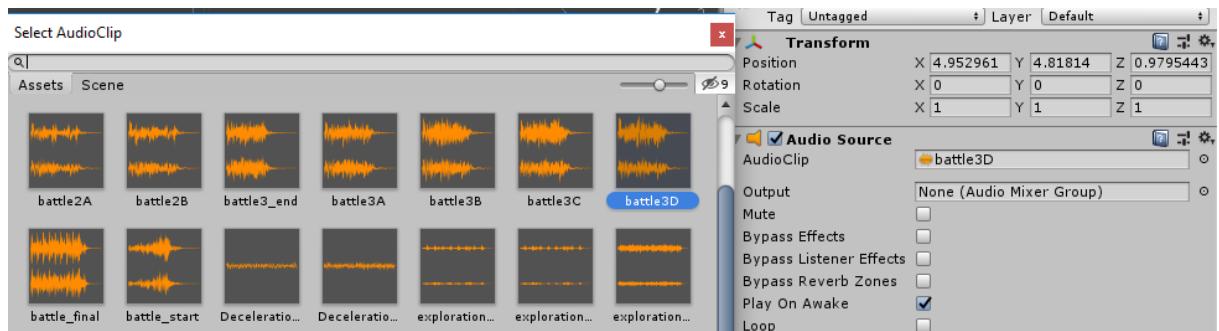


Рисунок 76 — Окно добавления нового AudioClip'a.

Укажем Output группу BackGround и поставим галочку на параметре Loop, для того, чтобы аудиоклип повторялся. Перед запуском сцены для проверки рекомендуется убавить звук в микшере Windows и сделать звук тише через тот же Mixer или параметр Volume в инспекторе источника звука.

Запустим сцену и послушаем результат. Музыка все время будет проигрываться, если все сделано правильно.

Теперь добавим каждому из падающих объектов по собственному источнику звука (Рисунок 7).

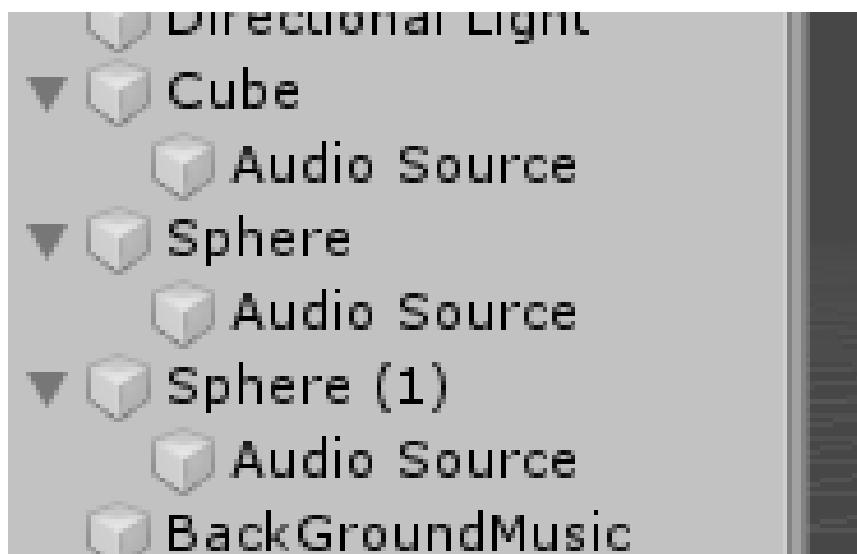


Рисунок 77 — Добавление к объектам дочернего Audio Source.

Создадим новый C# скрипт под названием AudioPlay и откроем его в редакторе. В нем пишем не замысловатый код, где при столкновении с каким-нибудь объектом будет проигрываться клип. (Рисунок 8).

```
Ссылка: 0
public class SoundEffect : MonoBehaviour
{
    public AudioSource SoundEff;
    Ссылка: 0
    private void OnCollisionEnter(Collision collision)
    {
        SoundEff.Play();
    }
}
```

Рисунок 78 — Код для проигрывания аудиоклипа.

Повесим на каждый падающий объект по скрипту и добавим ссылку на собственный AudioSource каждого из объектов и настроим сами источники звука.

В инспекторе добавляем в проигрываемый клип Land из стандартных ассетов, output группа будет SoundEff и убираем все галочки из параметров (Рисунок 9).

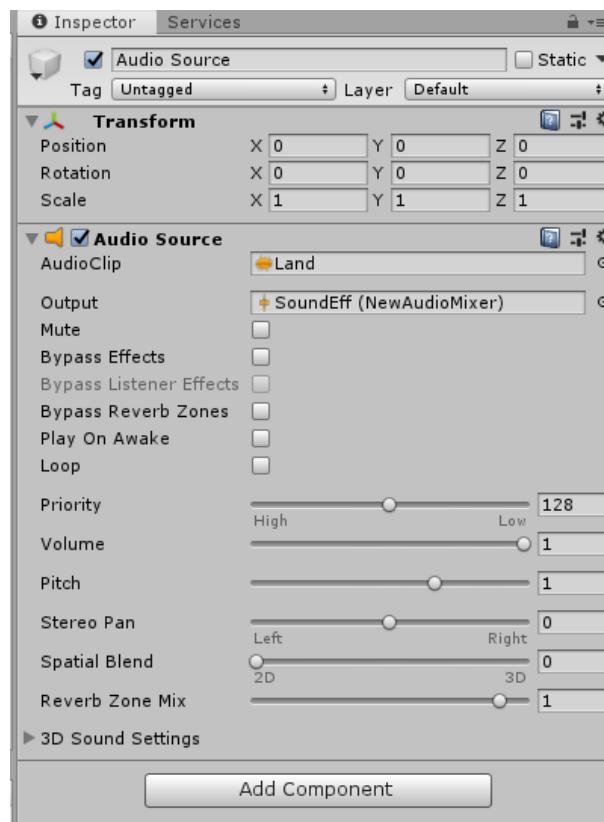


Рисунок 79 — Пример параметров для дочерних Audio Source.

Теперь запускаем сцену и проверяем, если необходимо, можно отрегулировать звук через Mixer.

Результат работы показать преподавателю.

Практическая работа 10

Цель практической работы: изучить свойства и принципы работы с системой частиц.

Задачи:

- Подготовить сцены;
- Добавить и настроить систему частиц с настройками для пара;
- Добавить и настроить систему частиц с настройками для падающего снега;
- Написать переключение между сценами через клавиши E и R;
- Продемонстрировать преподавателю.

Описание выполнения работы

Перед началом практической работы нужно подготовить две сцены к этому занятию. Создаем вторую сцену и добавляем на каждую из них по плейну (Рисунок 1). Сохраняем и пока удаляем Scene2 через правую кнопку мыши и вариант из контекстного меню Remove Scene.



Рисунок 80 — Иерархия объектов на двух сценах.

Создаем новую систему частиц (particle system) в Scene1 как показано на рисунке 2.

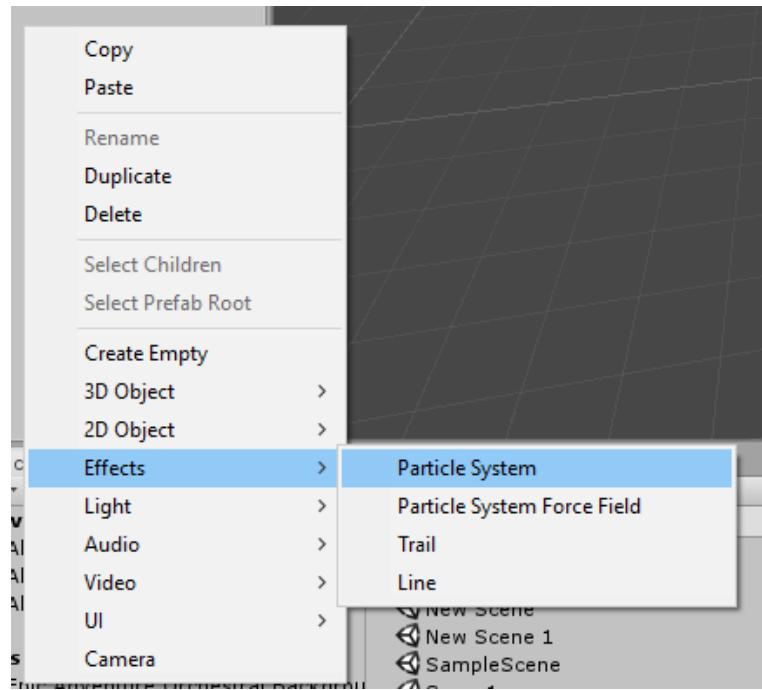


Рисунок 81 — Создание системы частиц.

Система частиц очень мощный инструмент, с его помощью можно создать почти любой спецэффект. На этом практическом занятии будет рассмотрено создание эффектов пара и снега. Для начала нужно выбрать отображение единичного элемента, для этого нужно зайти в раздел Renderer и задать материал из папки Standard assets файл с названием ParticleSmokeMobile (Рисунок 3).

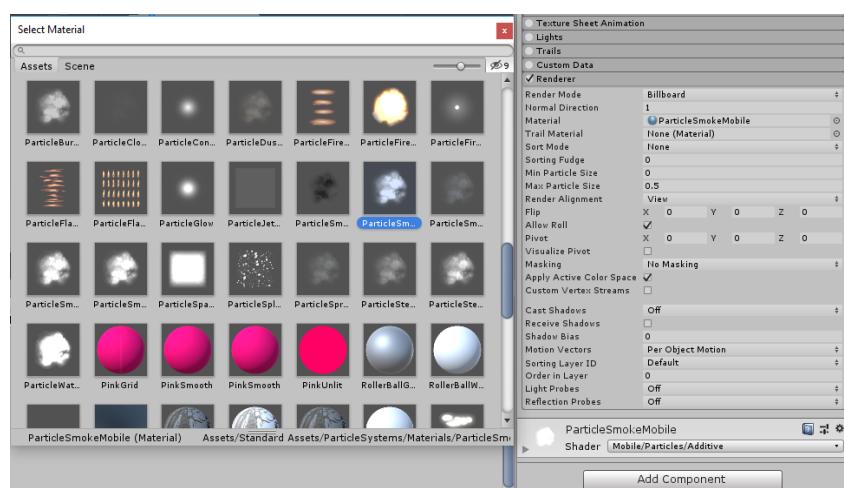


Рисунок 82 — Выбор материала для элементов.

После этой операции стандартный материал заменится на новый и из объекта станут вылетать заданные элементы. В главных параметрах зададим

StartSpeed 0, чтобы элементы не вылетали из объекта, а оставались в радиусе их создания (Рисунок 4).

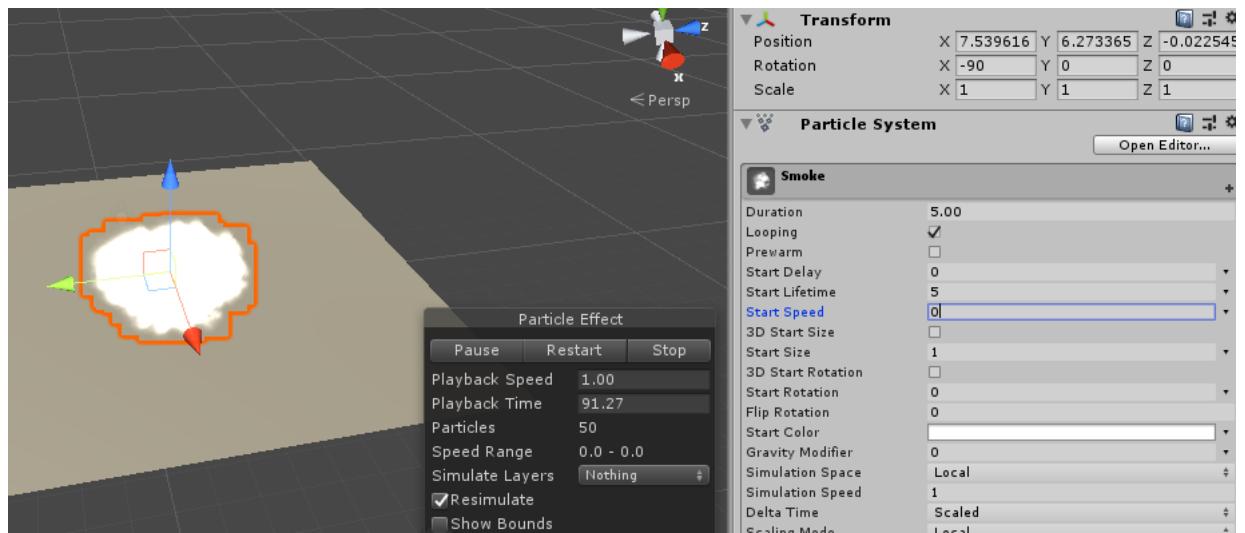


Рисунок 83 — Выставление параметра Start Speed на 0.

В StartSize выберем размер между двумя случайными числами, чтобы каждый элемент был не похож на предыдущий, для появления контекстного меню следует нажать стрелочку вниз справа от параметра (Рисунок 5).

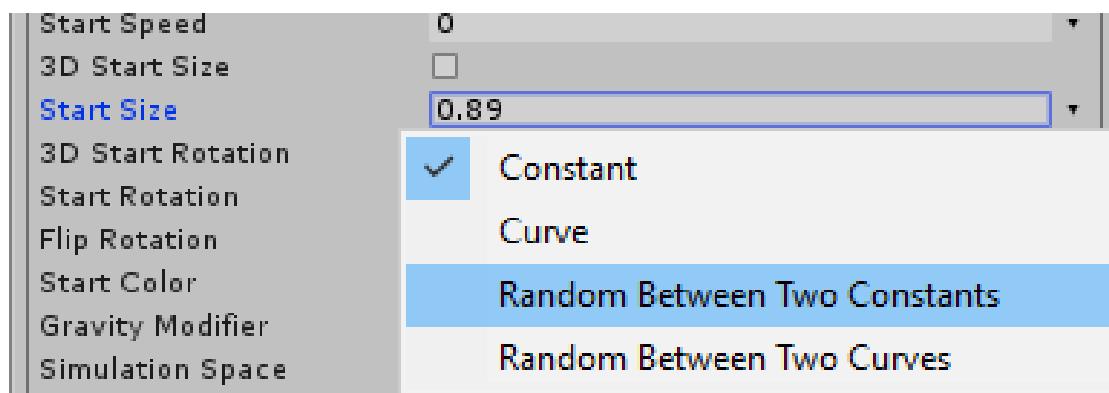


Рисунок 84 — Включение случайного числа между двумя константами при определении начального размера элемента.

В этих константах нужно будет задать значения 20 и 60.

Чтобы объект был не плоский, а объёмный, для этого поставим галочку на параметре 3D Start Size и значения зададим в остальных координатах (Рисунок 6).

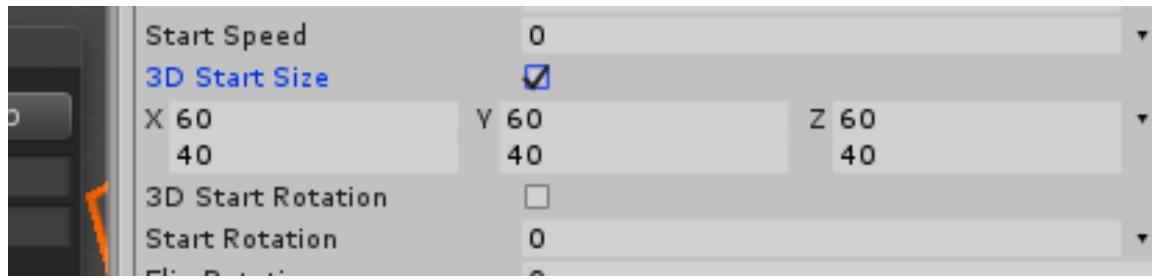


Рисунок 85 — Настройка параметра 3D Start Size.

Теперь следует сделать пар более прозрачным, для этого всего нужно изменить параметр StartColor. Стоит обратить внимание, что за передачу прозрачности отвечает альфа канал (Рисунок 7).

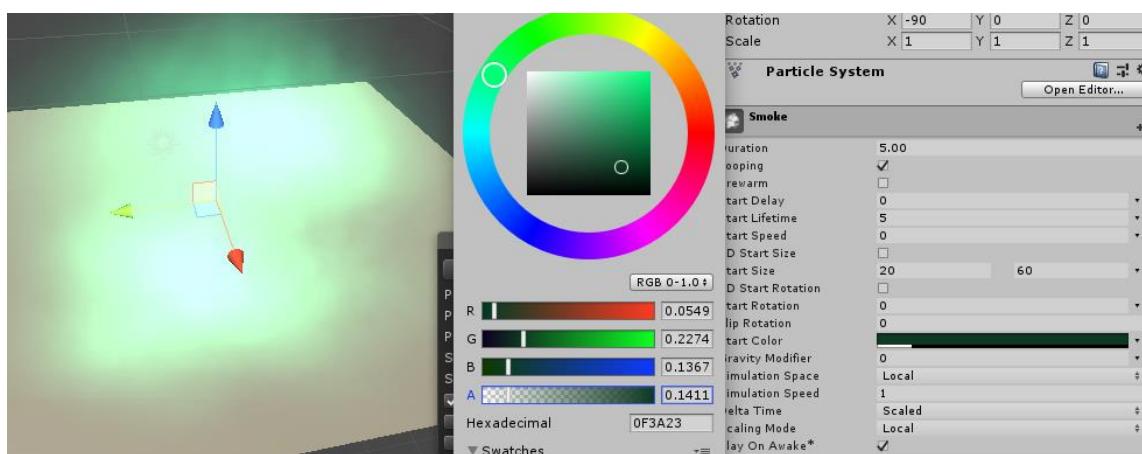


Рисунок 86 — Установка цвета.

Параметр Max Particles стоит снизить до 30, чтобы не создавать лишнего шума.

Для более живости эффекта нужно включить функцию Rotation over lifetime или другими словами вращение пока элемент и исчезнет. Там всего один параметр, который отвечает за угол поворота за жизненный цикл элемента, ему ставим значение 15 (Рисунок 8).

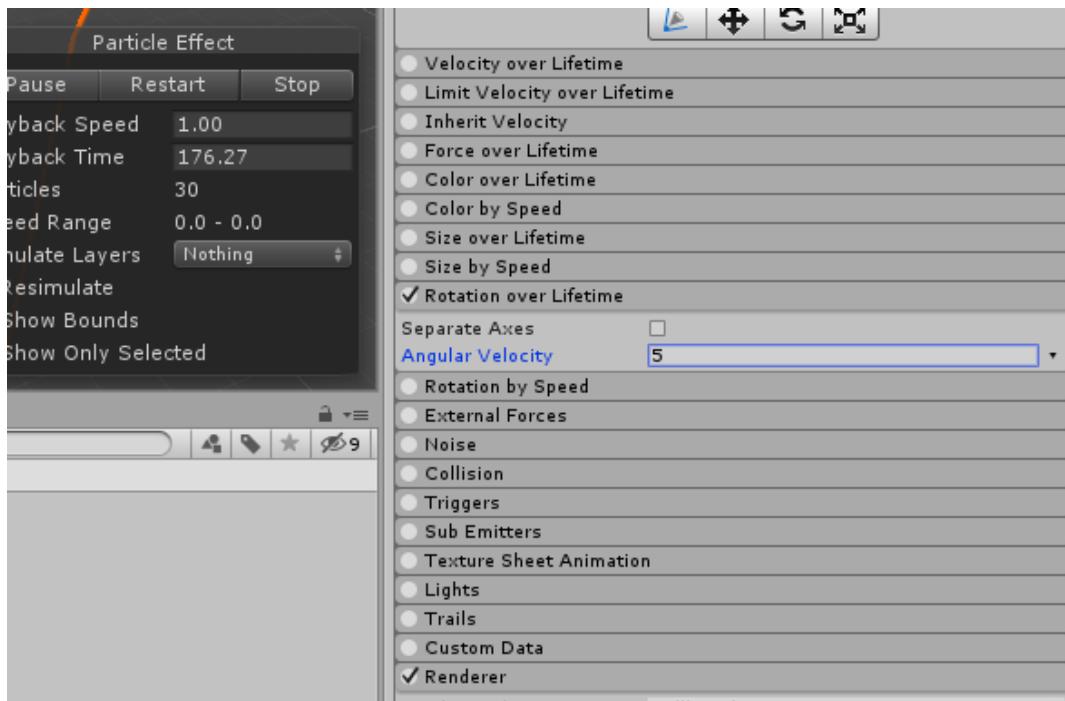


Рисунок 87 — Угол поворота элемента.

Запустим сцену и проверим работоспособность (Рисунок 9). Если все сделано верно, то можно приступить к эффекту на второй сцене. Открываем Scene2 в иерархии, текущую сцену сохраняем и нажимаем на ней Remove Scene, как уже делалось ранее с Scene2.

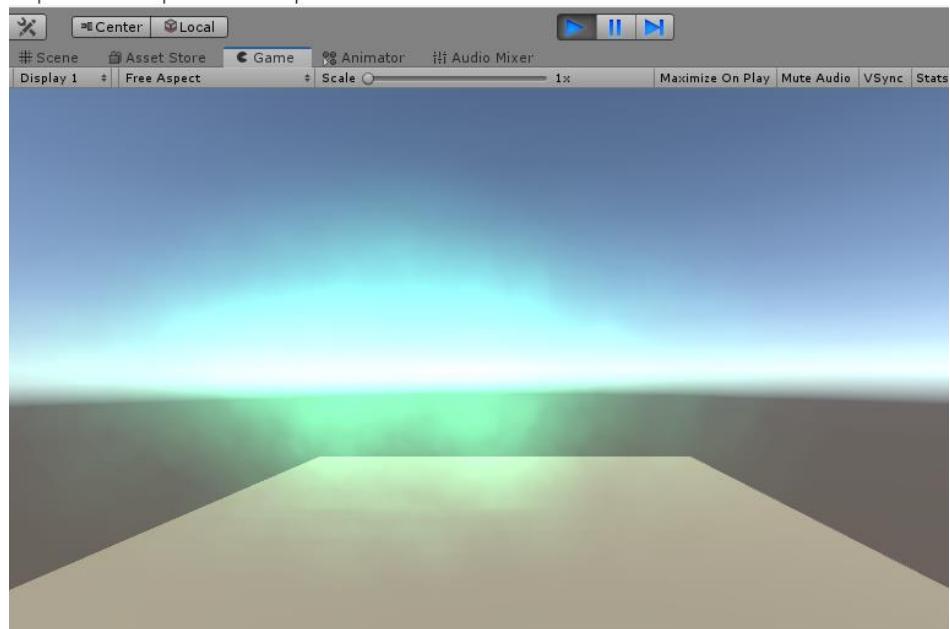


Рисунок 88 — Пример Scene1.

На ней также создаем новый объект системы частиц, и в Renderer задаем материал ParticleSplashes (Рисунок 10).

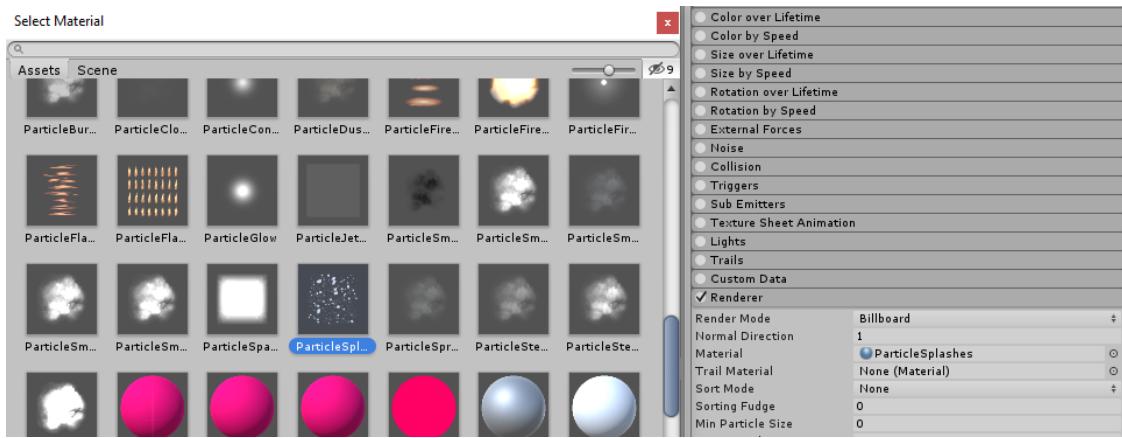


Рисунок 89 — Выбор материала для создания снега.

Сразу же зайдем в функцию Shape, зададим форму Box, поставим параметры Scale на 9 и 9 по X и Y, перевернем фигуру через Rotation, укажем в этом параметре Y = 180 (Рисунок 11).

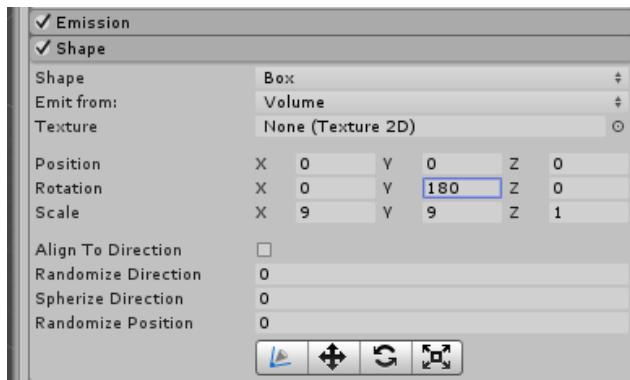


Рисунок 90 — Установка параметров в функции Shape.

В функции Emission поставим Rate over time на значение 400, увеличив Max Particles до 5000. Добавим Start Speed вернем на 0 и добавим функцию Velocity over lifetime. Зададим значения по Y между двумя случайными как показано на рисунке 12.

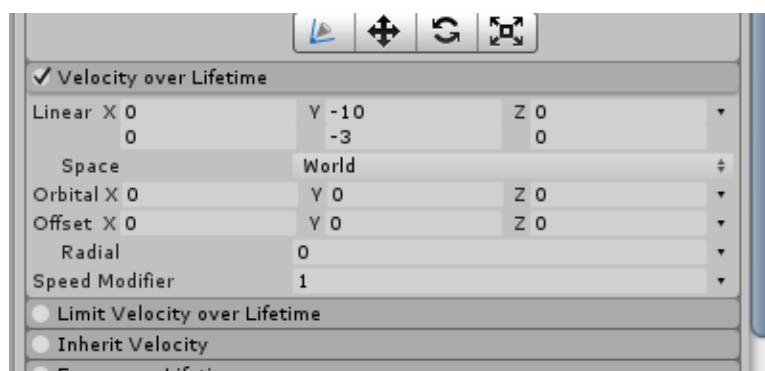


Рисунок 91 — Установка параметров для Velocity over lifetime.

Осталось только задать цвет в главных настройках и задать 3D start size в главных настройках и можно запускать сцену (Рисунок 13).

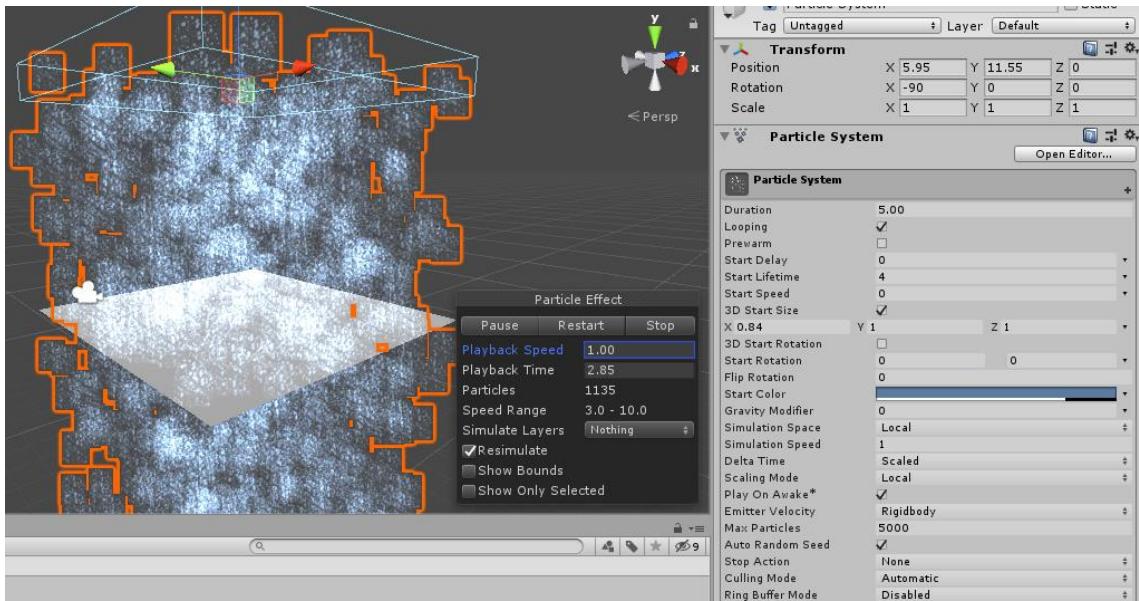


Рисунок 92 — Установка основных параметров для снега.

Для наглядности можно поменять цвет окружения на камере на какой-нибудь темный оттенок, это делается путем замены параметра Clear Flags с Skybox на Solid color.

При запуске сцены должен получиться подобный эффект как на рисунке 14.

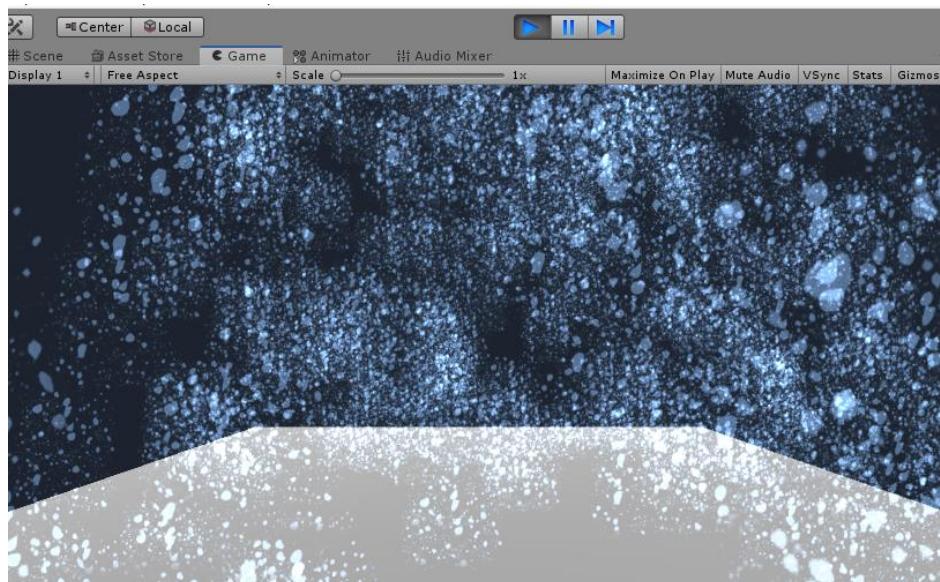


Рисунок 93 — Пример Scene2.

Работа почти завершена, осталось сделать переход между сценами по нажатию клавиш. Создадим новый C# script под название SceneChange, в нем

напишем код, с помощью которого можно будет переключать эти две сцены между собой по нажатию клавиш. В эдиторе пишем условие для клавиш E и R и функцию, которая загружает сцену (Рисунок 15).

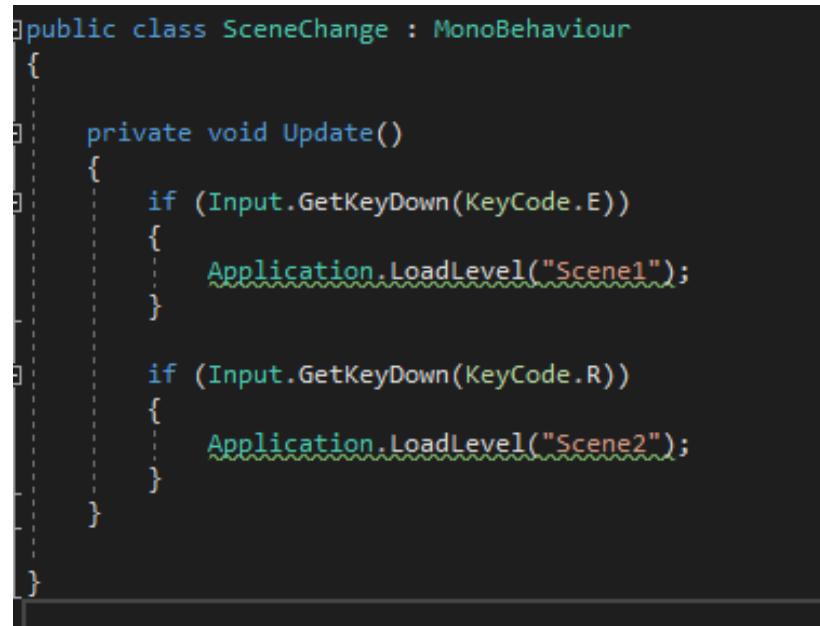


Рисунок 94 — Код для скрипта SceneChange.

Сохраняем и добавляем этот скрипт на каждую из камер на двух сценах, а также скрываем Scene2, чтобы они между собой не пересекались (Рисунок 16).

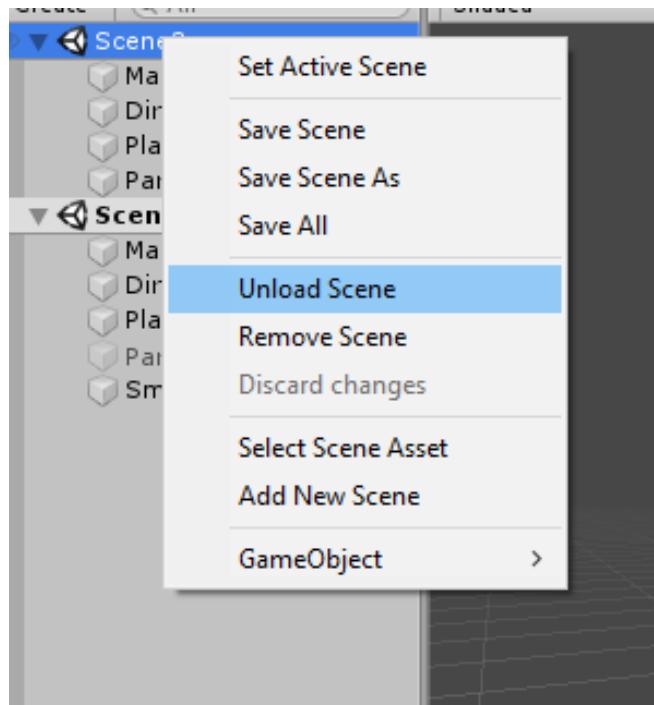


Рисунок 95 — Отгрузка сцены.

Теперь проверяем работоспособность, если все работает верно, сцены переключаются на клавиши E и R, то работа выполнена правильно и ее можно показать преподавателю.

Практическая работа 11

Цель практической работы: изучить свойства и принципы работы с UI элементами в Unity.

Задачи:

- Создать панель с различными UI элементами;
- Написать функционал для UI кнопок;
- Продемонстрировать преподавателю.

Описание выполнения работы

UI (user interface) - интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы. Элементами пользовательского интерфейса будут являться: кнопки, ползунки, картинки, текст и многое другое.

Все UI элементы будут отображаться с помощью специального объекта – Canvas (полотно). Для того, чтобы UI элементы отображались, нужно сделать их дочерними для Canvas'a.

Создаем Canvas через контекстное меню в иерархии через пункт UI (Рисунок 1).

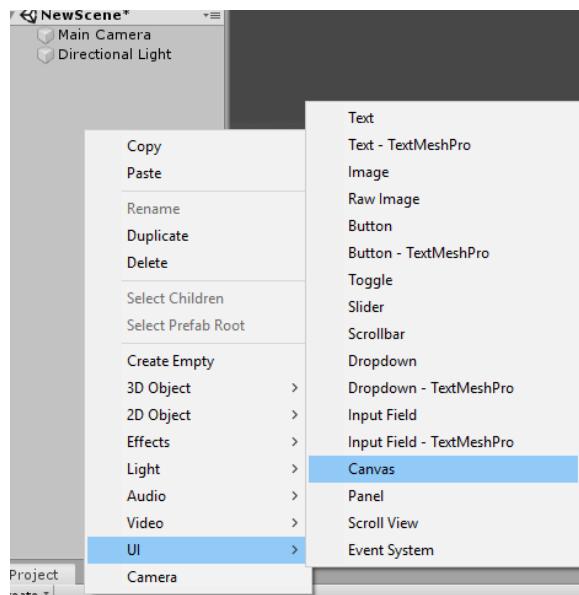


Рисунок 96 — Создание Canvas.

Помимо полотна создаётся автоматически объект EventSystem, это тот инструмент, который обрабатывает все события на сцене, например, нажатие клавиши мыши на экране. Вернемся к Canvas'у и создадим через UI объект Panel (Рисунок 2).

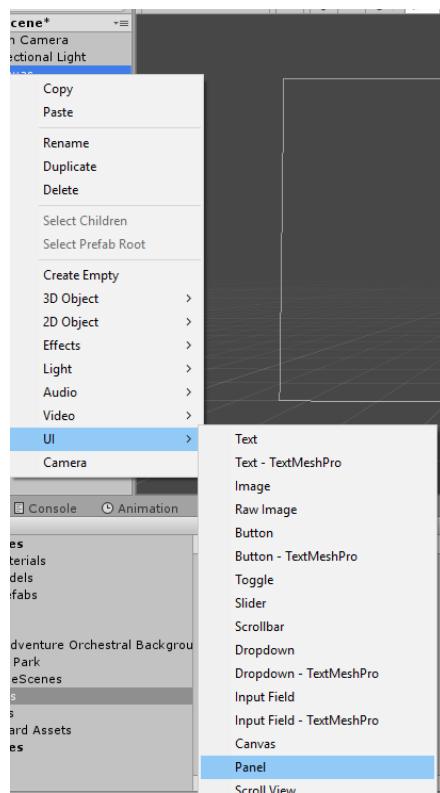


Рисунок 97 — Создание Panel.

Если перейти на Canvas, то можно увидеть, как появилась новая панель серого цвета. Её можно немного уменьшить с помощью инструмента Rect tool (Рисунок 3). Чтобы сделать его не прозрачным, нужно поменять альфа канал в цвете объекта.

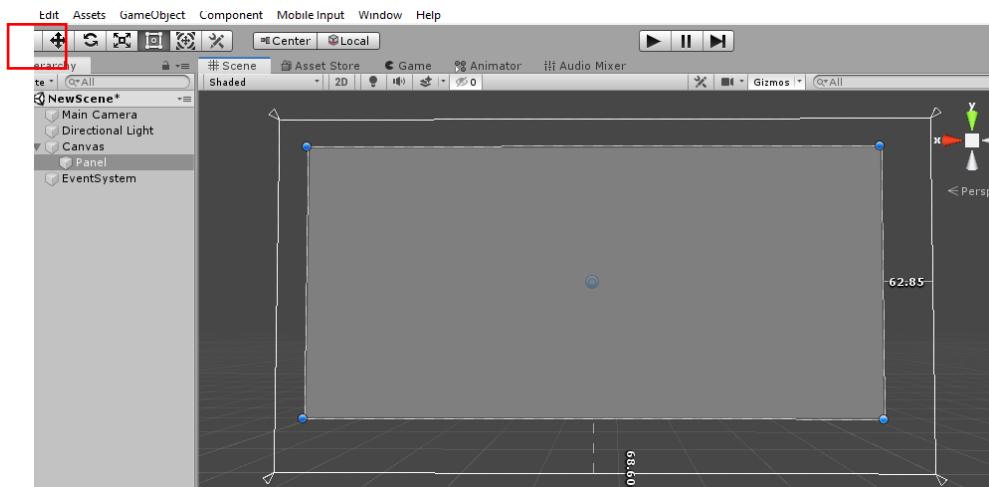


Рисунок 98 — Применение инструмента Rect Tool.

Теперь создадим две кнопки (Button) и одно изображение (Image). Обязательно сделать их дочерними к объекту Panel. Создаются кнопки и изображения таким же образом, как и панель (Рисунок 4).

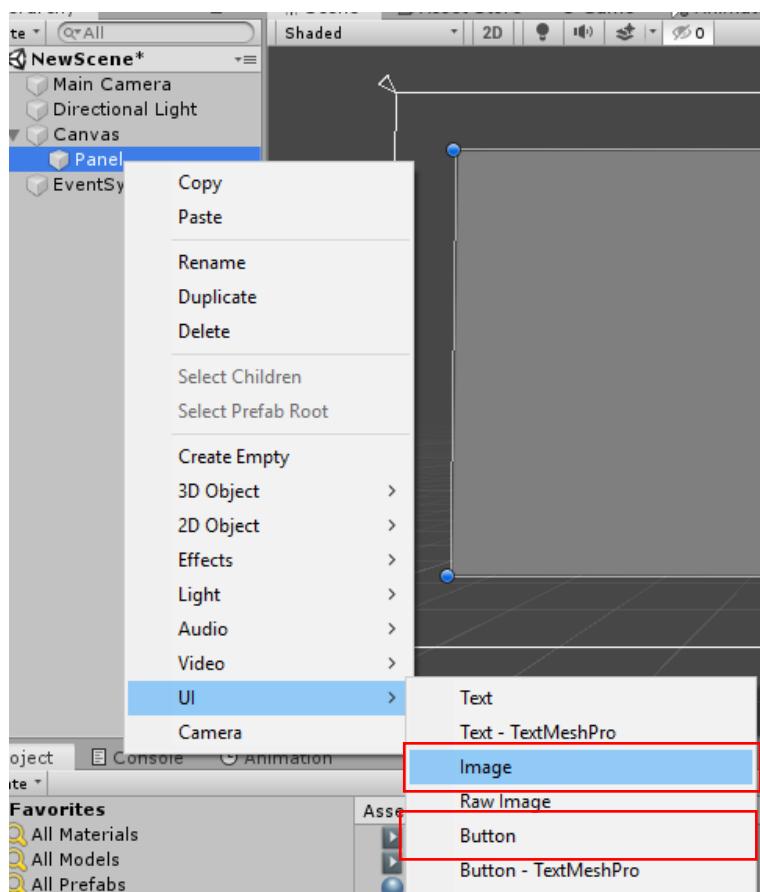


Рисунок 99 — Создание Image и Button.

Зададим для Image и Button подходящие размеры (Рисунок 5).

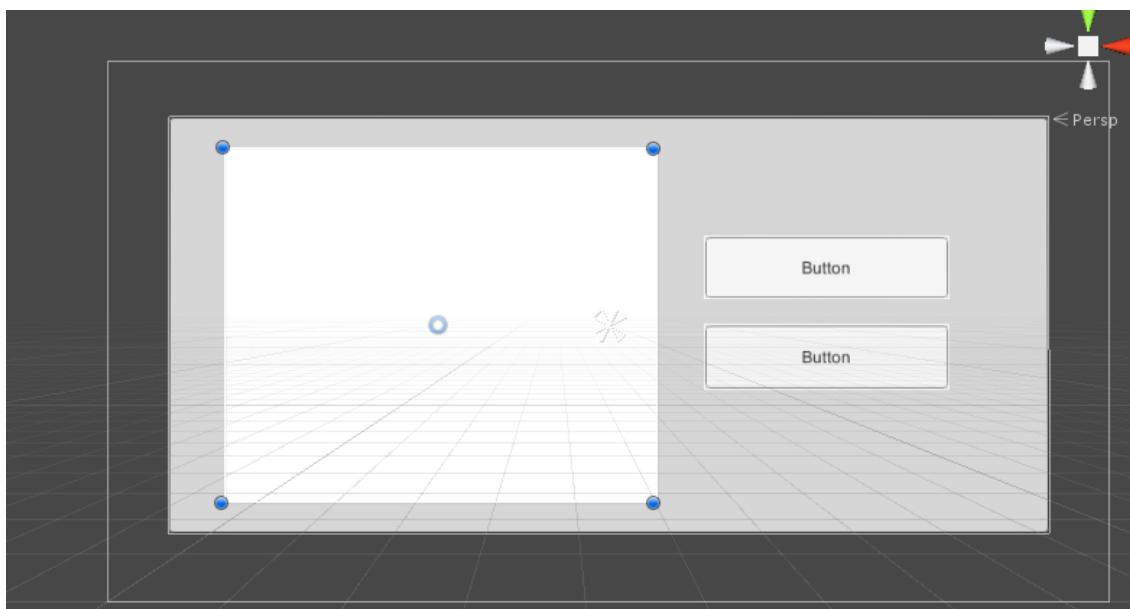


Рисунок 100 — Пример сцены.

Теперь нужно поставить для Image ссылку на то, что этот объект будет показывать, за отображение отвечает параметр Source Image. Например, возьмем спрайт ButtonBrakeUpSprite из папки Standard Assets (Рисунок 6).

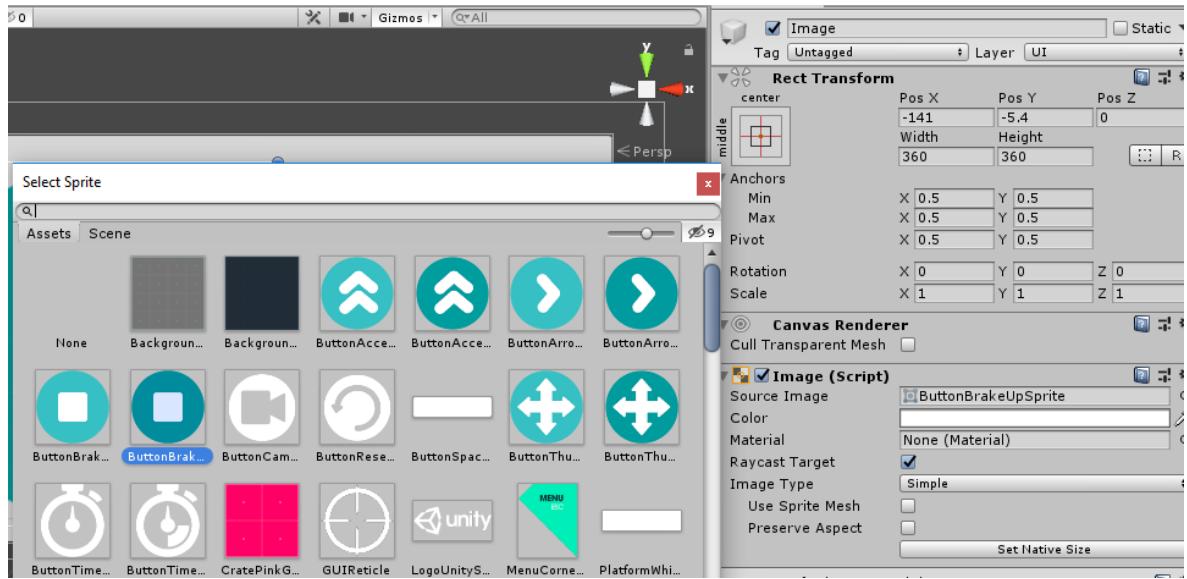


Рисунок 101 — Изменение параметра Source Image.

Теперь перейдем к элементу текста на кнопках через иерархию. Через одноименный компонент можно редактировать отображаемый текст. Для одной кнопки нужно написать “Смена изображения”, для другой “Закрыть” (Рисунок 7).

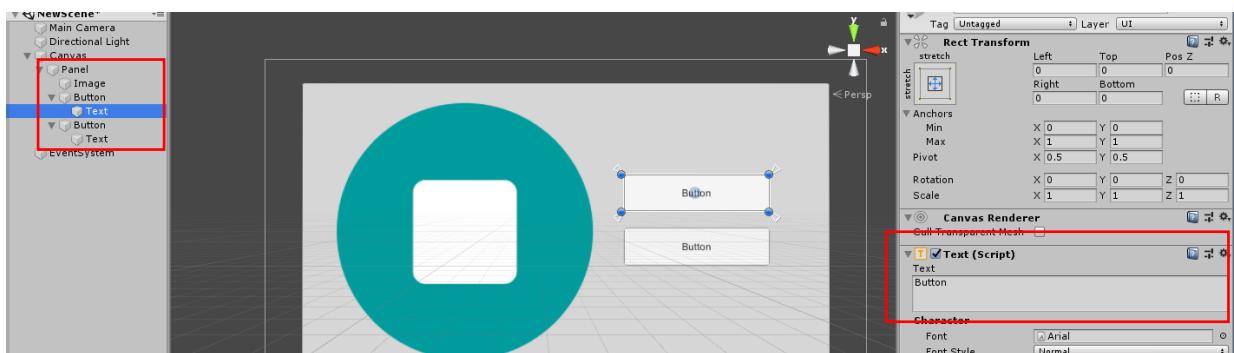


Рисунок 102 — Изменение текста на кнопках.

На рисунке 8 показан результат проделанных действий, которые были описаны выше.

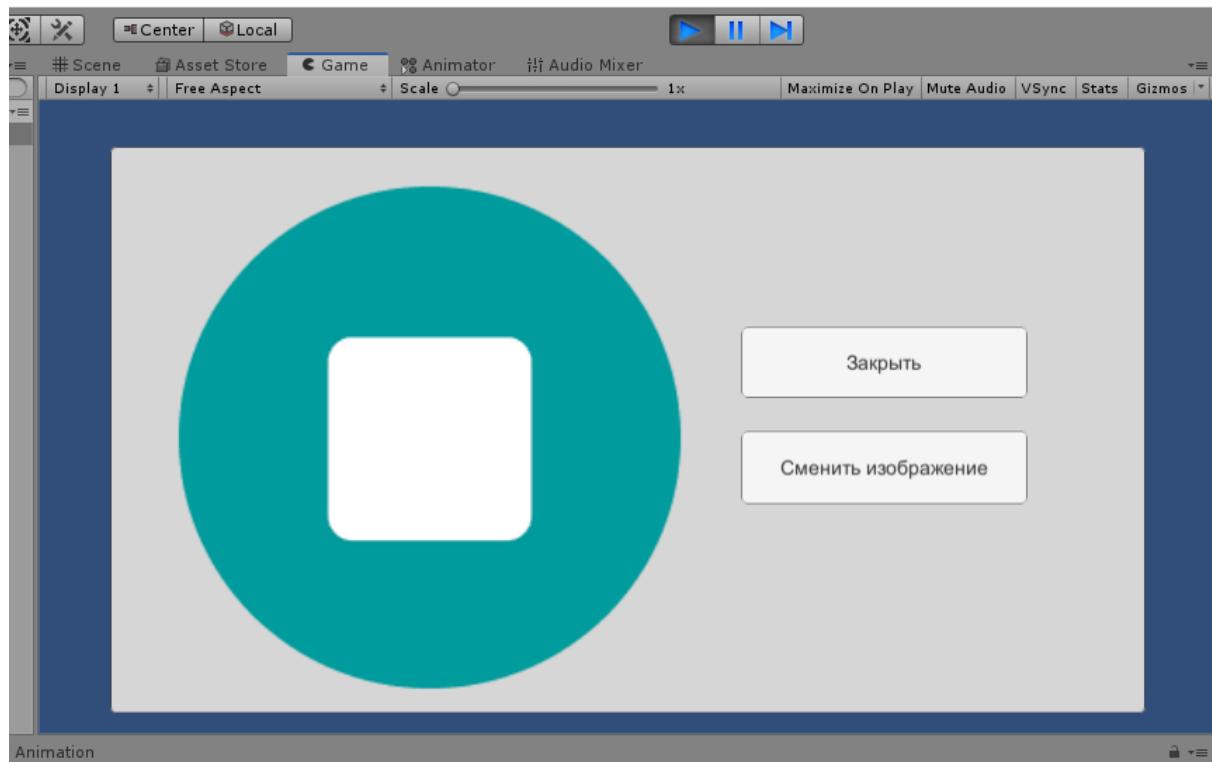


Рисунок 103 — Пример сцены при запущенной сцене.

Теперь нужно прописать функционал для этих двух кнопок, создадим новый C# скрипт с названием ButtonScript. Для этого скрипта понадобится подключить UI библиотеку. Чтобы ее подключить, напишем перед классом строчку: `using UnityEngine.UI`.

Создадим три переменные, одна будет содержать ссылку на панель, другая ссылку на изображение на сцене, а последняя ссылку на новое изображение.

Для того, чтобы две кнопки работали, нужно написать соответствующие методы, которые будут вызываться по клику.

Первый метод `Close()` закрывает окно, через метод `SetActive`. Второй метод `ChangeImage()` меняет изображение, обращаясь к параметру спрайт через компонент `Image` (Рисунок 9).

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ButtonScript : MonoBehaviour
{
    public GameObject gameObjectPanel;
    public GameObject gameObjectImage;
    public Sprite Newsprite;

    public void Close()
    {
        gameObjectPanel.SetActive(false);
    }

    public void ChangeImage()
    {
        gameObjectImage.GetComponent<Image>().sprite = Newsprite;
    }
}

```

Рисунок 104 — Код для скрипта ButtonScript.

Сохраняем полученный код и добавляем этот скрипт, например, на EventSystem. Важный момент, везде в параметрах скрипта нужно указать ссылки: на новое изображение, на панель и на изображение, которое будет меняться (Рисунок 10).

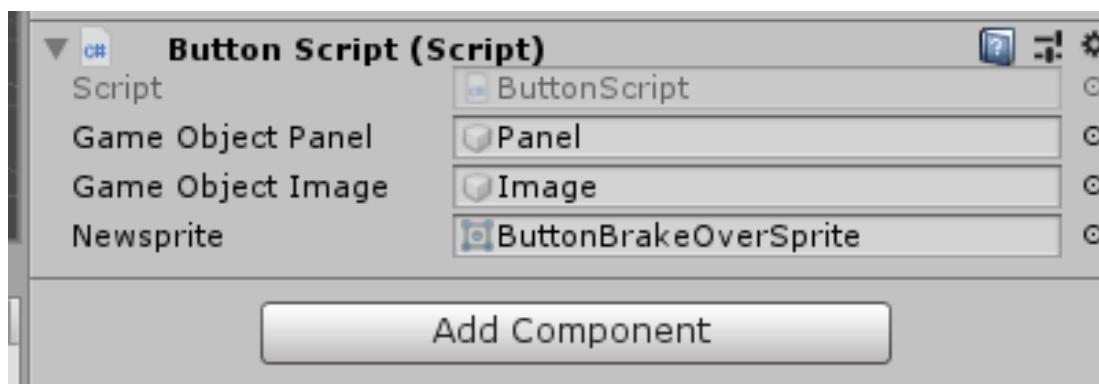


Рисунок 105 — Заданные ссылки в скрипте.

Теперь откроем объекты кнопок в инспекторе, найдем параметр OnClick, нажмем на плюсик и укажем ссылку на объект EventSystem, это значит, что скрипт, который будет срабатывать, находится на этом объекте (Рисунок 11).

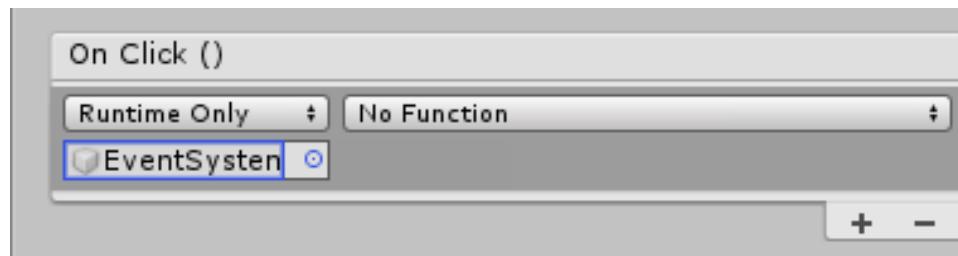


Рисунок 106 — Указание ссылки на EventSystem (носитель скрипта).

Нажав на кнопку No function, появится меню, где нужно будет выбрать название написанного скрипта и метод, соответствующий кнопке (Рисунок 12). Повторить эти действия с применением второго метода для второй кнопки.

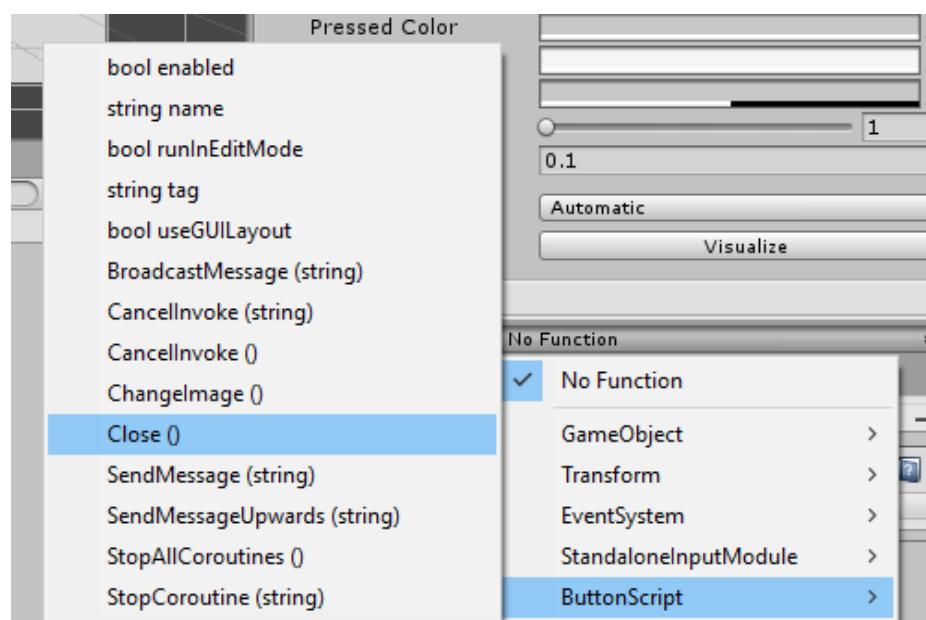


Рисунок 107 — Присвоение метода при нажатии на кнопку.

Запускаем сцену и проверяем работоспособность, если на кнопку “Сменить изображение”, картинка меняется, а на кнопку “Закрыть”, панель пропадает, то все выполнено правильно.

Результат показать преподавателю.

Практическая работа 12

Цель практической работы: создать пользовательский интерфейс для меню и изучить работу билдинга проектов.

Задачи:

- Сделать пользовательский интерфейс для меню;
- Написать функционал для меню;
- Сделать сборку проекта под платформу Windows;
- Продемонстрировать преподавателю.

Описание выполнения работы

В данной практической работе, нужно будет создать меню из стандартных элементов Unity, написать функционал через скрипт и сделать сборку проекта под платформу Windows.

Для этого занятия потребуется две сцены, сцена главного меню и сцена для перехода. Вторая сцена должна наименоваться как Scene2.

Для начала нужно создать объекты Canvas и Plane, задать для второго приемлемую ширину и высоту (Рисунок 1).

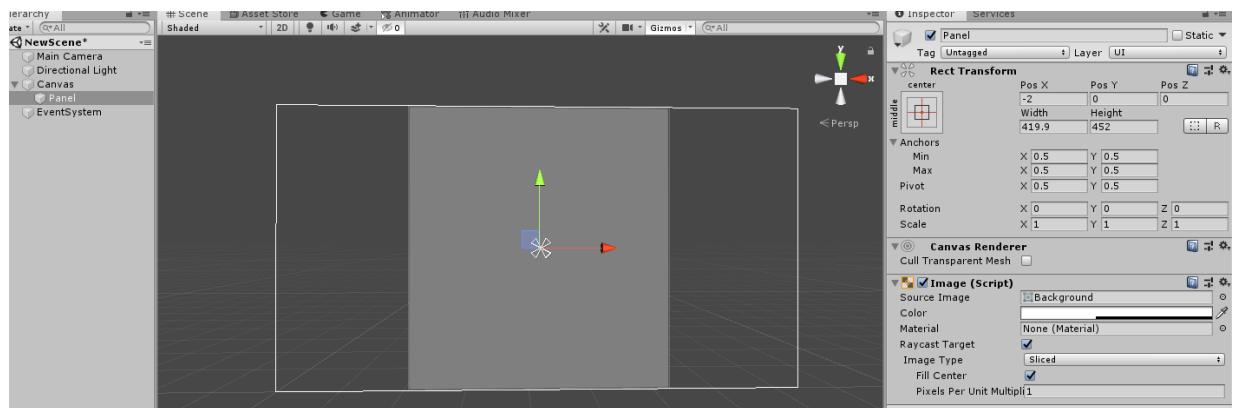


Рисунок 108 — Созданные Plane и Canvas.

Этот Plane будет своего рода держателем всех остальных элементов для меню, поэтому можно его соответствующе переименовать в Menu.

Теперь нужно добавить кнопки. В самом меню будет располагаться 3 кнопки:

1. Start – запускает другую сцену;
2. Options – открывает окно, в которой можно регулировать звук;

3. Quit – закрывает приложение.

Создаем эти три кнопки, переименовываем и в тексте кнопке пишем соответствующее кнопке название (Рисунок 2).

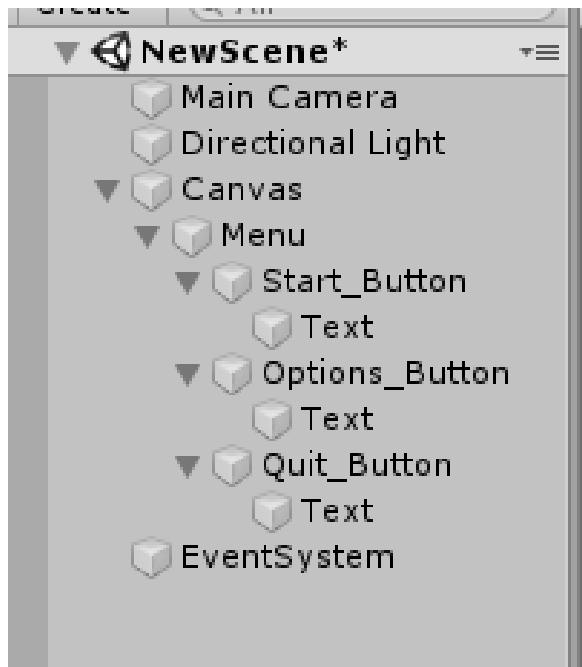


Рисунок 109 — Иерархия объектов в меню.

Теперь перейдем к оформлению, кнопки стоит расположить на канвасе в порядке, данном выше. Задать им одинаковые размеры применить цвет и задать спрайты. Для меню и кнопок можно использовать свои вариации спрайтов или применить к ним TouchpadSprite (Рисунок 3).

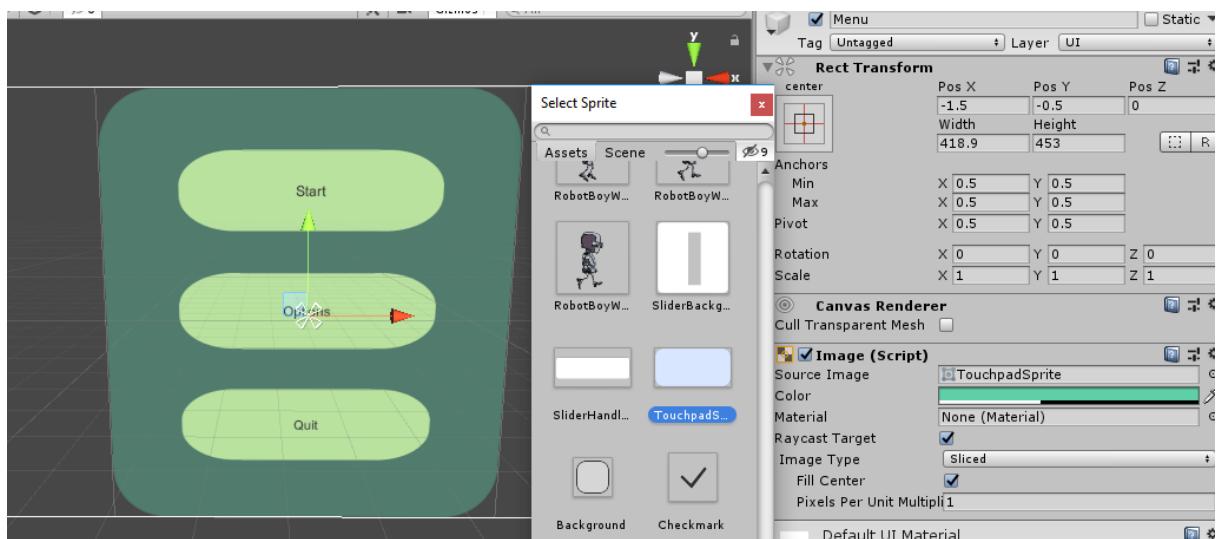


Рисунок 110 — Смена сцена и спрайта у UI элементов.

Для того, чтобы копировать уже использовавшийся цвет, существует инструмент пипетка, он расположен около параметра Color. После активации, необходимо просто навестись на объект с желаемым цветом и кликнуть по нему.

Что касается текста, его необходимо увеличить до приемлемых размеров через параметр Font Size, чтобы текст не уходил за границы кнопки и поставить Font Style на значение Bold (Рисунок 4).

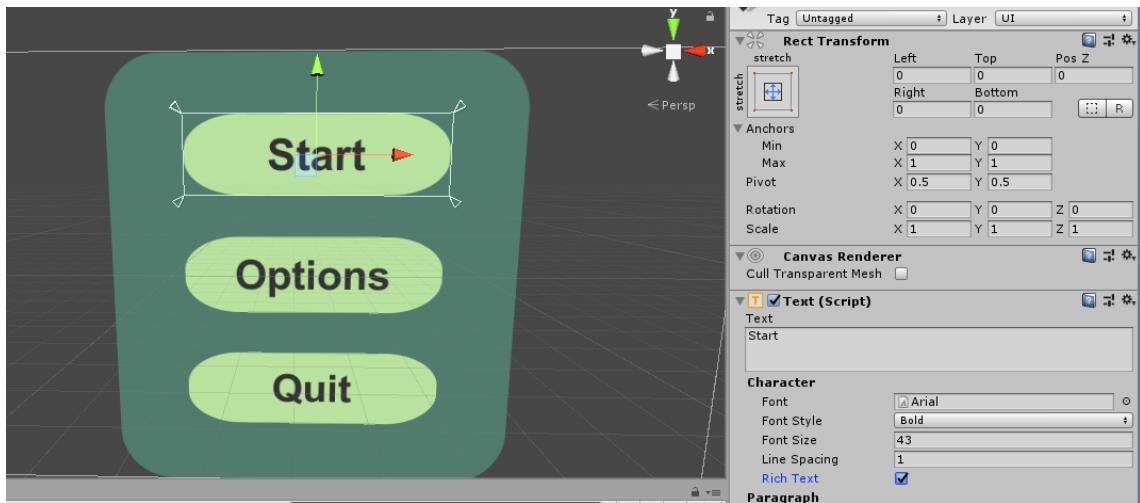


Рисунок 111 — Редактирование текста.

Создаем новый C# скрипт, в нем прописываем 3 метода, по одному на каждую кнопку (Рисунок 5).

```
public class MenuButton : MonoBehaviour
{
    public void StartScene()
    {
        Application.LoadLevel("Scene2");
    }

    public void Options(GameObject window)
    {
        window.SetActive(true);
    }

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

Рисунок 112 — Код для функционала кнопок.

Сохраняем скрипт и добавляем его на Canvas.

Каждой из кнопок задаем ее собственный функционал, как это делалось в практической работе 11 (Рисунок 6).

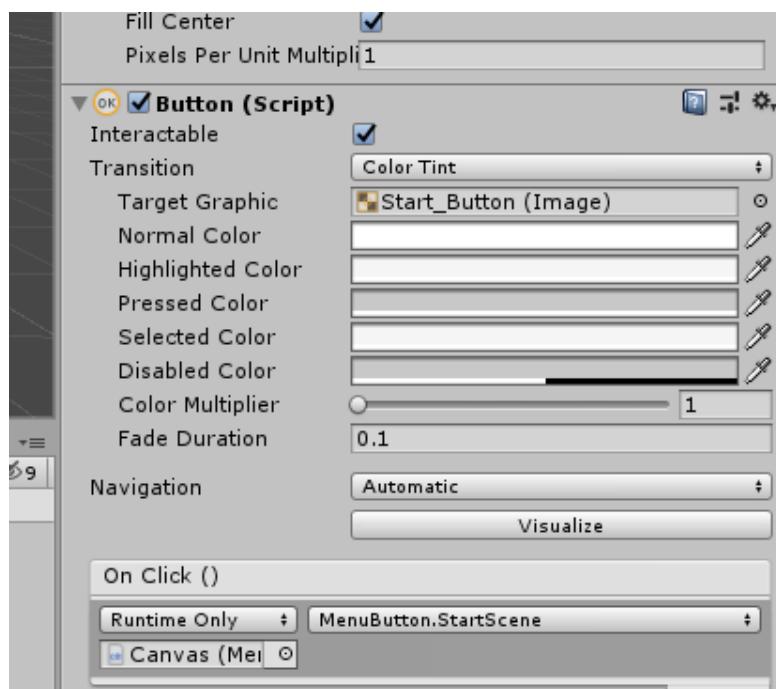


Рисунок 113 — Применение функционала к кнопке.

Для кнопки Options потребуется задать ссылку на аргумент GameObject, поэтому создадим новую панель с надписью сверху Options и элементом, помимо этого, нужно сделать кнопку закрытия этого окна и переход в главное меню. Для этого переходим к методу OnClick() на кнопке, перетаскиваем объект, который следует закрыть и в функционале указывает GameObject.SetActive, если галочка стоит (true), значит окно будет появляться, если не стоит (false), то окно закроется (Рисунок 7).

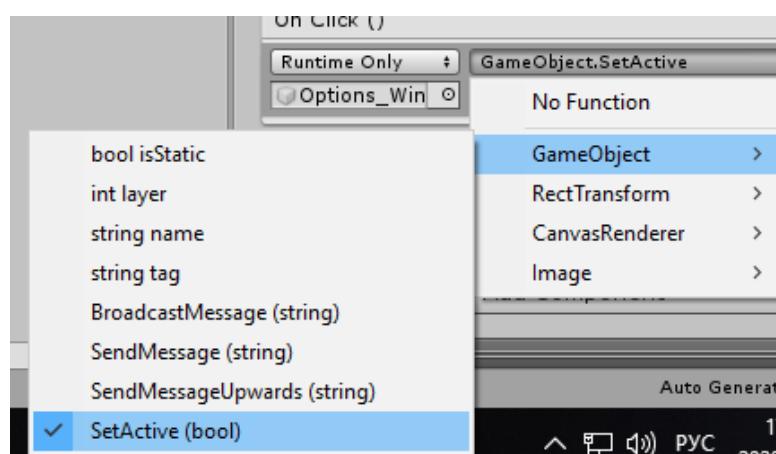


Рисунок 114 — Функционал к кнопке закрытия.

Для удобной регулировки звука необходим ScrollBar, этот элемент создается через ту же вкладку UI (Рисунок 8).

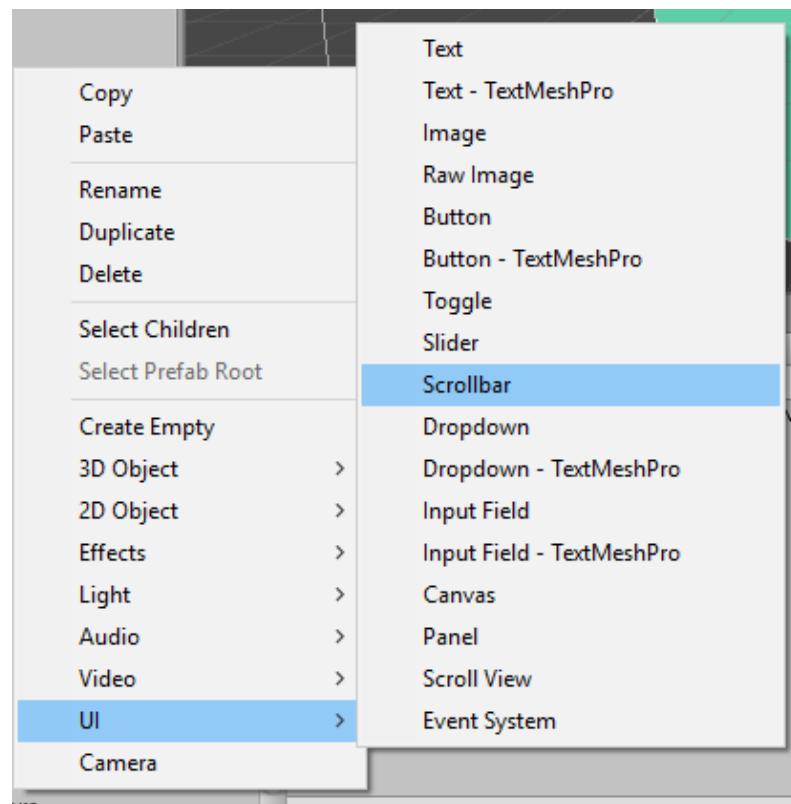


Рисунок 115 — Создание Scrollbar элемента.

Создаем новый Audio Source. В него можно поставить любой аудиокlip в Loop режиме (Рисунок 9).

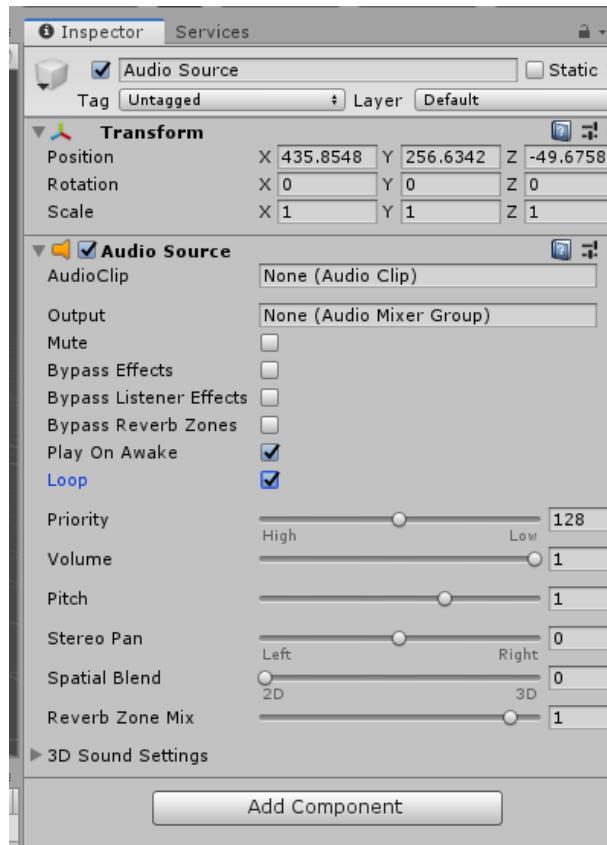


Рисунок 116 — Настройки для нового Audio Source.

Осталось задать этому элементу ссылку на параметр, значение которого он будет регулировать, а именно Volume (звук). В параметрах скроллбара есть метод `OnValueChanged()`, в нем указываем новый Audio Source и ссылаемся на значение звука (Рисунок 10).

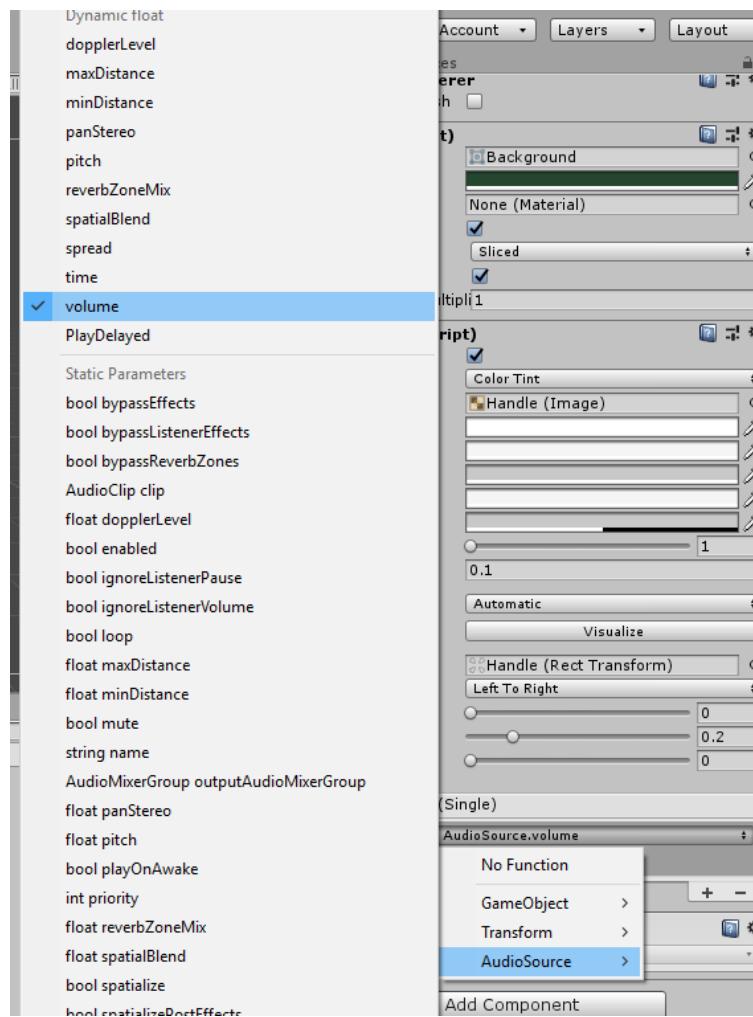


Рисунок 117 — Установка ссылки на меняющийся параметр.

Через галочку у объекта Options_Window скрываем его (Рисунок 11) и проверяем функционал, если все работает верно, то можно приступить к завершающему этапу.

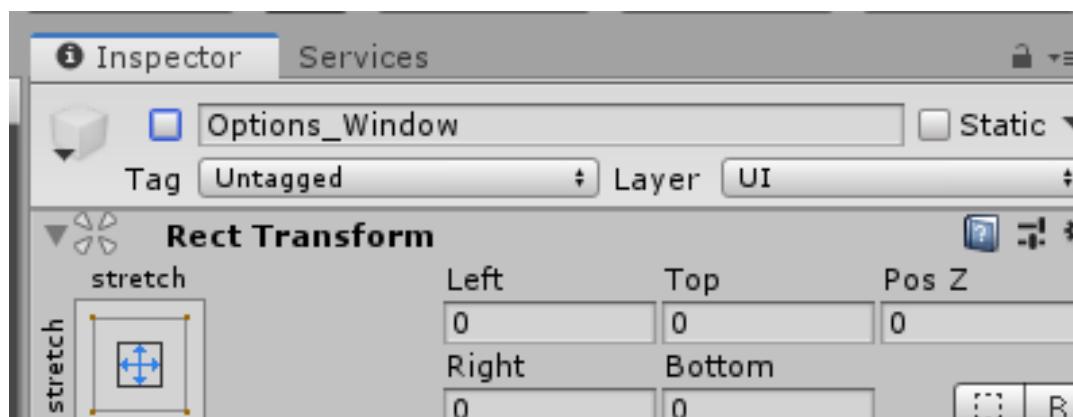


Рисунок 118 — Отключение окна.

Для перехода в настройки билда необходимо перейти через File в Build Settings (Рисунок 12).

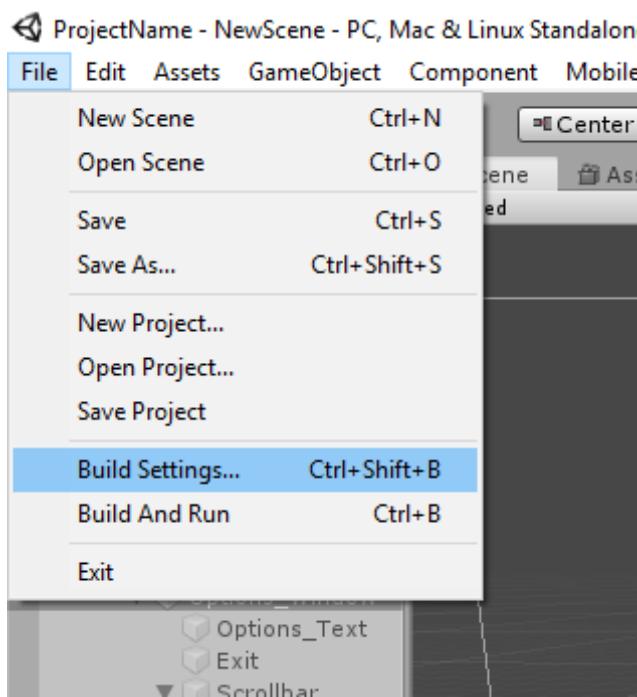


Рисунок 119 — Переход в настройки сборки проекта.

Откроется окно сборки проекта, в верхнюю часть нужно перенести две сцены, с индексом 0 будет главное меню, а с индексом 1 будет вторая сцена. Архитектуру следует поменять на x86_64 (Рисунок 13).

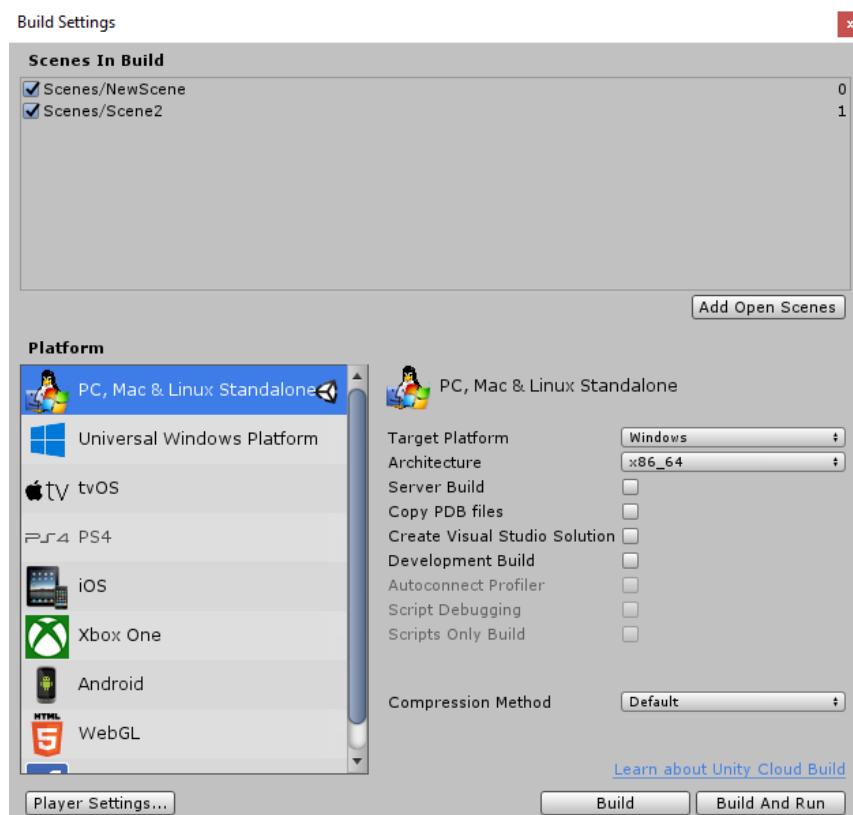


Рисунок 120 — Настройки сборки проекта.

Для завершения всех настроек, нажимаем кнопку Build, указываем папку, где будет храниться сборка.

Открываем папку со сборкой и запускаем файл .exe с названием проекта (Рисунок 14).

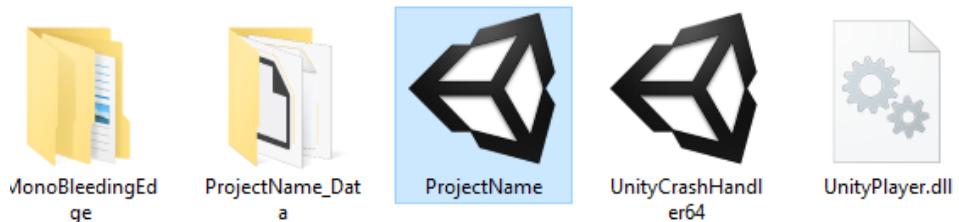


Рисунок 121 — Файл запуска приложения.

На рисунках 15, 16 показан пример, как может выглядеть приложение.

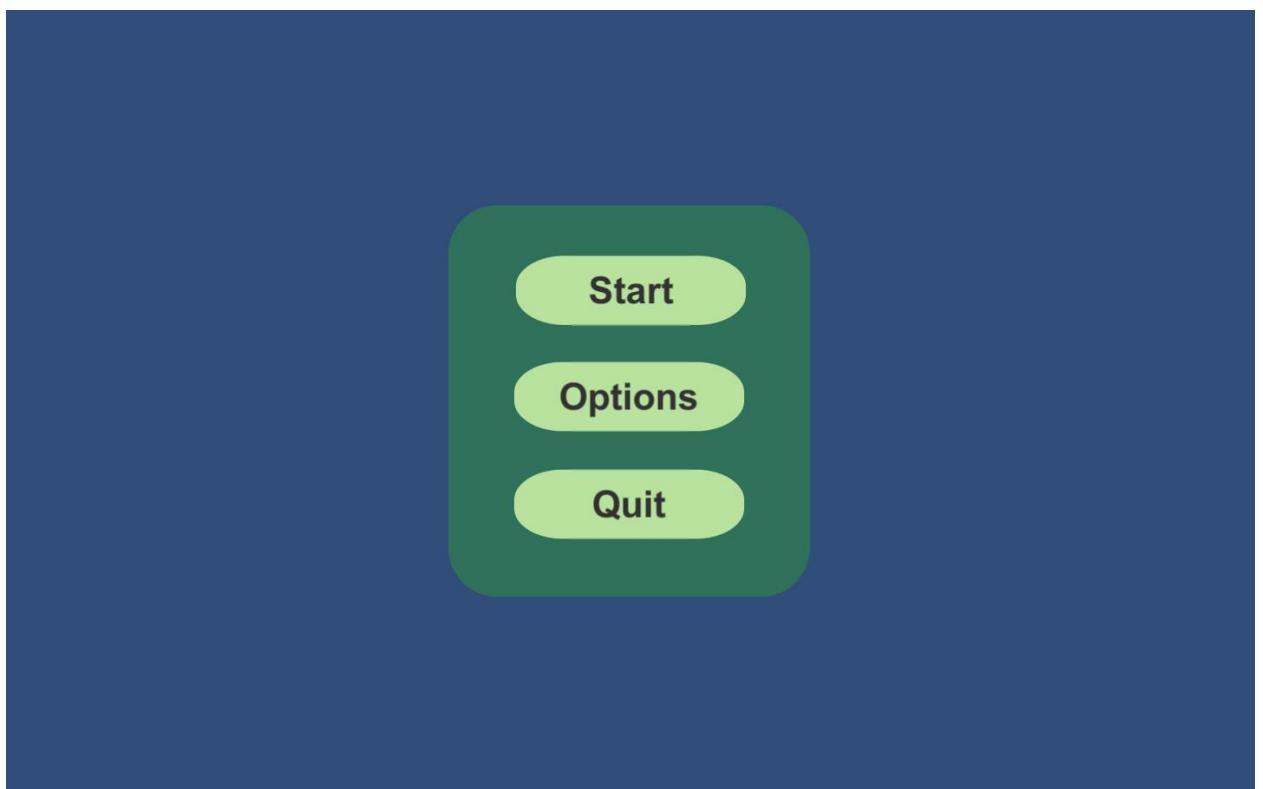


Рисунок 122 — Пример главного меню при запущенном приложении.

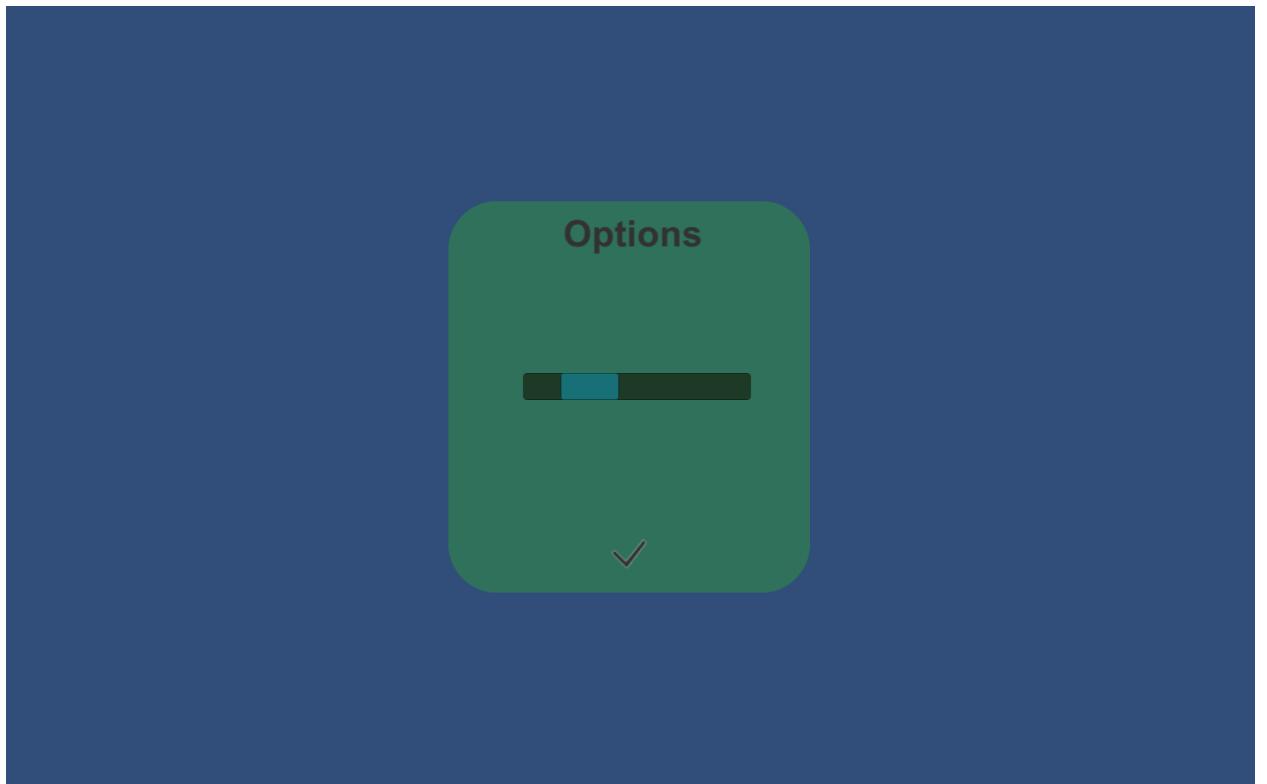


Рисунок 123 — Пример окна опций при запущенном приложении
Результат работы продемонстрировать преподавателю.

Практическая работа 13

Цель практической работы: создать first person controller в среде Unity.

Задачи:

- Написать скрипт для возможности двигать камерой;
- Написать скрипт для передвижения, падения и прыжка first person controller'a;
- Отрегулировать значения под оптимальные;
- Продемонстрировать преподавателю.

Описание выполнения работы

В данной практической работе, нужно будет написать свой FPC контроллер.

Для начала создаем Plane. После этого создаем пустой объект и добавляем на него компонент Character Controller (Рисунок 1).

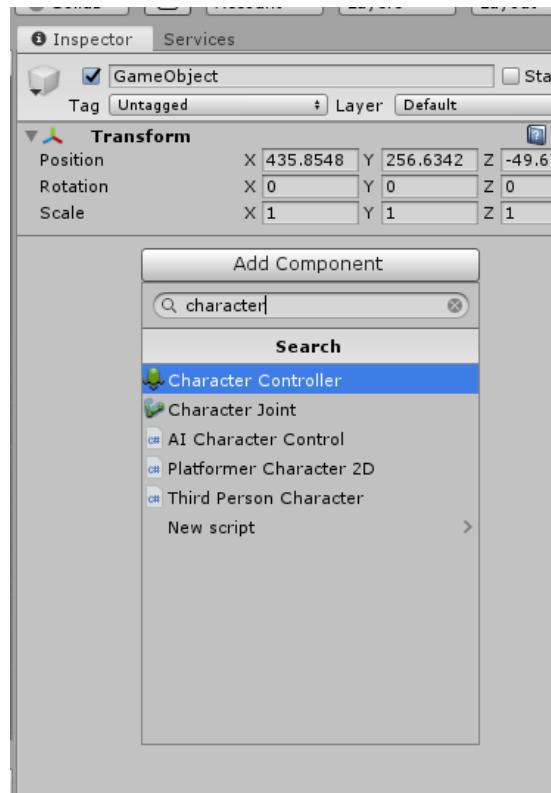


Рисунок 124 — Компонент Character Controller.

Чтобы границы Character Controller отображались на сцене, необходимо включить гизмос, нажав на соответствующую кнопку (Рисунок 2).

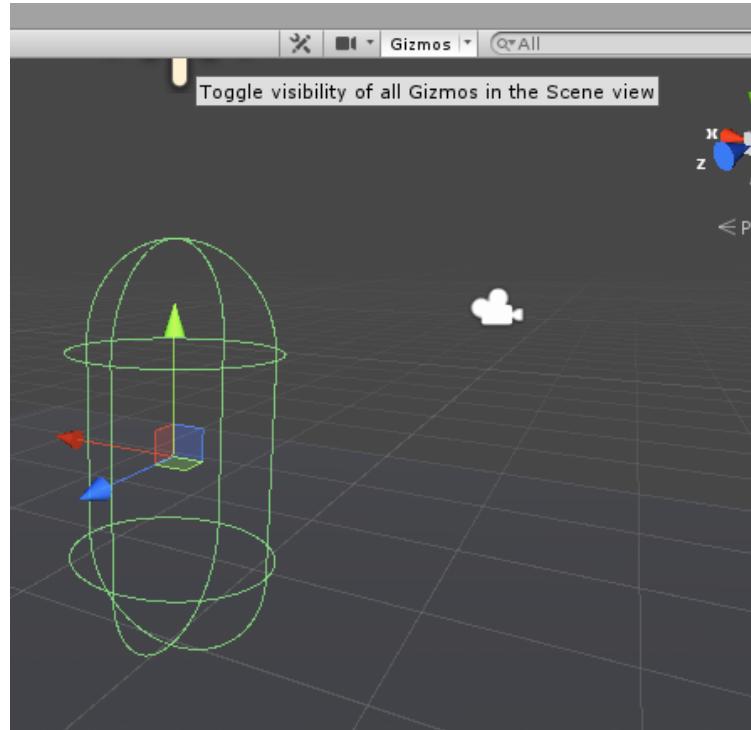


Рисунок 125 — Включенный Gizmos.

Создаем цилиндр дочерним объектом к объекту с Character Controller и выставляем цилиндру параметр Scale на 1.2, 1.8, 1.2.

Во избежание не желательных коллизий уберем цилинду его компонент, отвечающий за эту функцию (Рисунок 3).

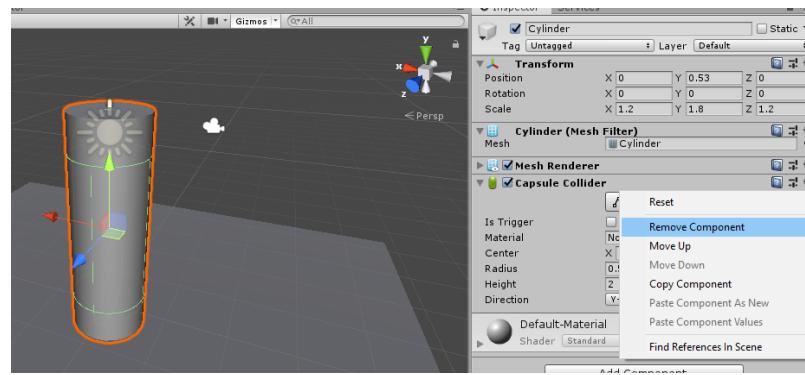


Рисунок 126 — Удаление компонента.

Теперь перетаскиваем мейн камеру в дочерние объекты пустого, туда же, где находится цилиндр и устанавливаем обзор на уровне глаз (Рисунок 4).

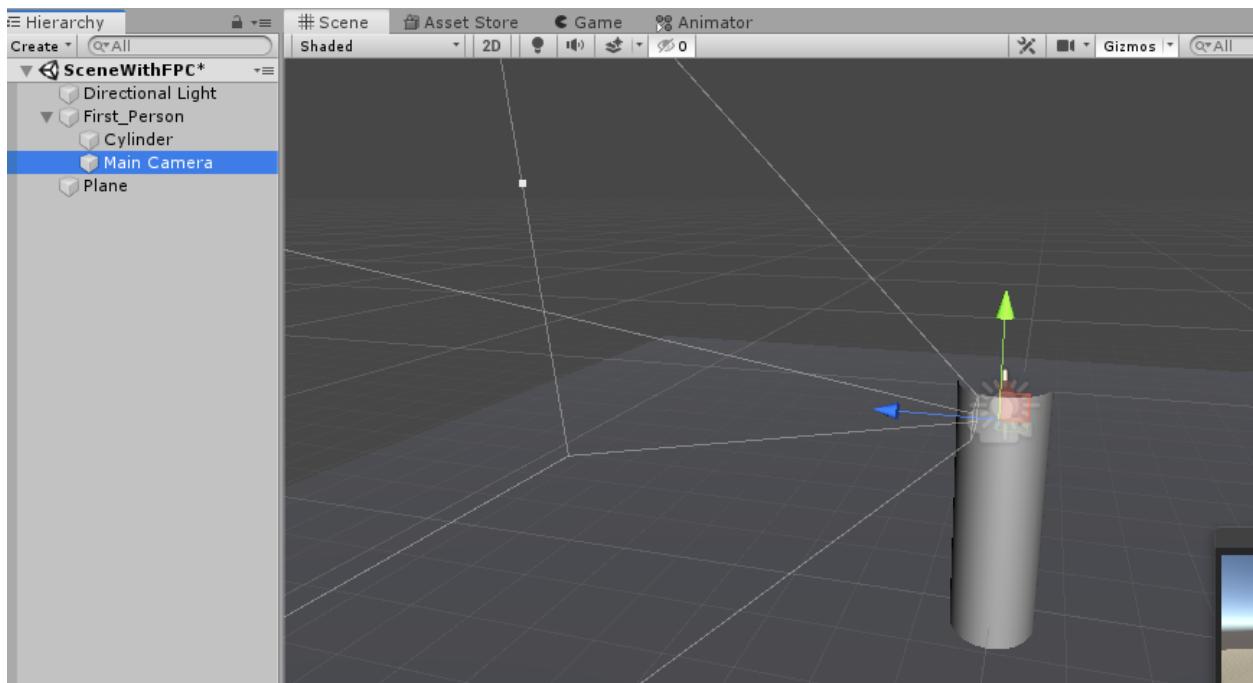


Рисунок 127 — Перемещение камеры в точку на уровне глаз.

Теперь создадим новый C# скрипт с названием MouseLook и повесим его на главную камеру. В Editor'е напишем код, который будет регулировать поведение камеры с помощью поворота курсора (Рисунок 5).

```
public class MouseLook : MonoBehaviour
{
    //Коэффициент чувствительности
    public float mouseSensitivity = 100f;

    //Сам объект FPS
    public Transform Controller;

    //Переменная, в которую будет записываться постоянно меняющееся значение поворота
    float xRotarion = 0f;

    void Start()
    {
        //Блокировка курсора
        Cursor.lockState = CursorLockMode.Locked;
    }

    void Update()
    {
        //Ввод по оси X и оси Y
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

        //Ограничения угла обзора сверху и снизу, по 90 градусов каждое
        xRotarion -= mouseY;
        xRotarion = Mathf.Clamp(xRotarion, -90f, 90f);

        transform.localRotation = Quaternion.Euler(xRotarion, 0f, 0f);
        //Поворот камеры по оси X
        Controller.Rotate(Vector3.up * mouseX);
    }
}
```

Рисунок 128 — Скрипт для поворота камеры мышью.

После сохранения, у скрипта в редакторе Unity следует добавить ссылку на основной контроллер и подрегулировать чувствительность мыши через значение на наиболее оптимальное (Рисунок 6).

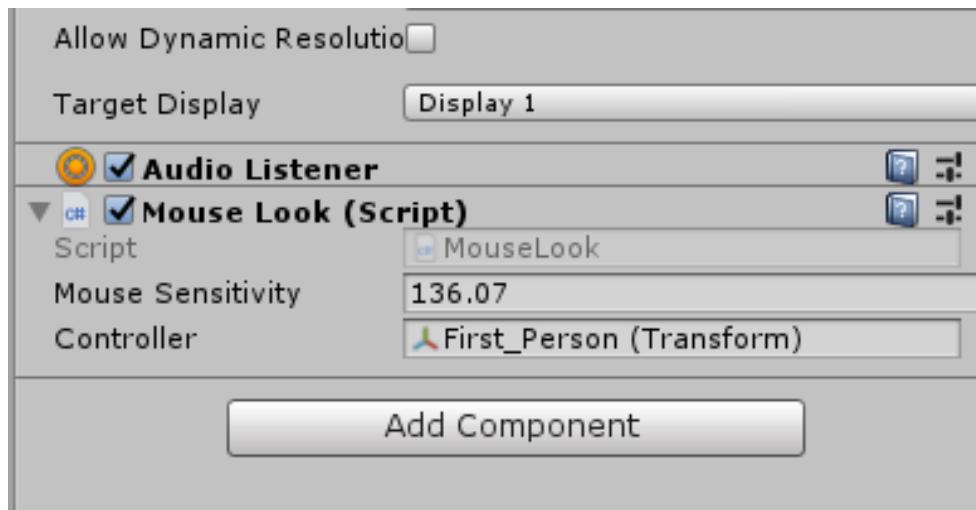


Рисунок 129 — Значения переменных в Inspector'е скрипта Mouse Look.

Запускаем сцену и проверяем работоспособность.

Теперь необходимо добавить движение на стандартные клавиши AWSD. В Unity для этого уже предусмотрены инструменты, поэтому это не составит труда (Рисунок 7).

```
public class Movement : MonoBehaviour
{
    //Переменная со ссылкой на Character Controller
    public CharacterController characterController;

    //Параметр для изменения скорости передвижения
    public float speed = 12f;

    void Update()
    {
        //Установка распознавания ввода сигналов с клавиш A,W,S,D
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");

        //Логика изменения координат объекта, в зависимости от того,
        //какие координаты вводятся (x или z).
        //      (forward)
        //      z
        //-(left)-x   x(right)
        //      -z
        //      (backward)
        Vector3 move = transform.right * x + transform.forward * z;

        //Вызов метода Move() на СС с формулой для передвижения
        characterController.Move(move * speed * Time.deltaTime);
    }
}
```

Рисунок 130 — Скрипт Movement. Передвижение объекта.

Скрипт вешаем на основной объект передвижения, на котором находится CharacterController, переносим этот компонент на переменную в Inspector'е и задаем удобное значение скорости (Рисунок 8).

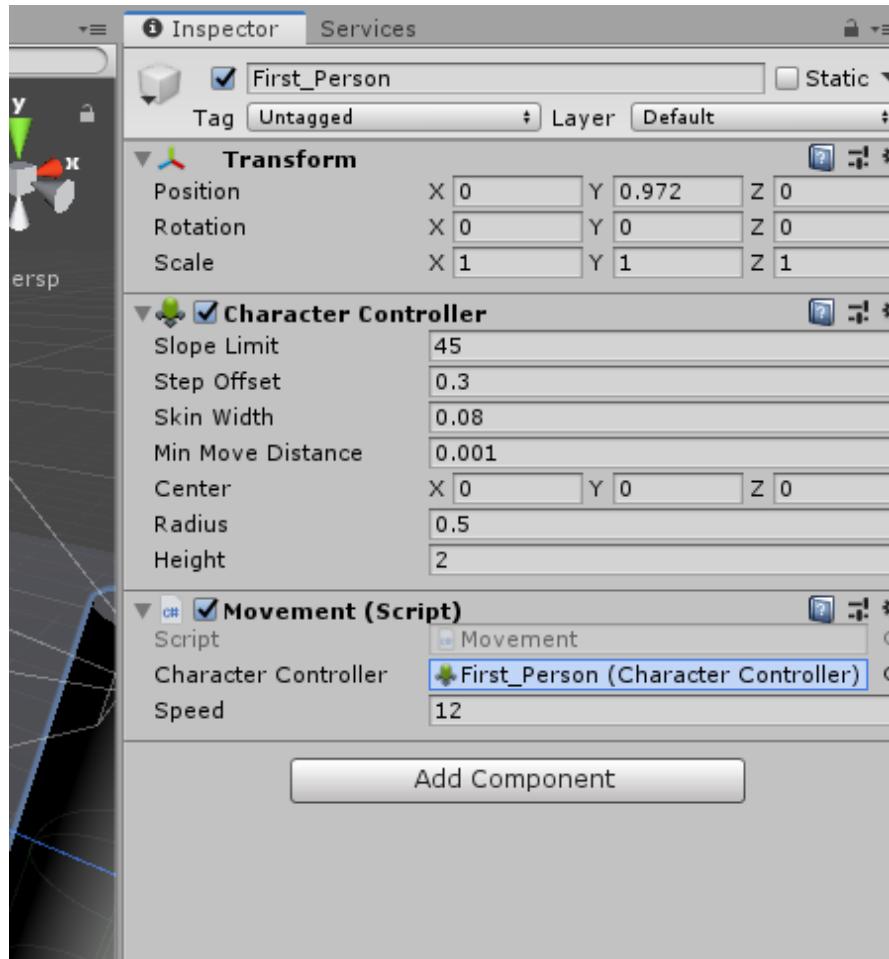


Рисунок 131 — Ссылка на компонент Character Controller.

Осталось добавить физику для объекта. Для того, чтобы Controller падал, когда у него нет твердой поверхности под ногами нужно изменять его скорость падения через вектор. Для измерения величины ускорения будет использована физическая формула $h = \frac{g*t^2}{2}$. Все новые строчки кода добавить в скрипт Movement (Рисунок 9).

```

public class Movement : MonoBehaviour
{
    [old prop]

    //Значение гравитационной постоянной (g) или просто гравитация
    public float gravity = -9.8f;

    //Переменная, в которой будет храниться значение при изменении скорости в падении
    Vector3 velocity;

    void Update()
    {
        [old]

        //Изменение координаты Y в векторе velocity
        velocity.y += gravity + Time.deltaTime;

        //Еще один вызов метода Move(), но уже для движения в падении
        //из логики формулы ускорения свободного падения h = (gt^2)/2
        characterController.Move((velocity * Time.deltaTime)/2);
    }
}

```

Рисунок 132 — Скрипт Movement. Гравитация и падение.

Если переместить объект выше и запустить сцену, то он упадет на твердую поверхность, как происходит с предметами в реальном мире, но машина не имеет данных о том, что предмет упал, и даже после коллизии будет и дальше считать значение по этой формуле, что приведет к огромной скорости при следующем падении. Для того, чтобы предотвратить это, необходимо добавить то, что будет проверять землю под ногами. Создаем пустой дочерний объект внутри Character Controller'a. Переименуем его в GroundCheck и переместим в нижнюю часть цилиндра (Рисунок 10).

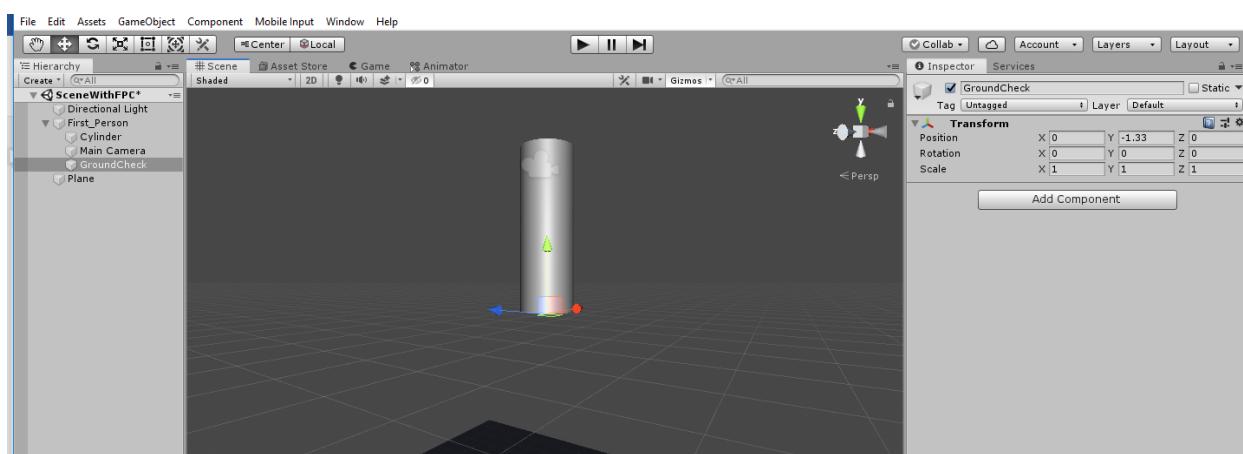


Рисунок 133 — Установка GroundCheck.

Теперь перейдем к скрипту и продолжим код в скрипте Movement (Рисунок 11).

```

public class Movement : MonoBehaviour
{
    old prop

    public Transform GroundCheck;

    //Растояние до земли, на котором будет триггериться невидимая сфера
    public float groundDistance = 0.4f;

    //Маска для распознавания земли
    public LayerMask groundMask;

    //Переменная, которая изменит свое значение на true после приземления CC на ground
    bool isGrounded;

    void Update()
    {
        old

        //Переменная становится истиной, если невидимая сфера, созданная в координатах GroundCheck.position
        //с радиусом groundDistance, соприкосается со слоем groundMask
        isGrounded = Physics.CheckSphere(GroundCheck.position, groundDistance, groundMask);

        //Проверка истинности выражения
        ////(значение -2f берется только из-за лучшего эффекта
        ////значение 0f тоже допустимо)
        if (isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }
    }
}

```

Рисунок 134 — Скрипт Movement. Проверка на наличие слоя Ground на твердой поверхности.

Вернемся в Unity editor, поставим ссылку на объект GroundCheck и добавим слой Ground через вкладку Layers (Рисунок 12, 13).

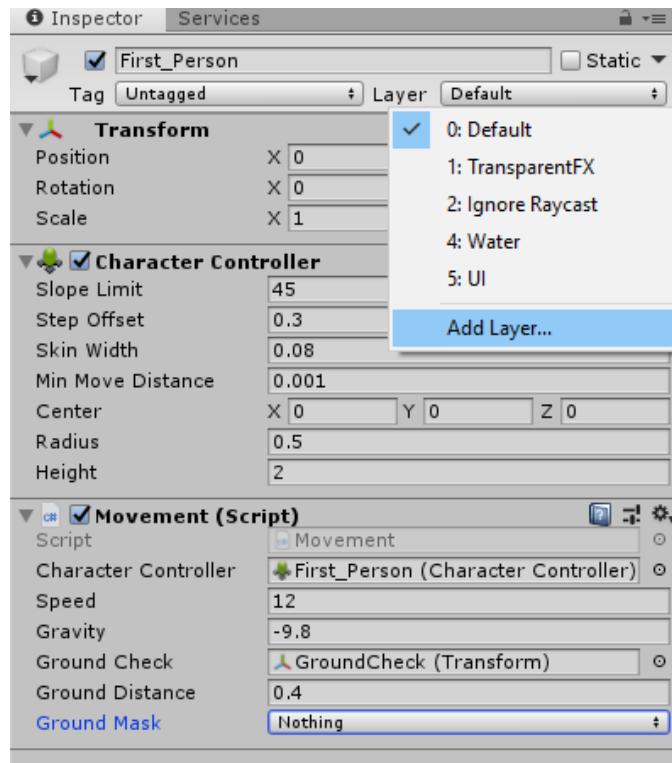


Рисунок 135 — Добавление нового слоя.

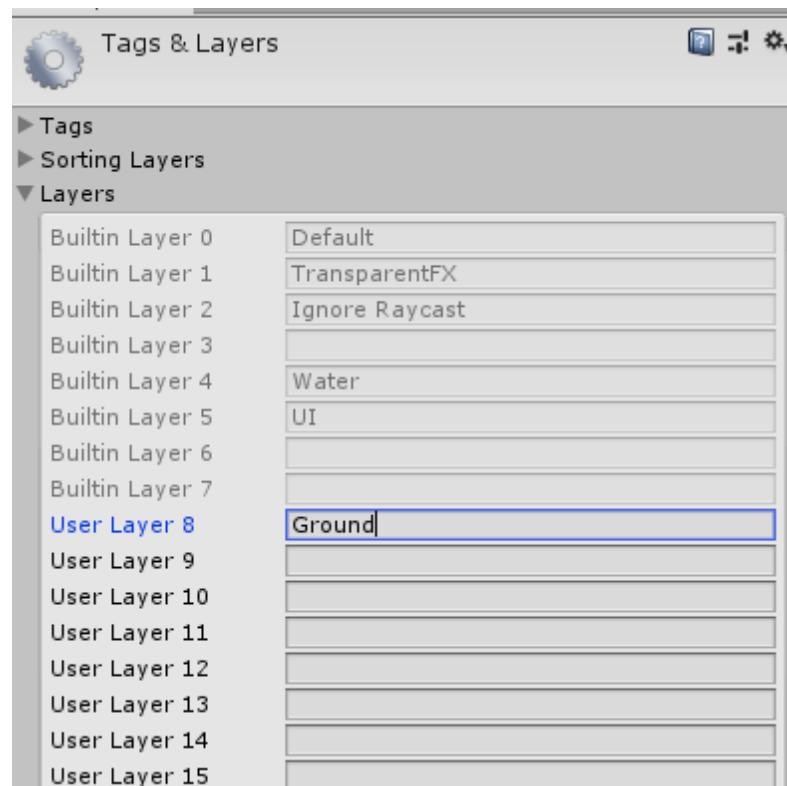


Рисунок 136 — Добавление слоя Ground.

Вернемся к Character Controller'у и в Inspector'е укажим новый слой в переменной Ground Mask (Рисунок 14).

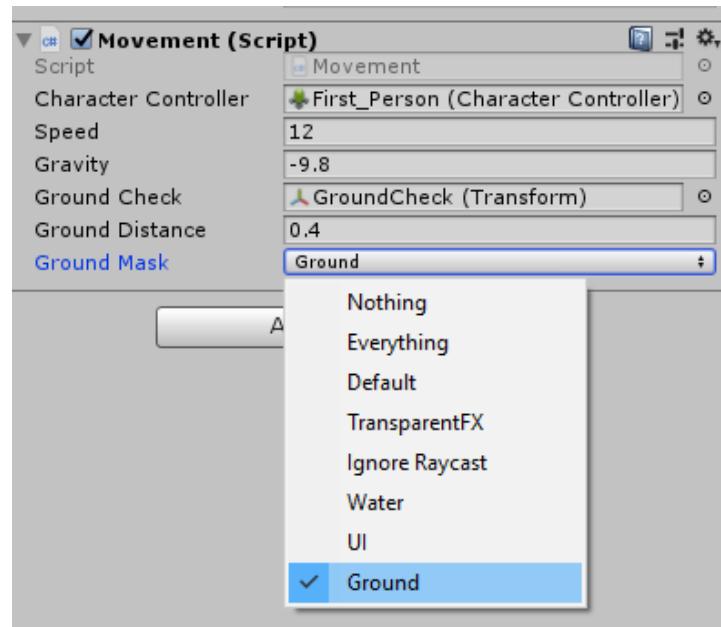


Рисунок 137 — Выбор нового слоя на стрипте.

У твердой поверхности, на которую падает Character Controller, необходимо изменить слой на Ground (Рисунок 15).

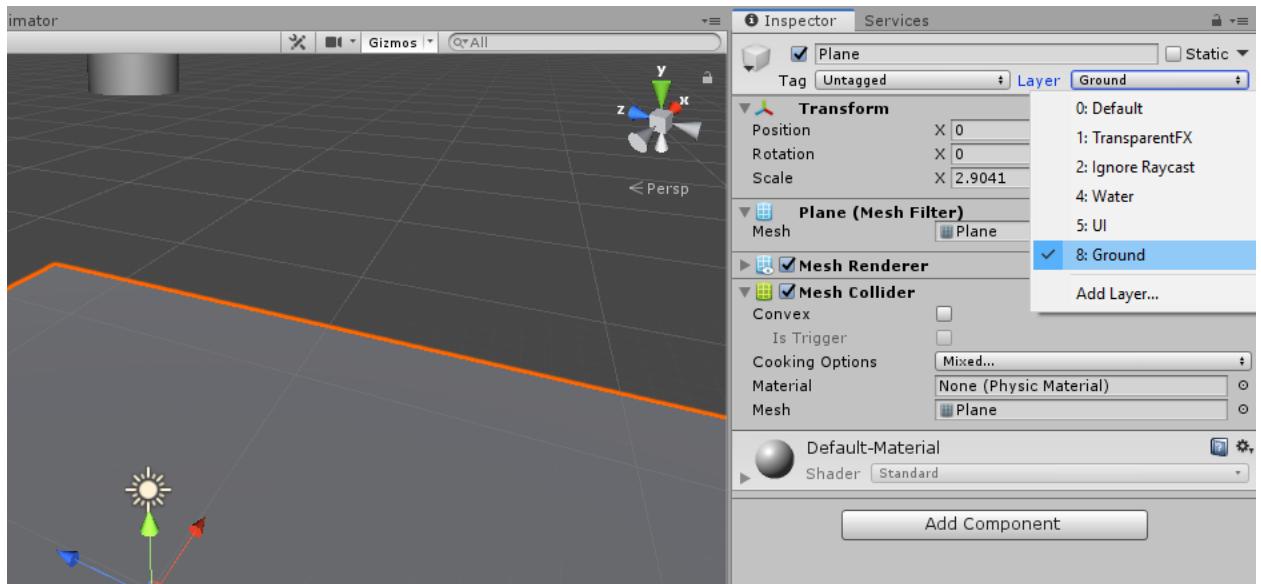


Рисунок 138 — Добавление слоя к твердым телам

Вернемся к скрипту и допишем код, при котором Character Controller будет прыгать (Рисунок 16).

```
public class Movement : MonoBehaviour
{
    old prop

    //Переменная силы прыжка
    public float jumpForce = 1000;

    void Update()
    {
        old

        //При нажатии клавиши Scape по умолчанию, СС подпрыгивает
        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpForce * -2f * gravity);
        }
    }
}
```

Рисунок 139 — Скрипт Movement. Прыжок.

Последнее, что осталось сделать, это создать ступеньки и возвышенности. Лестницу можно создать при помощи небольших кубов и добавить им слой Ground. Через Character Controller можно сделать автоматическое передвижение по лестнице через параметр Step Offset, здесь задается максимальная высота ступенек. Помимо этого, чтобы все выглядело органично, силу прыжка можно уменьшить, а гравитацию увеличить до -2 (Рисунок 17).

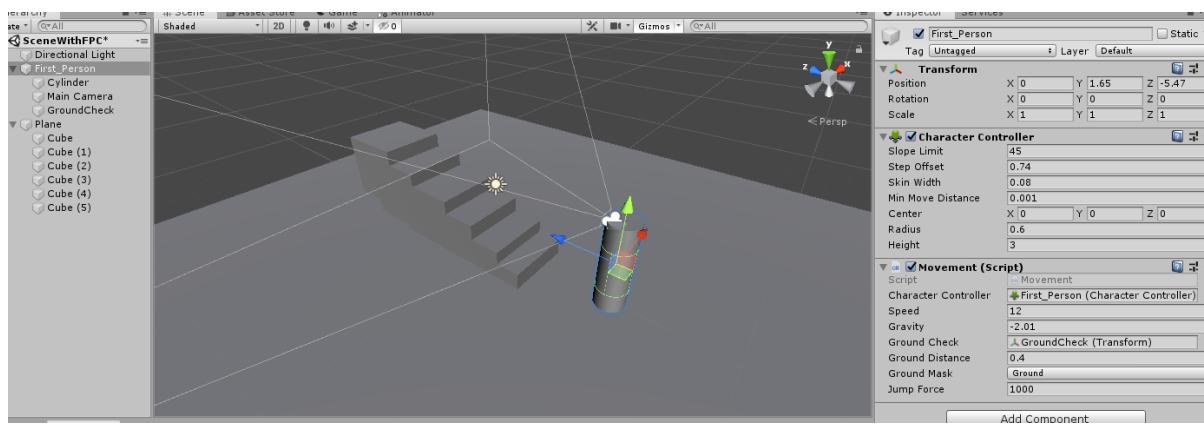


Рисунок 140 — Пример готовой сцены.

Результат протестировать, отрегулировать переменные и показать преподавателю.

Практическая работа 14

Цель практической работы: сборка VR-проекта.

Задачи:

- Настройка сцены;
- Настройка VR камеры и контролера движения FibrumSDK;
- Создание и настройка GUI элементов;
- Написание скрипта функционала GUI;
- Продемонстрировать преподавателю.

Описание выполнения работы

Создаем новую сцену, куда добавляем объект класса Plane и удаляем основную камеру.

Теперь необходимо установить FibrumSDK package. Для этого необходимо скачать соответствующий prefab и импортировать как package в Unity (Рисунок 1).

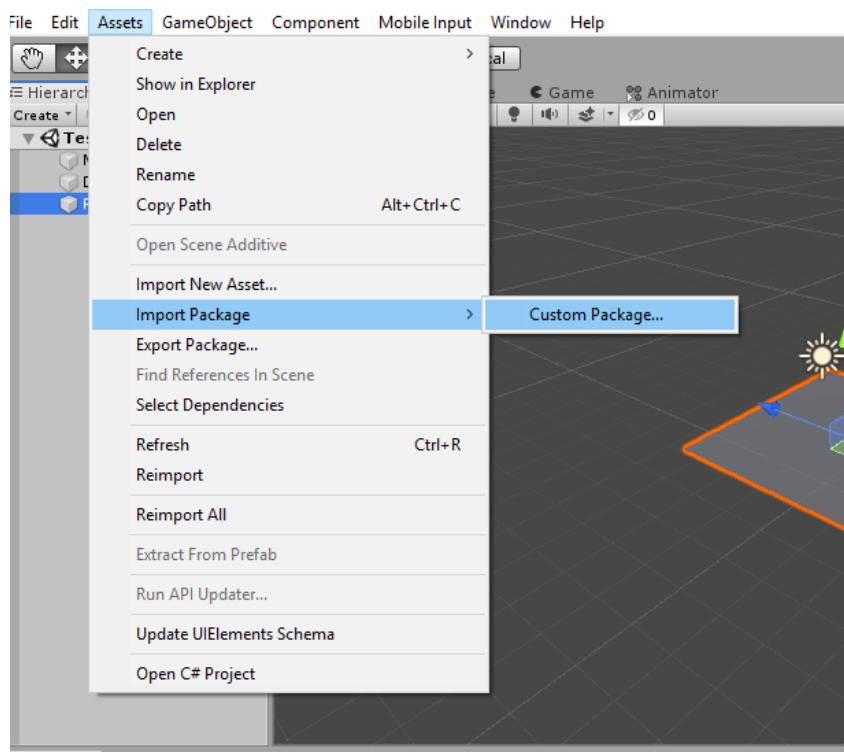


Рисунок 141 — Импорт Package’а.

Из prefabов этого пака, на сцену перетаскиваем Joystick_Simple_FPS_character. В дочерних объектах к

Joystick_Simple_FPS_character отключаем VR_Camera, а в настройках самого объекта в параметре Bullet_Prefab (Рисунок 2).

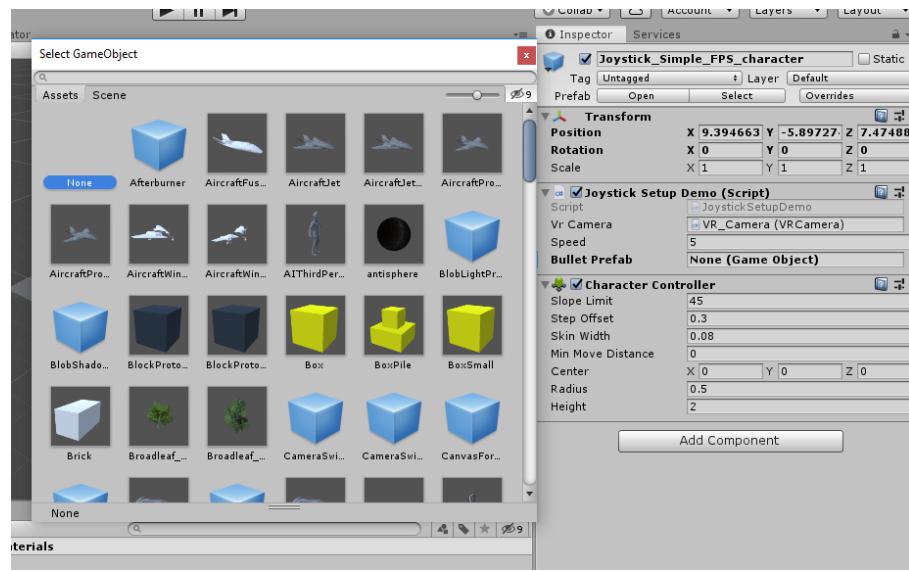


Рисунок 2 — Удаление объекта из параметров JoyStick'a.

Следом, из папки с prefabов перетаскиваем на сцену обычный объект VR_Camera и делаем его дочерним к Joystick_Simple_FPS_character'y.

Запустим сцену и протестируем работоспособность.

Создадим канвас и в его параметрах укажем Render Mode – World Scape (Рисунок 3).

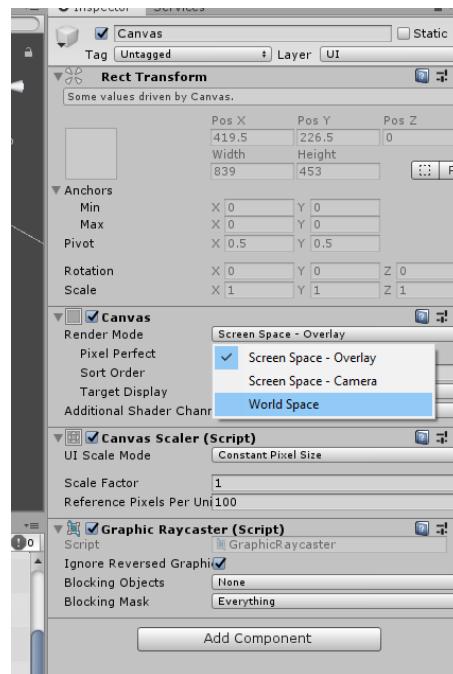


Рисунок 3 — Смена Render Mode на World Space.

Следом, создаем на этом канвасе Panel и три кнопки, две из них будут создавать разные объекты, а последняя будет выходить из приложения (Рисунок 4). После создания канваса нужно его расположить так, чтобы его координаты были приблизительно координатам Plane, а также размер канваса хорошо воспринимался пользователем.

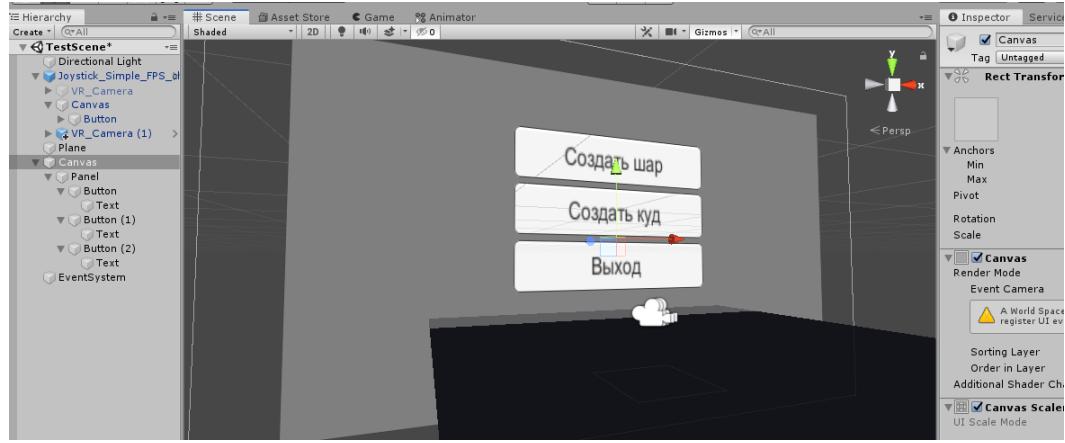


Рисунок 4 —Пример меню.

Напишем новый C#, в котором пропишем функционал для созданных ранее кнопок (Рисунок 5).

```
public class MenuButtonsScript : MonoBehaviour
{
    public GameObject BoxPrefab;
    public GameObject SpherePrefab;
    public GameObject PanelMenu;
    public GameObject SpawnObject;
    bool isActive;

    void Update()
    {
        if (Input.GetKeyUp(KeyCode.Escape))
        {
            if (!isActive)
            {
                PanelMenu.SetActive(true);
                isActive = true;
            }
            else
            {
                PanelMenu.SetActive(false);
                isActive = false;
            }
        }
    }

    public void CreateBox()
    {
        Instantiate(BoxPrefab, SpawnObject.transform.position, SpawnObject.transform.rotation);
    }

    public void CreateSphere()
    {
        Instantiate(SpherePrefab, SpawnObject.transform.position, SpawnObject.transform.rotation);
    }

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

Рисунок 5 — Код функционала для меню.

Сохраняем и переходим в UnityEditor. Создаем Шар и Куб и переносим их в Ассеты, чтобы сохранить их prefabs. Создаем пустой объект, который назовем Spawn и помешаем его в место, где будут появляться объекты. Сам скрипт помещаем на FPS объект и назначаем ему все параметры (Рисунок 6).

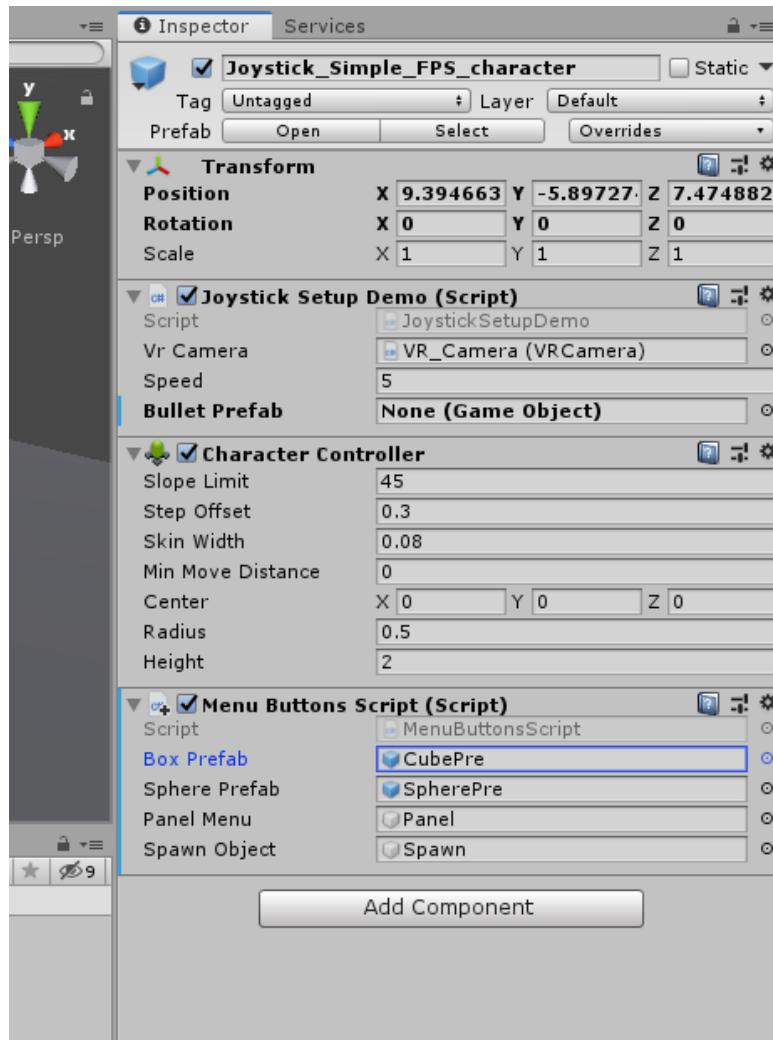


Рисунок 6 — Параметры созданного скрипта.

На кнопках в функции OnClick() назначаем необходимые функции и запускаем сцену, проверяя работоспособность. На кнопку Esc панель должна скрываться и появляться вновь при повторном нажатии. А при нажатии на разные кнопки на панели создаваться разные объекты (Рисунок 7).

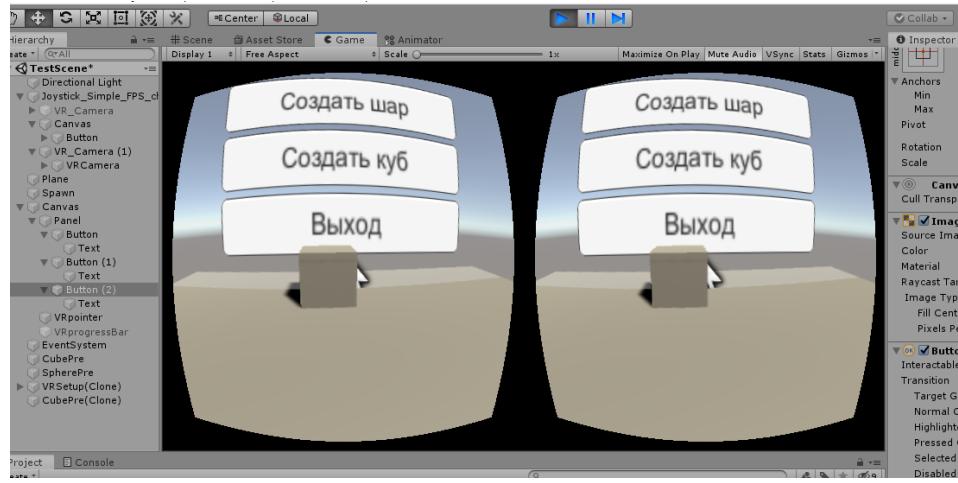


Рисунок 7 — Пример работы сцены.

Осталось создать билд. Для этого зайдем во вкладку файл и откроем Build Settings. Здесь добавляем сцену, которую создали, и выбираем Target Platform – Windows. Нажимаем кнопку Build и указываем кореную папку для приложения.

После завершения обработки, нужно зайти в кореную папку сборки и открыть .exe файл. Результат продемонстрировать преподавателю.

Практическая работа 15

Цель практической работы: изучение сетевой системы Unity.

Задачи:

- Создать интерфейс для чата
- Написать скрипты для чата
- Настроить чат
- Собрать проект.
- Настроить подключение Host

Описание выполнения работы

Первым делом необходимо создать интерфейс для чата, для этого создадим объект Canvas и добавим на него объект panel.

Далее на объект panel добавим UI объекты Text, Button и InputField и разместим их на панели. (Рис.1)

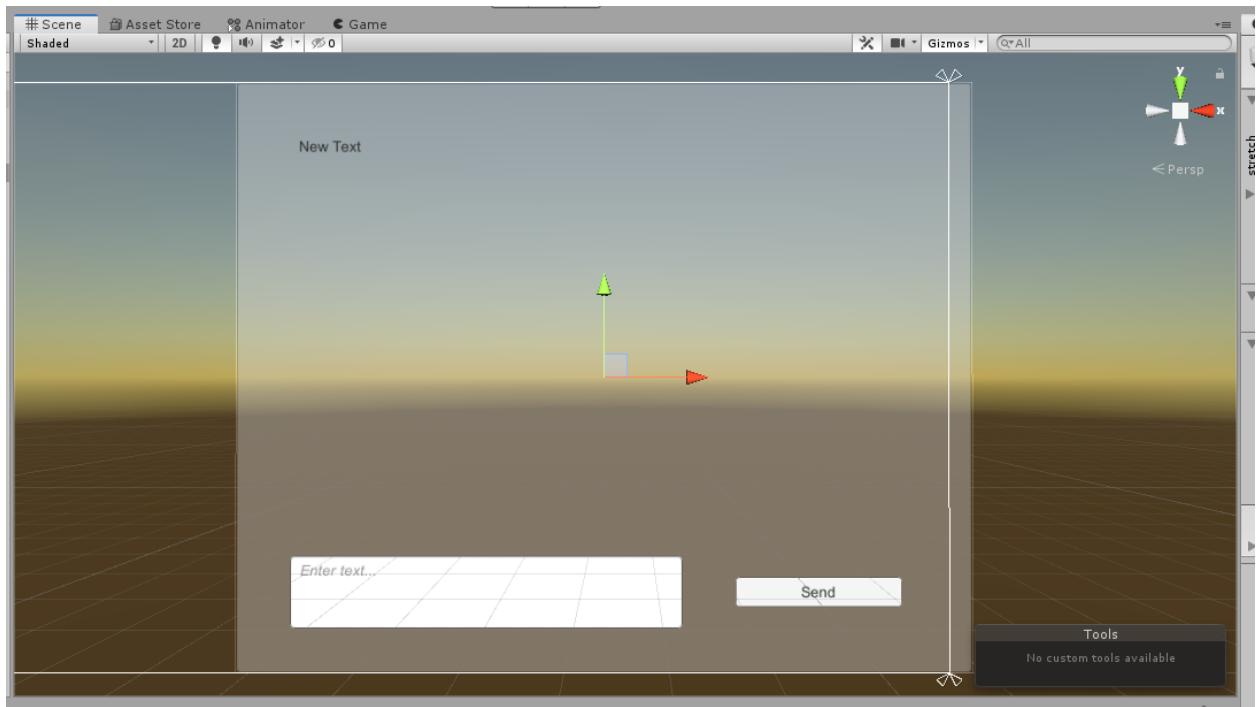


Рисунок 1 - Интерфейс чата

Затем необходимо подключить новый пакет multiplayer HLAPI для этого в меню Window> Package Manager необходимо нажать на «плюс» и добавить необходимый пакет.

Создадим пустой объект Network и добавим на него компонент NetworkManger, сохраним сцену и перенесем её в настройки компонента offline scene и online scene (Рис.2)

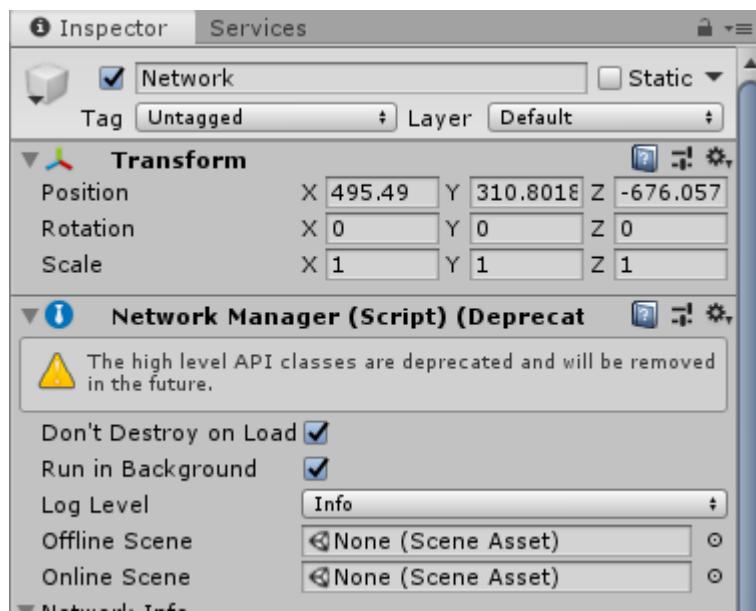


Рисунок - 2 Настройки Network Manager

Затем в подменю Spawn Info необходимо добавить player prefab, для этого создадим пустой объект, сделаем из него prefab и перенесем его в это меню (Рис.3)

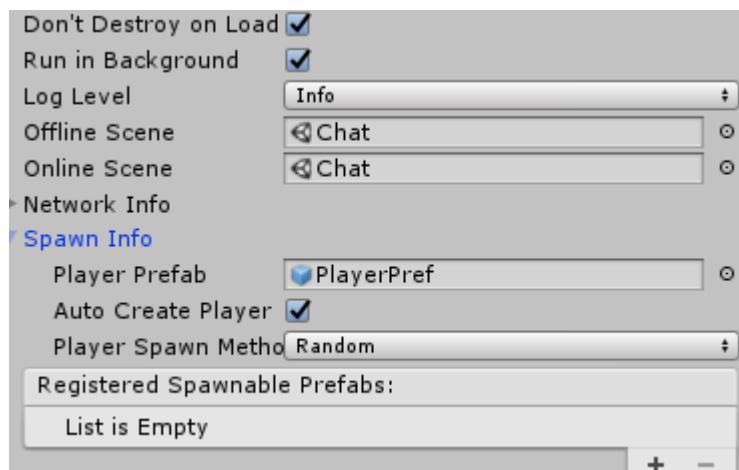
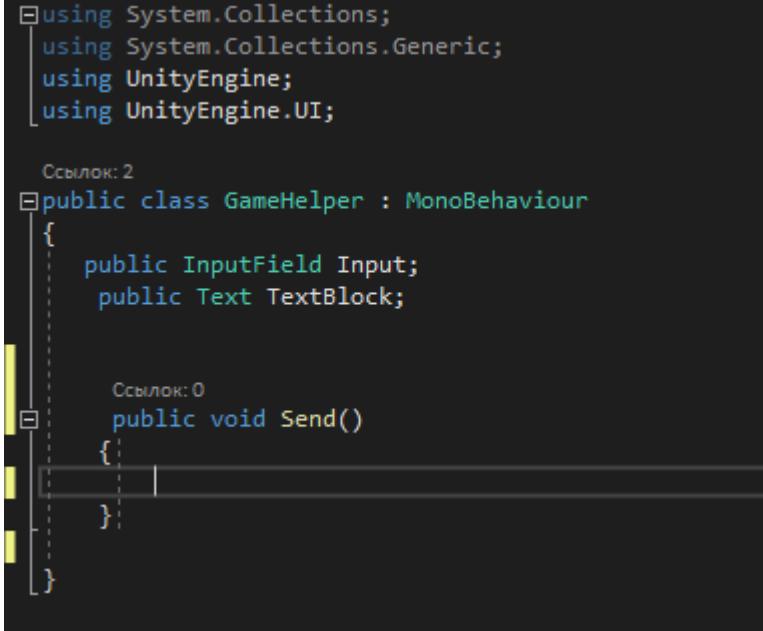


Рисунок 3 - Player Prefab

Также необходимо добавить компонент Network Manager HUD.

Далее идет настройка скриптов, для начала создадим 2 скрипта PlayerHelper GameHelper и еще один пустой объект GameHelper.

Сначала в GameHelper напишем следующий код (Рис.4)



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

Ссылок: 2
public class GameHelper : MonoBehaviour
{
    public InputField Input;
    public Text TextBlock;

    Ссылок: 0
    public void Send()
    {
    }
}
```

Рисунок 4 - GameHelper

Добавим этот скрипт на пустой объект GameHelper и в его настройках перенесем все ссылки на игровые объекты InputField, Text, а в настройках Button в настройках on Click () выведем ссылку на объект GameHelper GameHelper> Send (рисунки 5 и 6)

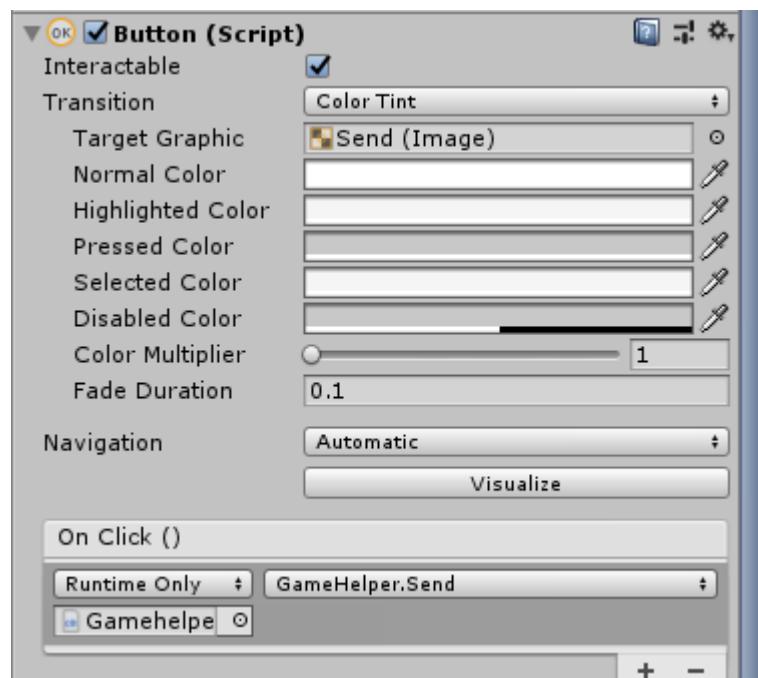


Рисунок 5 Настройка Button

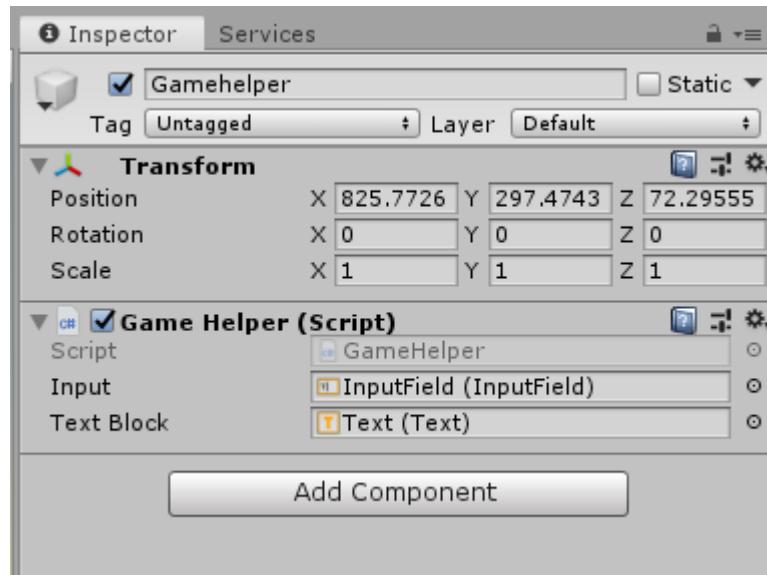


Рисунок – 6 Настройка GameHelper

Далее необходимо дописать скрипты (рисунки 7, 8)

Результат продемонстрировать преподавателю.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

Ссылок: 2
public class PlayerHelper : NetworkBehaviour
{
    GameHelper __gameHelper;
    Ссылок: 0
    void Start()
    {
        __gameHelper = GameObject.FindObjectOfType<GameHelper>();
        if (isLocalPlayer)
        {
            __gameHelper.CurrentPlayer = this;
        }
    }

    [Command]
    ссылка: 1
    public void CmdSend(string message)
    {
        Debug.Log("Send =" + message);
        RpcSend(message);
    }
    [ClientRpc]
    ссылка: 1
    public void RpcSend(string message)
    {
        __gameHelper.TextBlock.text += System.Environment.NewLine + message;
    }
    Ссылок: 0
    void Update()
    {
    }
}

```

Рисунок 7 - Скрипт PlayerHelper

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

Ссылок: 2
public class GameHelper : MonoBehaviour
{
    public InputField Input;
    public Text TextBlock;

    private PlayerHelper __currentPlayer;
    Ссылок: 2
    public PlayerHelper CurrentPlayer
    {
        get { return __currentPlayer; }
        set { __currentPlayer = value; }
    }

    Ссылок: 0
    public void Send()
    {
        CurrentPlayer.CmdSend(Input.text);
    }
}

```

Рисунок 8 - Скрипт GameHelper

Далее требуется собрать проект и запустить его в двух экземплярах – один через exe файл, а второй через саму программу Unity. В одном из них в HUD окне Network необходимо нажать Lan Host, а на втором Lan Client, написать что-либо и отправить. Сообщение должно появиться в общем чате в двух открытых проектах (рисунки 9, 10).

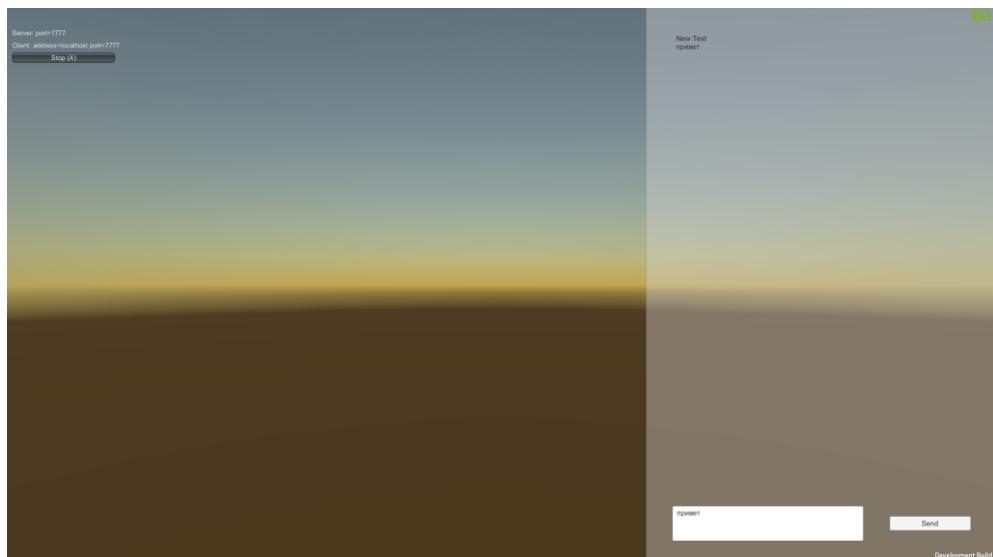


Рисунок 9 - Сообщение в первом открытом окне

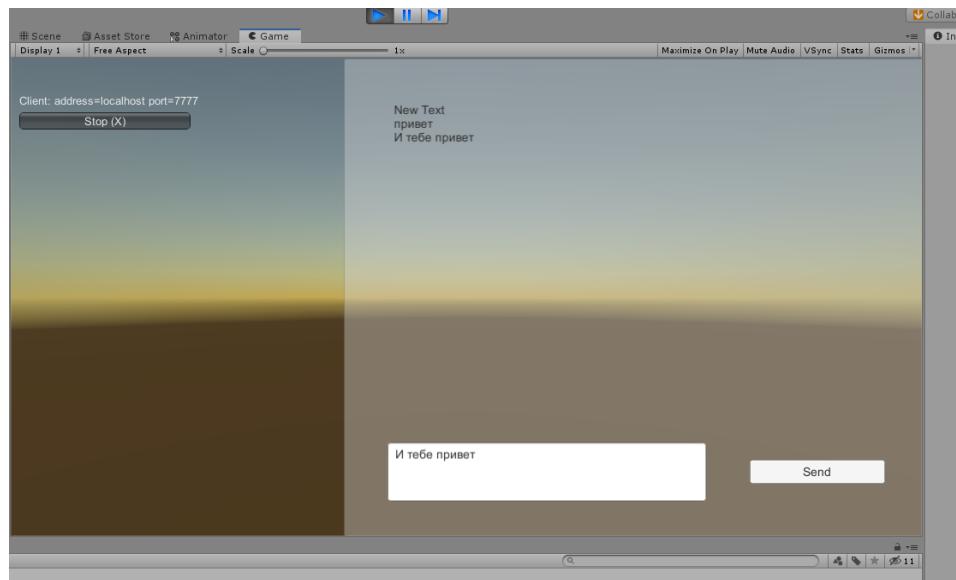


Рисунок 10 - Сообщение во втором открытом окне

Теперь необходимо создать персонажа, за которого пользователи будут пользователи ходить, для этого понадобится скрипты из практической работы номер 13. Создаем капсулу, в нее помещаем камеру и присоединяем скрипты. Помимо них понадобится еще два компонента NetWork Transform и NetWork Identity (рисунок 11).

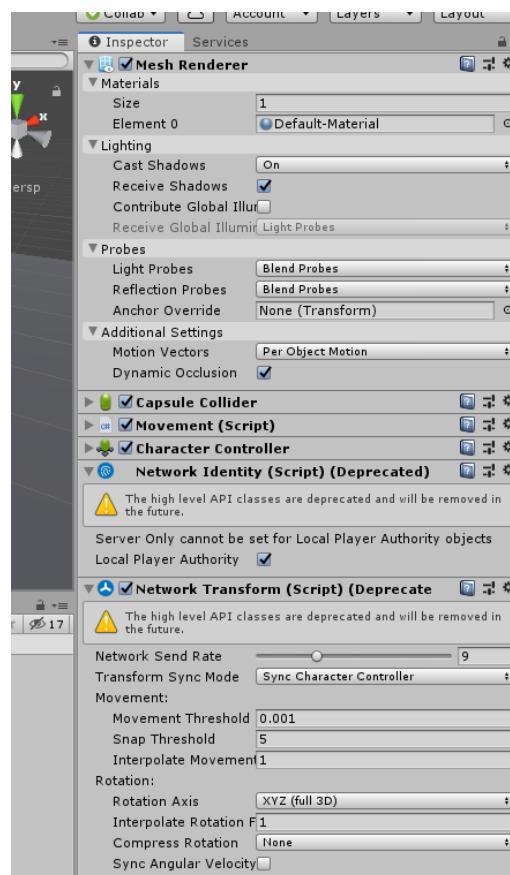


Рисунок 11 - Компоненты персонажа.

Здесь нужно поставить галочку на параметре Local Player Authority.

Вернемся в подменю SpawnInfo и поставим prefab плеера (Рис. 12).

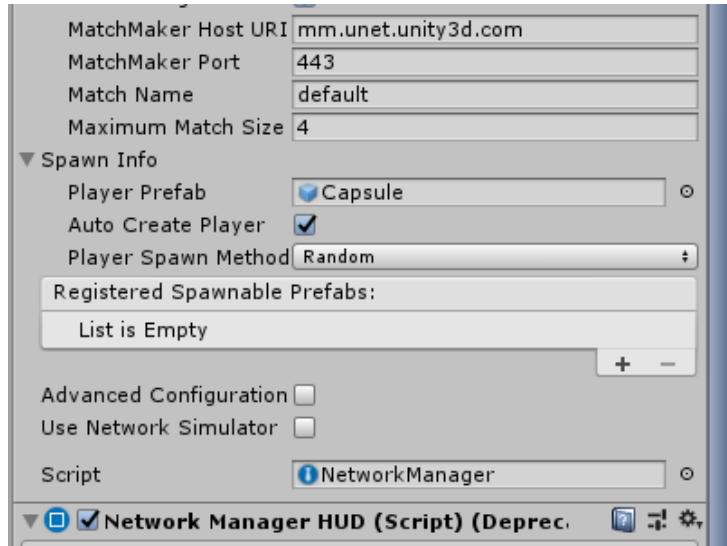


Рисунок 12 - Добавление prefaba пользователя

Создадим плейн, а в нем два пустых объекта: Spawn1 и Spawn2, здесь пользователи будут появляться, чтобы обозначить их как метки для появления добавим компонент Network Start Position (Рис. 13).

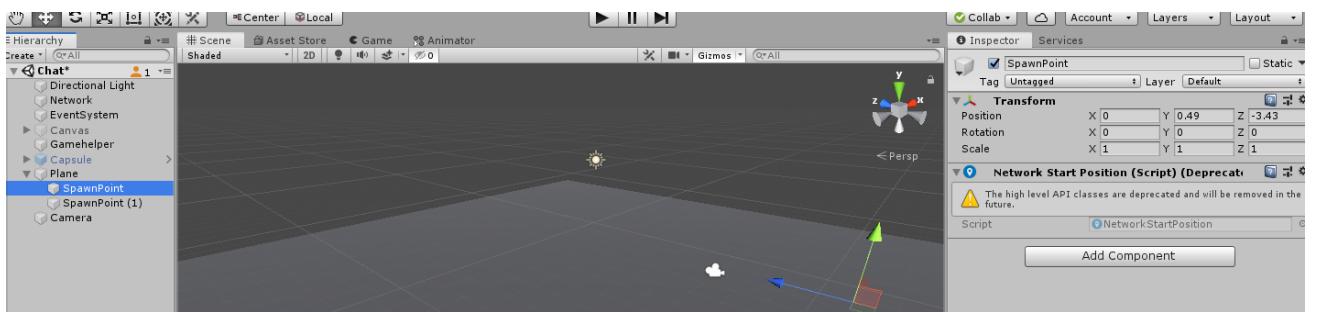


Рисунок 13 - Добавление компонента NSP к объекту Spawn

Теперь перейдем к скрипту Movement, здесь добавим следующие записи (Рис. 14).

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class Movement : NetworkBehaviour
{
    [SerializeField] float oldProp;

    private void FixedUpdate()
    {
        if (!isLocalPlayer)
            return;

        oldProp = old;
    }

    public override void OnStartLocalPlayer()
    {
        GetComponent<Material>().color = Color.red;
        base.OnStartLocalPlayer();
    }
}

```

Рисунок 14 - Изменения в скрипте Movement

Все, теперь сцена готова к работе. Для того, чтобы создать клиента, можно собрать приложение и из него зайти на сервер хоста.

Создаем сборку проекта (Рис. 15).

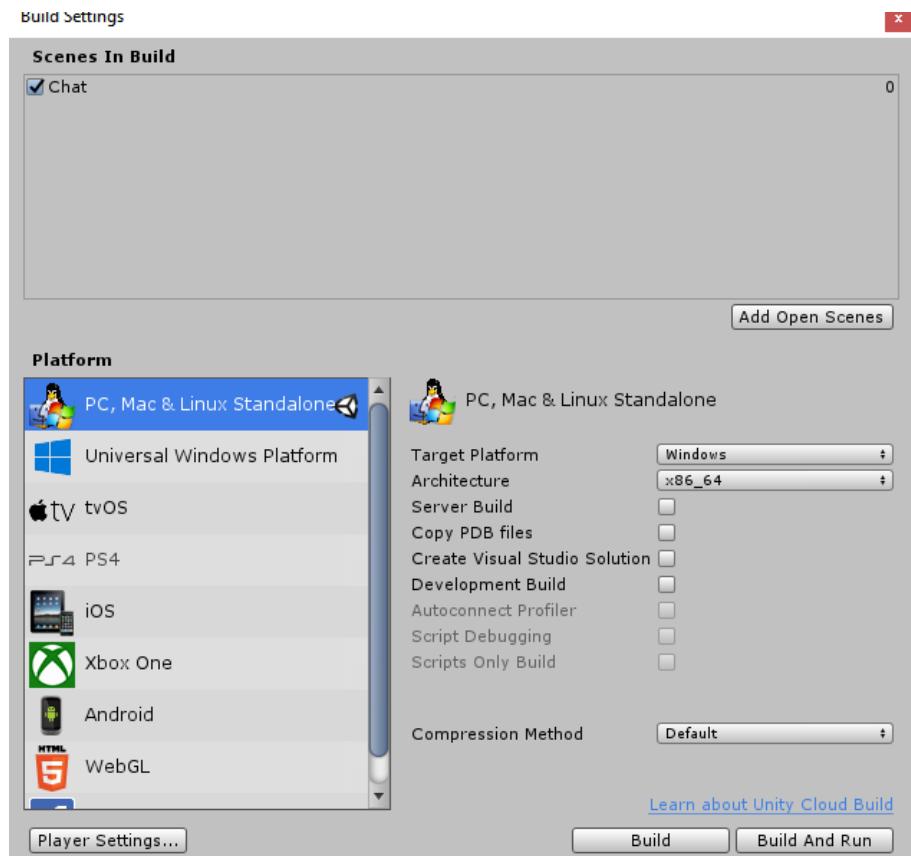


Рисунок 15 - Сборка проекта

Запускаем сцену в эдиторе и нажимаем на кнопку LAN Host (Рис. 16)

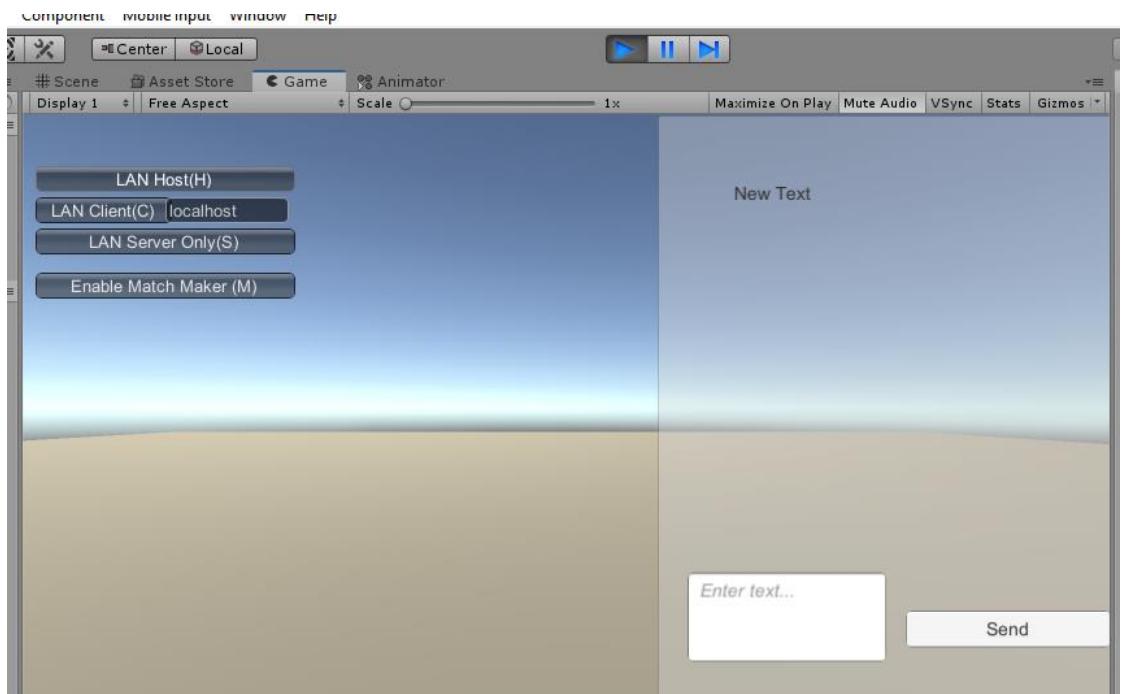


Рисунок 16 - Включенная сцена в эдиторе и создание хоста

В углу появился порт и адрес сервера. Теперь переходим в приложение, открыв его, нажимаем кнопку LAN Client и присоединяемся к серверу (Рис. 17).

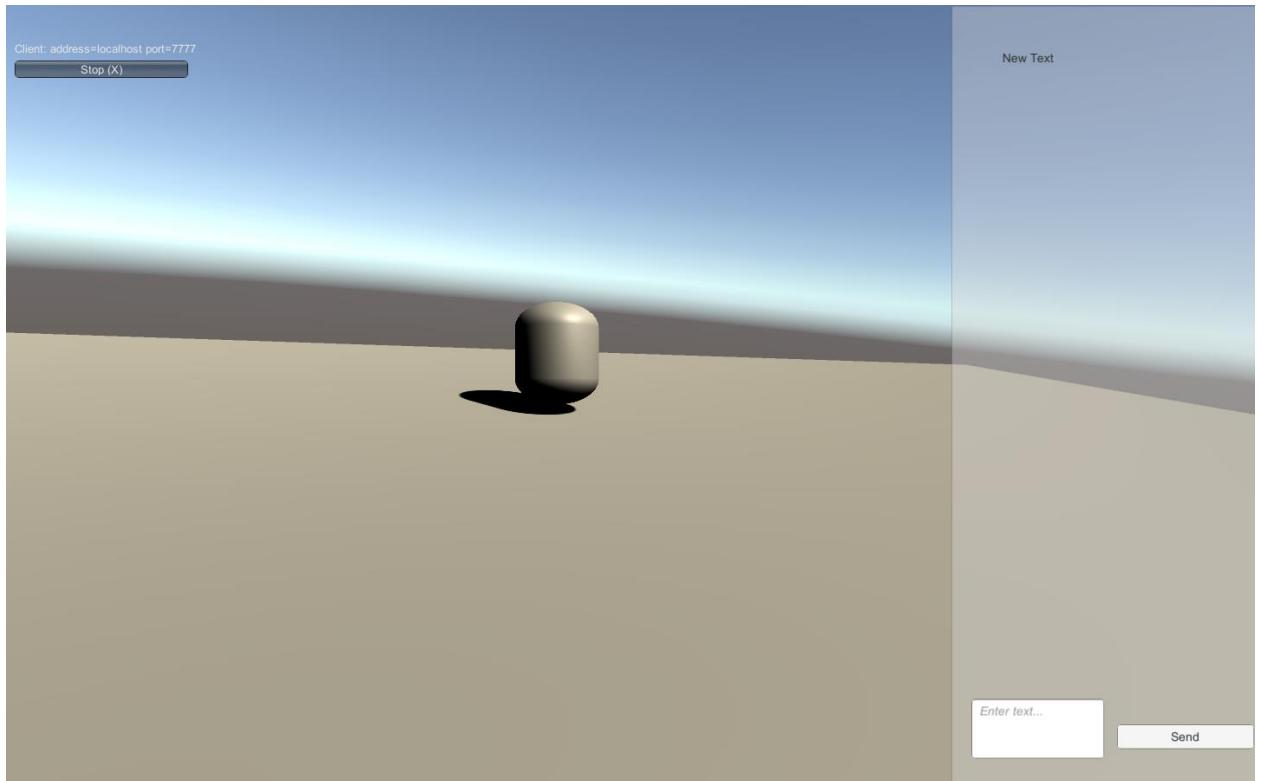


Рисунок 17 - Включенное приложение и подключение как клиент

Результат работы показать преподавателю.

Практическая работа 16

Цель практической работы: Создания цифрового архитектурного двойника, перенос его в виртуальную среду разработки Unity с последующей настройкой и подключением к сети.

Задачи:

- Создать двойник архитектурного сооружения
- Конвертация и перенос проекта, в среду виртуальной разработки Unity
- Наложение текстур
- Настройка коллаборации проекта
- Настройка освещения и постобработки

Описание выполнения работы

Данная практическая работа рассчитана на выполнение по выдаваемому варианту, соответственно для разбора построения 3D объекта был использован чертеж с номером варианта по рисунку - Рисунок 1.

План 2 этажа М 1:100

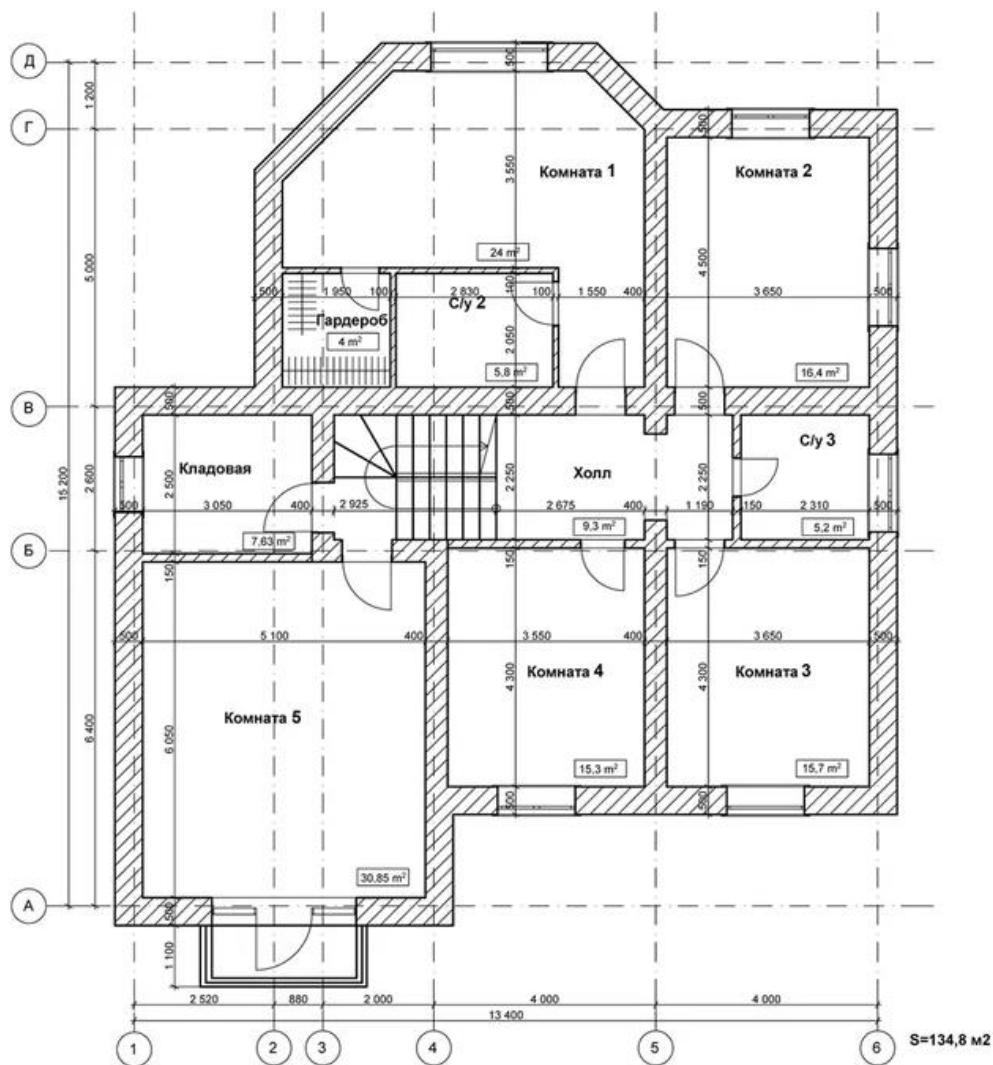


Рисунок 1 —Чертеж дома

Из-за того, что это план второго этажа, была проведена перепланировка, для создания первого этажа этого же здания. В ходе такой перепланировки была убрана лестница в холле, и балкон был заменен крыльцом.

Материал, на который был наложен чертеж, был применен к примитиву Plane и закреплен для невозможности случайного смещения объекта.

Для точности построения модели были подкорректированы размеры с помощью специального метода сравнения размеров, был создан объект Box и

его размер корректировался в соответствии со значением длины стены на чертеже Рисунок 2.

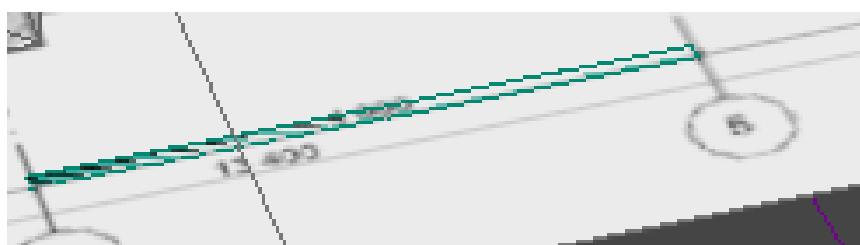


Рисунок 2 —Настройка размеров объекта и чертежа

Для точности в построении, лучше всего использовать spline, очерчивать стены по чертежу, конвертировать в изменяемый полигон и применять к объекту различные модификаторы.

При помощи инструментов Spline по чертежу был очерчен контур будущего здания. После этого, тем же инструментом были очерчены стены здания. Для редактирования 3D объекта и последующего добавления различных модификаторов, созданные Spline ы конвертированы в редактируемые полигоны (convert to editable poly). Был применен модификатор Extrude несколько раз в зависимости от положения окон и дверей, чтобы в последующем удалить полигоны в этих местах, для простого создания вырезов. Когда стены подняты на достаточное расстояние, был удален верхний полигон

Необходимо задать им объем, для этого хорошо подходит модификатор Shell. При применении данного модификатора для задан параметр Inner Scale в соответствии с границами стен на чертеже. Из-за того, что некоторые стены имеют разную толщину, они были разъединены и к ним был применен модификатор независимо от остальных элементов стен. Теперь, удалением передних и задних полигонов, можно создать отверстия для дверей и окон

Рисунок 3.

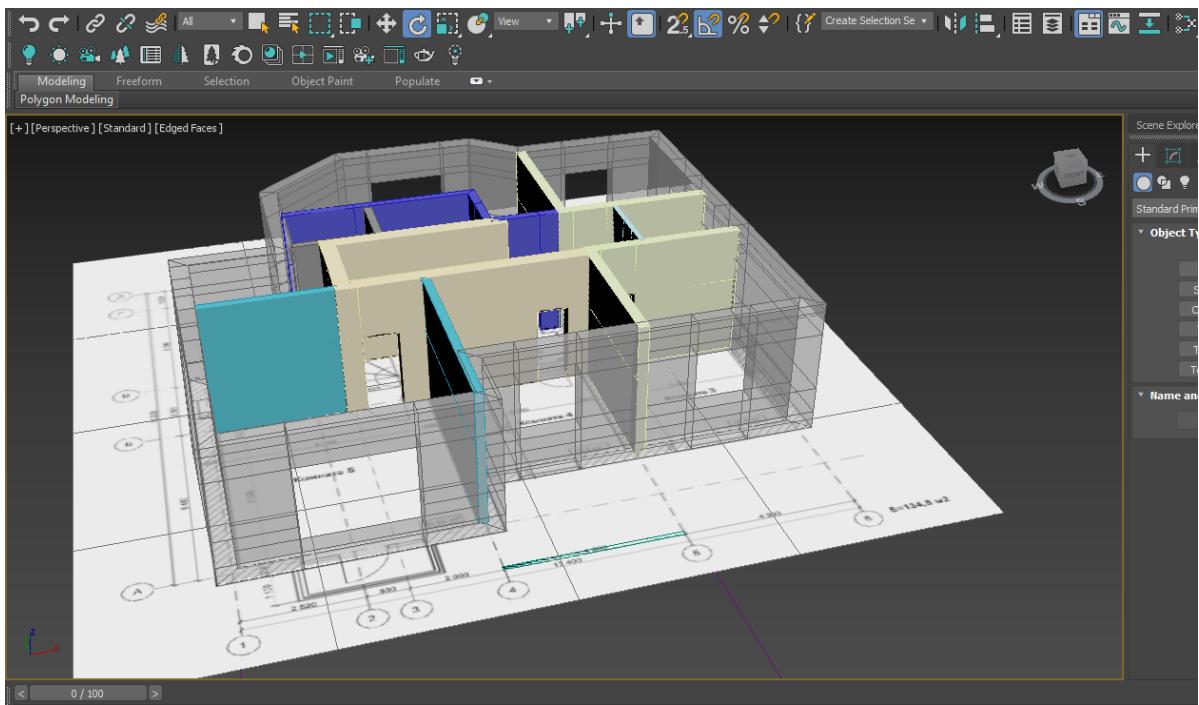


Рисунок 3—Модель дома с вырезанными отверстиями для дверей и окон

Окна созданы из примитивов 3ds Max, и разделены на элементы, с помощью функции detach с целью последующей простоты наложения материалов, тоже самое проделано с дверьми Рисунок 4.

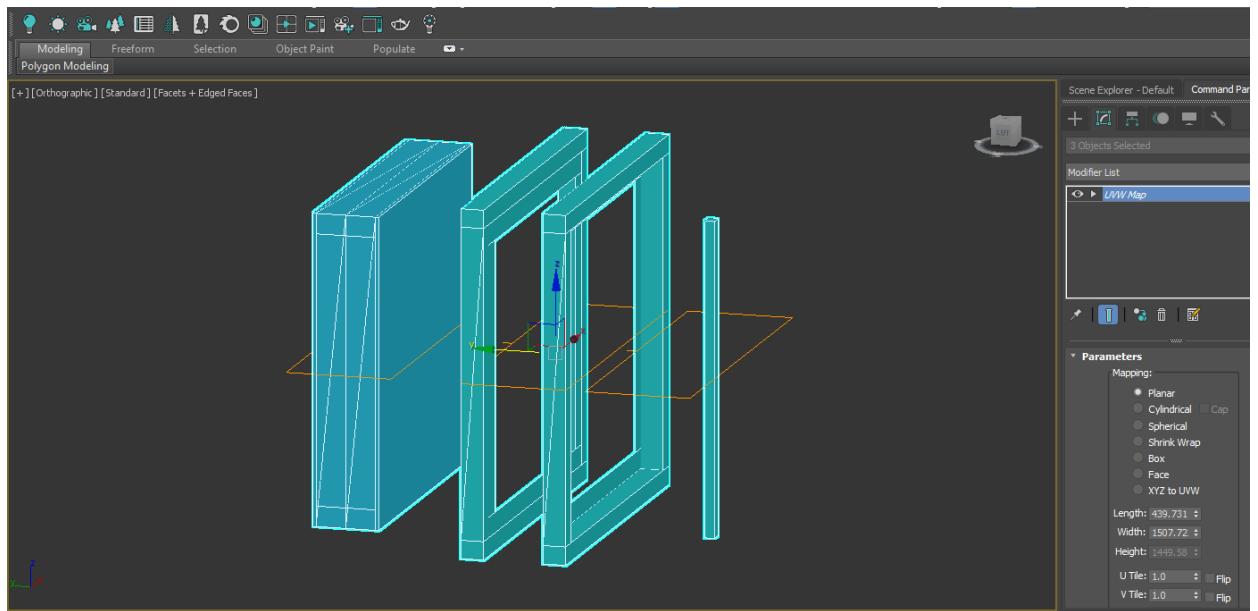


Рисунок 4—Создание модели окна

Все окна и двери были подстроены под размеры вырезов на чертеже (Рисунок 5).

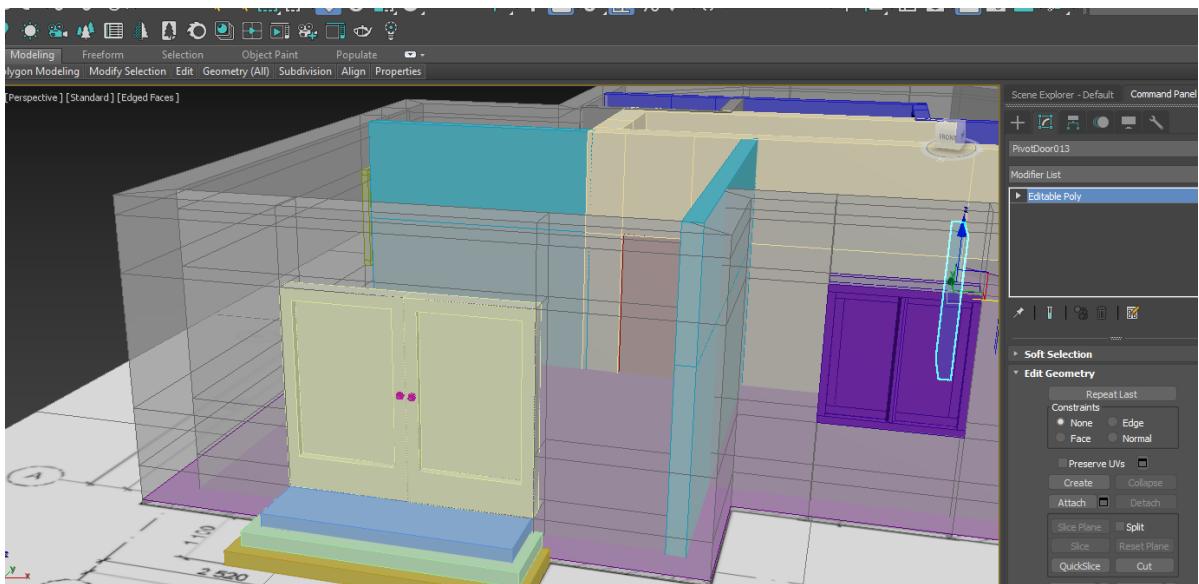


Рисунок 5 —Модель, с расставленными окнами и дверями

При помощи, сохраненного ранее контура здания, была создана подушка для крыши и немного увеличена в размерах, а после конвертации в полигональный объект подверглась применению модификатора Extrude.

Из копии полигонального объекта подушки получилась крыша, с помощью инструмента Cut были вырезаны линии, и, через смещение вершины на пересечении этих линий, получилась крыша (Рисунок 6).

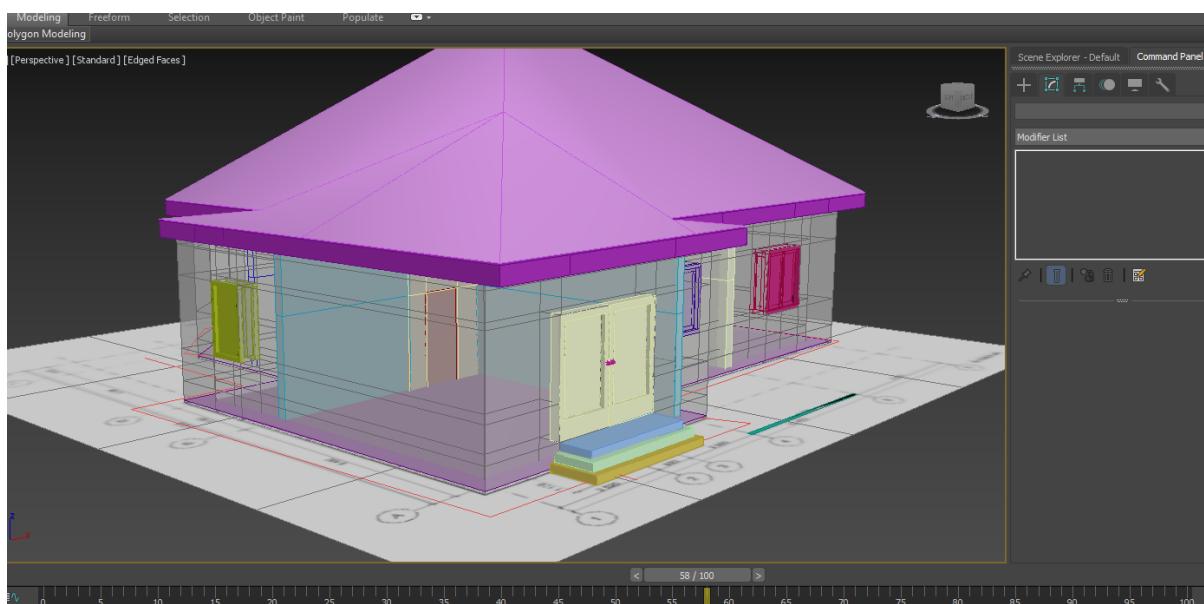


Рисунок 6 —Добавленная крыша к общей модели

На некоторые полигоны наложены разные материалы, чтобы при присвоении материалов в Unity, текстура не ложилась на объект полностью, а

только на определенную его часть. Например, на стенах внутри должны быть обои, а на стенах снаружи должна быть каменная кладка (Рисунок 7).

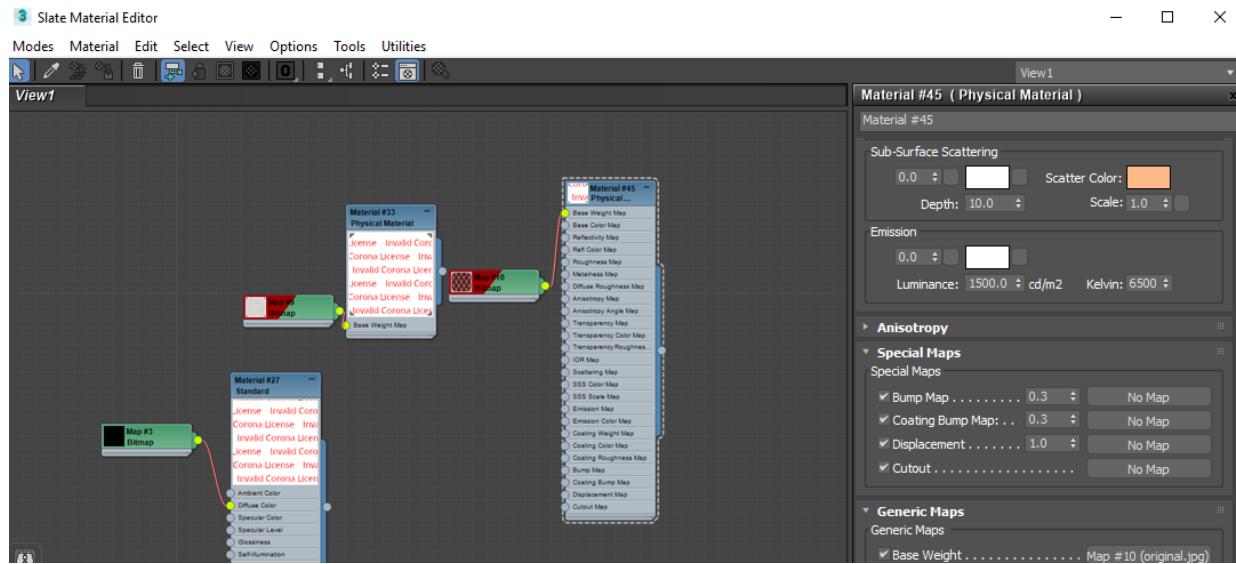


Рисунок 7 —Созданные материалы для наложения

После окончания работы с моделью, необходимо ее экспортовать в Unity, самый подходящий формат для экспорта — fbx.

После того, как файл модели сохранен в формате .fbx или .obj, его необходимо импортировать в unity. Сделать это можно несколькими способами:

- Перенести сохраненный файл в окно assets;
- Через панель инструментов на вкладке Assets нажать на функцию “Import New Asset” и выбрать нужный файл;
- Переместить сохраненный файл в папку проекта.

Теперь необходимо убедиться, что в инспекторе аксессуаров стоит галочка для импорта материалов, которые сохранены в данных модели (Рисунок 8).

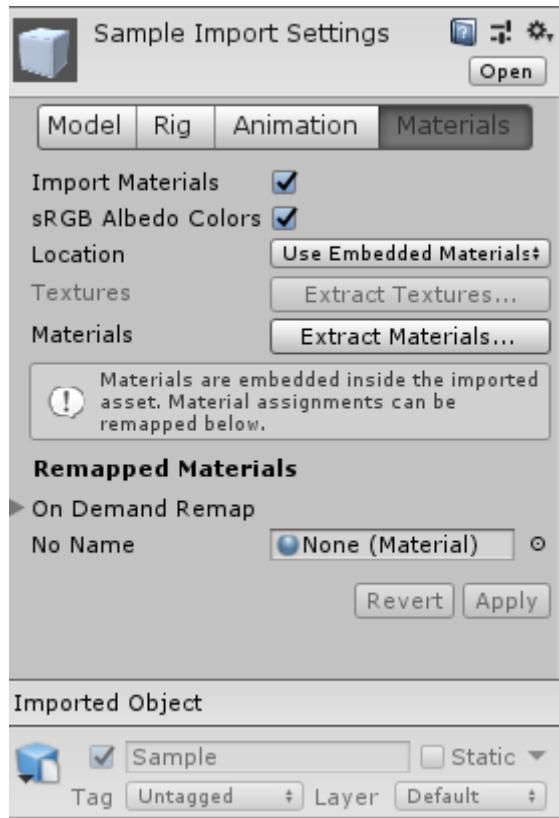


Рисунок 8 – Настройки импортированного файла

3D модель теперь можно добавить на сцену, перенеся prefab в иерархию объектов. Так как этот объект prefab, все изменения файла повлияют на объект на сцене. Если планируется, что в файл будут вноситься какие-то корректировки и пересохраняться обратно в Unity, то следует оставить его prefabом, если же необходимо создать несколько копий, которые будут существовать независимо от основного файла, тогда следует распаковать этот объект (Рисунок 9).

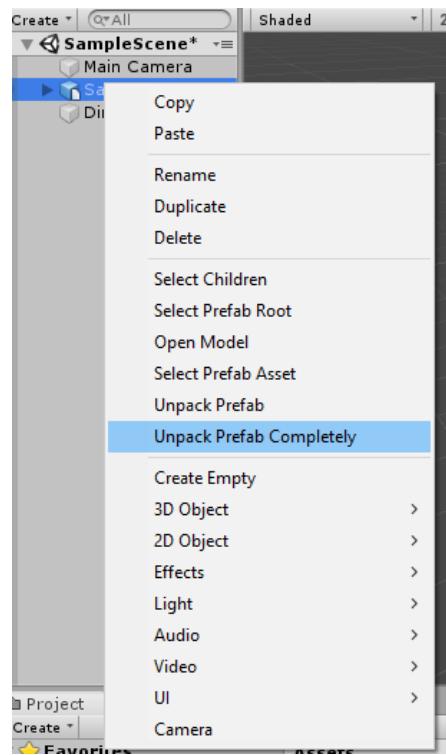


Рисунок 9 – Распаковка prefaba

Создаем новые материалы и накладываем на них текстуры (Рисунок 10).

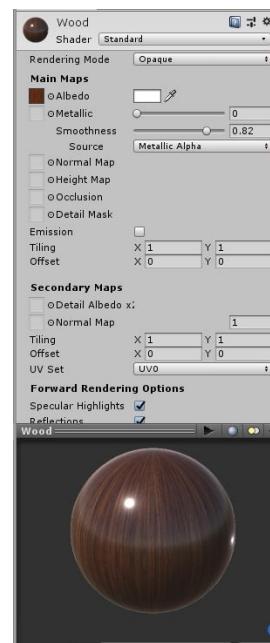


Рисунок 10 – Настройка материала

Теперь переносим эти материалы на модель. Если все сделано правильно, то UV координаты и id материалов должны сохраниться, и тогда материалы лягут ровно. Если же при наложении материалов возникли проблемы с размером, то отладить его можно в инспекторе параметром Tiling.

(Рисунок11).

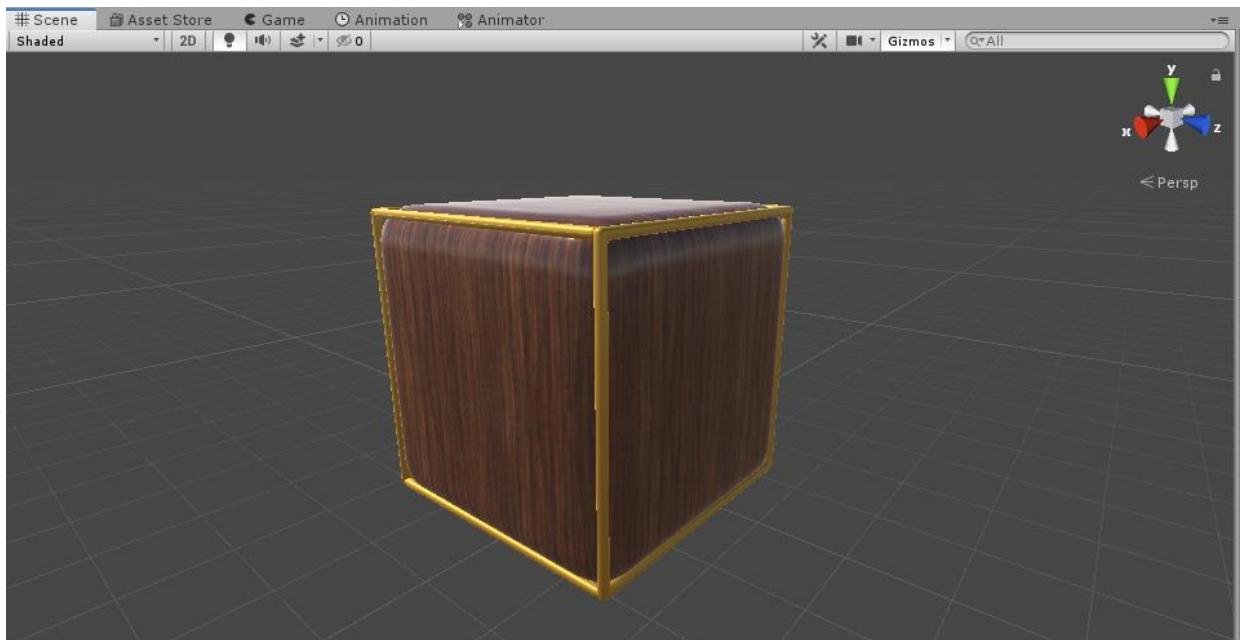


Рисунок 11 – Пример импортированной модели после наложения материалов

Для того, чтобы поделить проект между разработчиками, которые принимают участие в проекте, нужно нажать кнопку “Collab” и “Start Now!” (Рисунок 5).

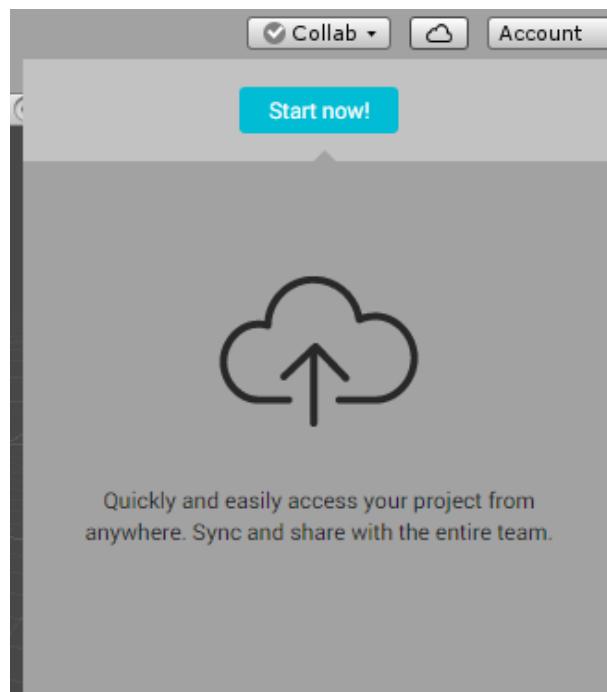


Рисунок 12 – Окно коллaborации

В открывшейся вкладке необходимо выбрать соответствующую организацию (Рисунок 13).

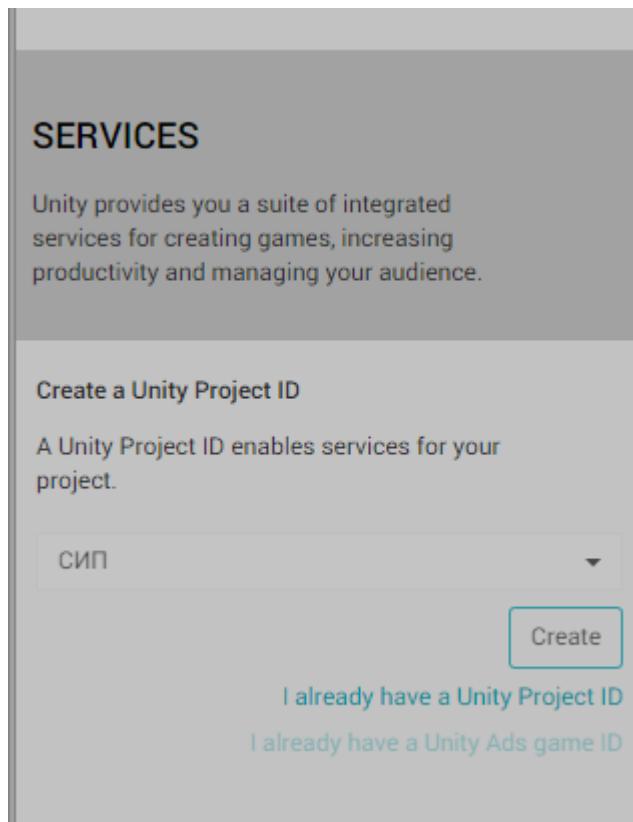


Рисунок 13 – Подключение организации к проекту

Чтобы создать организацию, нужно авторизоваться на сайте Unity, нажать на значок своего профиля на сайте и выбрать вкладку организации.

В открывшемся окне можно добавить новую организацию или вступить по приглашению в уже созданную (Рисунок 14).

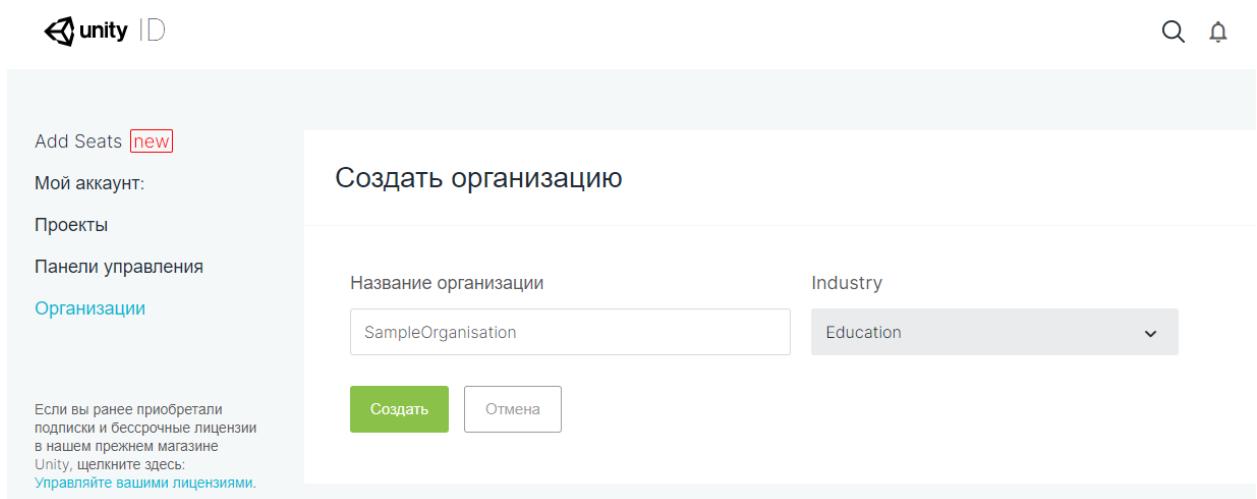


Рисунок 14 – Создание новой организации

После создания, необходимо зайти на страницу организации и перейти на вкладку “Члены”. Здесь можно выслать приглашения остальным

участникам проекта по электронной почте и выставить им подходящие роли (Рисунок 15).

Организации

SampleOrganisation

Члены

Подписки и сервисы

Использование сервисов

Изменить организацию

Asset Store Credit

Asset Management

Order Requests

Методы оплаты

Транзакции

Платежные реквизиты

Delete Organization

Если вы ранее приобретали подписки и бессрочные лицензии в нашем прежнем магазине Unity, щелкните здесь: Управляем вашими лицензиями.

Go to the [Unity Dashboard](#) to manage projects, discover and configure Unity services and find service support.

Пригласить

Ввести адреса эл. почты

Введите адреса эл. почты вручную или скопируйте их из списка. Пожалуйста, разделяйте их пробелами или запятыми, либо вводите по одному адресу на строку.

Дублированные адреса будут удалены из списка. Если участник уже добавлен, настройки этого участника обновятся.

Назначить участникам роль в организации

Если вы добавляете больше одного пользователя, рекомендуем добавлять их как пользователей, а затем вручную менять их роли.

manager

Роль участника можно изменить в любой момент.

Добавить участников в группы (По желанию)

Отмена Next

Рисунок 15 – Приглашение новых участников в организацию

Когда организация создана и указана в окне проекта, появится возможно сохранять изменения в облаке (Рисунок 16, 17).

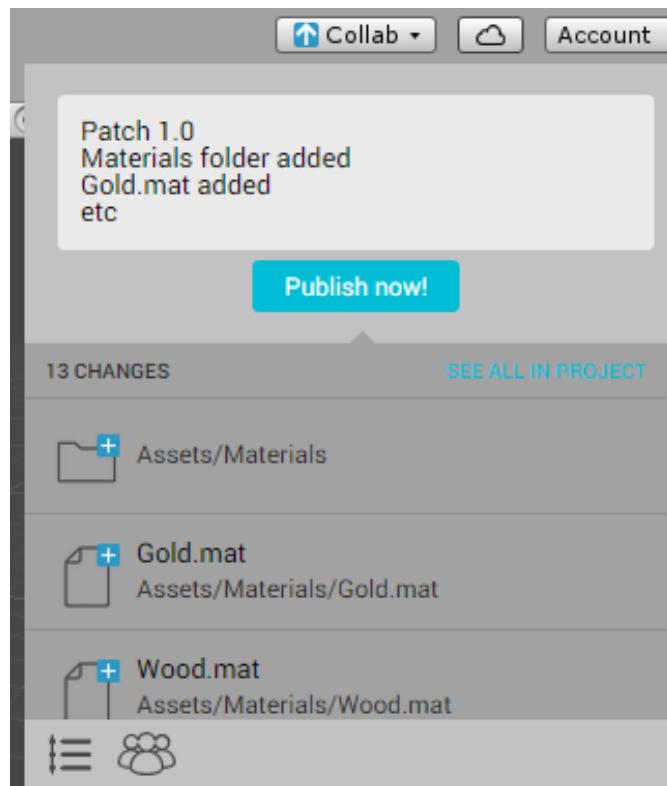


Рисунок 16 – Загрузка патча в облако

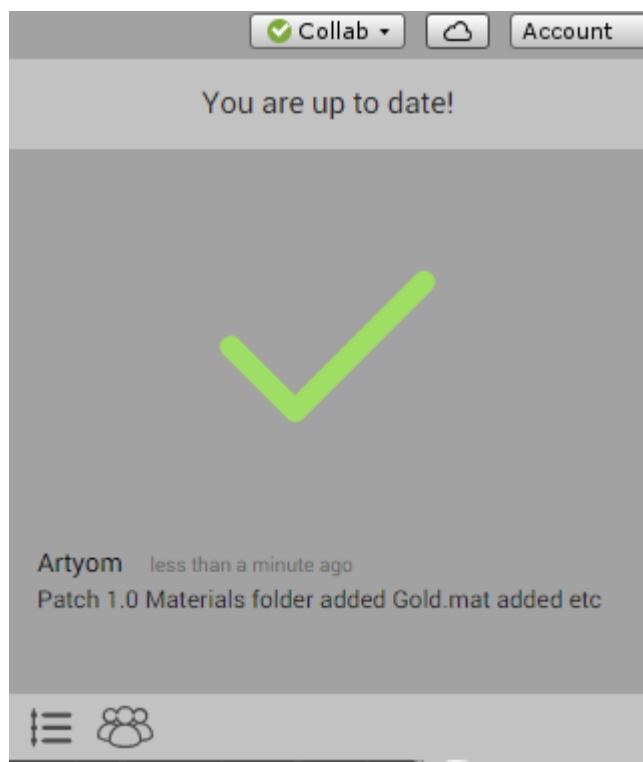


Рисунок 17 – Успешно выполненная операция по загрузке

Чтобы придать сцене более красивое оформление можно использовать плагин для Unity под название Post Processing. В этом плагине имеет большое

количество инструментов, которые позволяют создавать интересные визуальные эффекты в проекте.

Чтобы добавить плагин добавляется через вкладку Window, нажав на кнопку Package Manager (Рисунок 18).

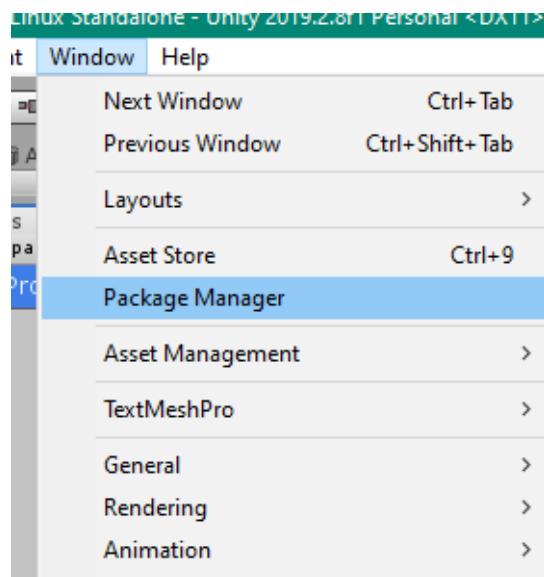


Рисунок 18 – Открытие окна Package Manager

Теперь с помощью поиска необходимо найти интересующий плагин и установить его через кнопку Install (Рисунок 19).

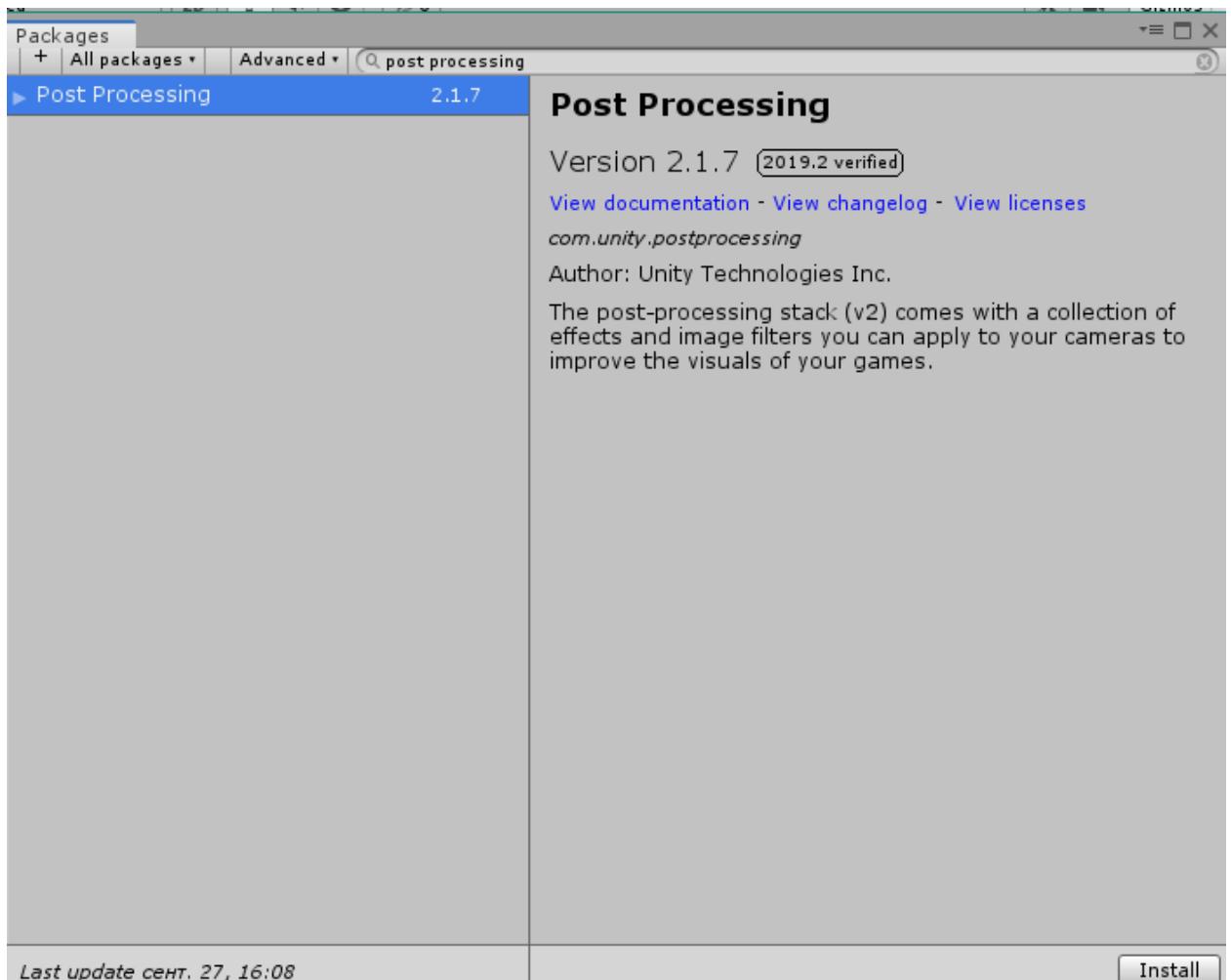


Рисунок 19 – Установка плагина Post Processing

Ключевыми компонентами плагина Post Processing являются: PP Layer и PP Volume. PP Layer – это основа всей пост обработки. Для того чтобы начать работу с эффектами, добавим новый слой, по аналогии с тегами (Рисунок 20).

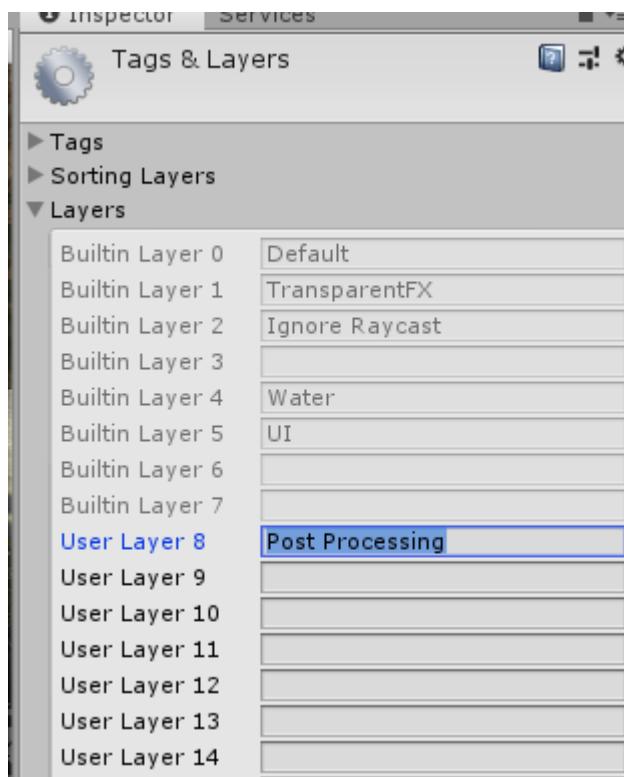


Рисунок 20 – Добавление нового слоя для пост обработки

К основной камере добавим компонент плагина Post-processing Layer и для него поставим созданный слой (Рисунок 21).

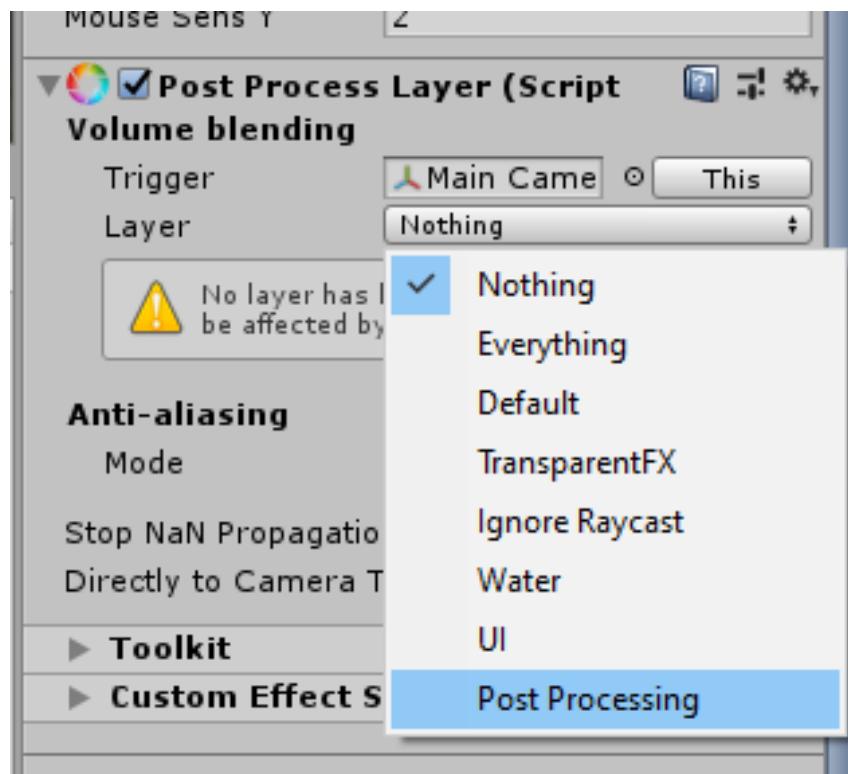


Рисунок 21 – Выбор слоя для обработки

Для сглаживания отображения моделей используется антиалайзинг, для этого примера будет использован мод Fast Approximate Anti-aliasing (Рисунок 22).

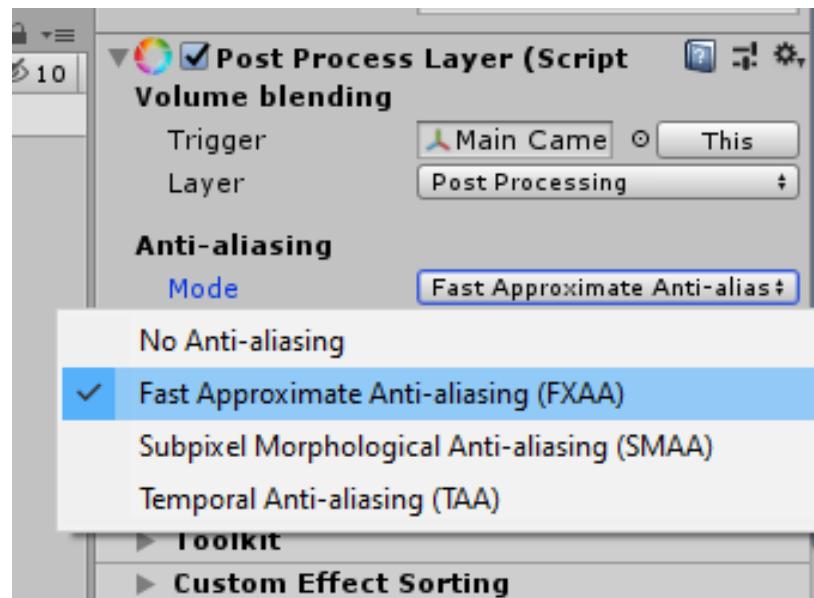


Рисунок 22 – Изменение мода антиалайзинга

Теперь создадим новый пустой объект и добавим ему в компоненты post-processing volume (Рисунок 23).

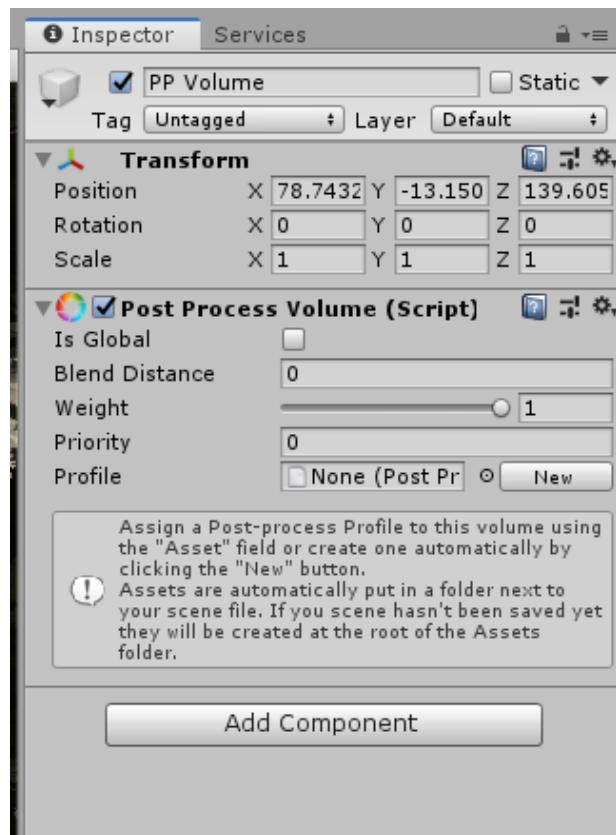


Рисунок 23 – Добавление компонента PP Volume на пустой объект

Нажимаем на кнопку New, чтобы создать профиль эффекта и поменять значение параметра Is Global на True, чтобы эффекты применялись вне зависимости от дистанции.

С помощью кнопки Add effect добавляем Bloom (Рисунок 24).

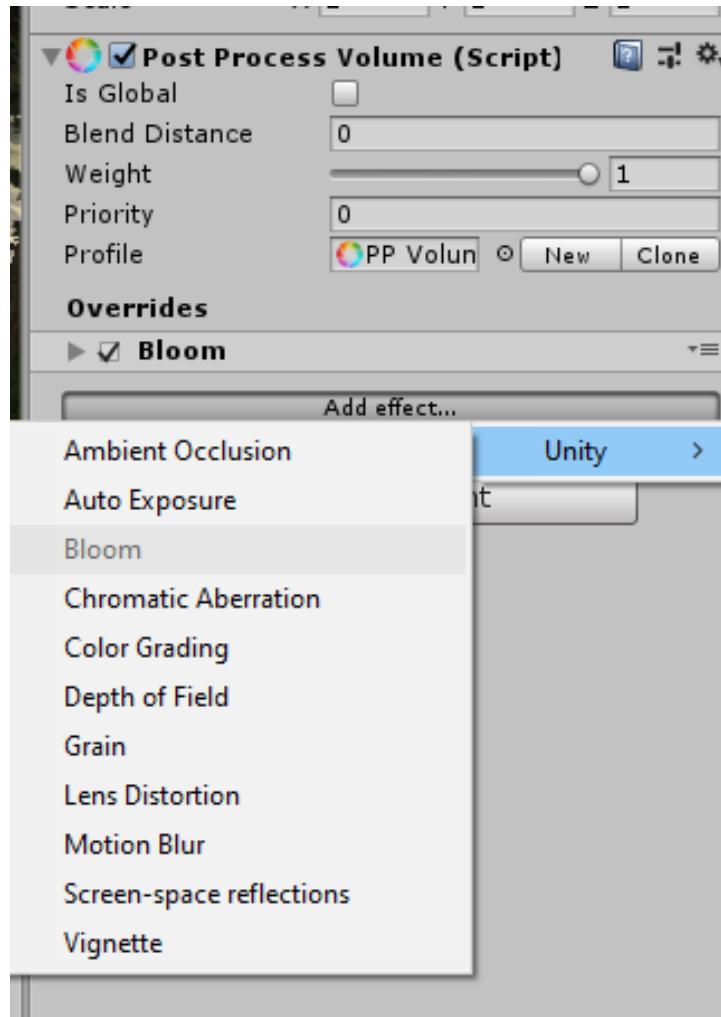


Рисунок 24 – Добавление эффекта Bloom

Настраиваем Intensity (интенсивность), Diffusion (диффузия) и Color (цвет) по вкусу, в примере имитируется закат (Рисунок 25).

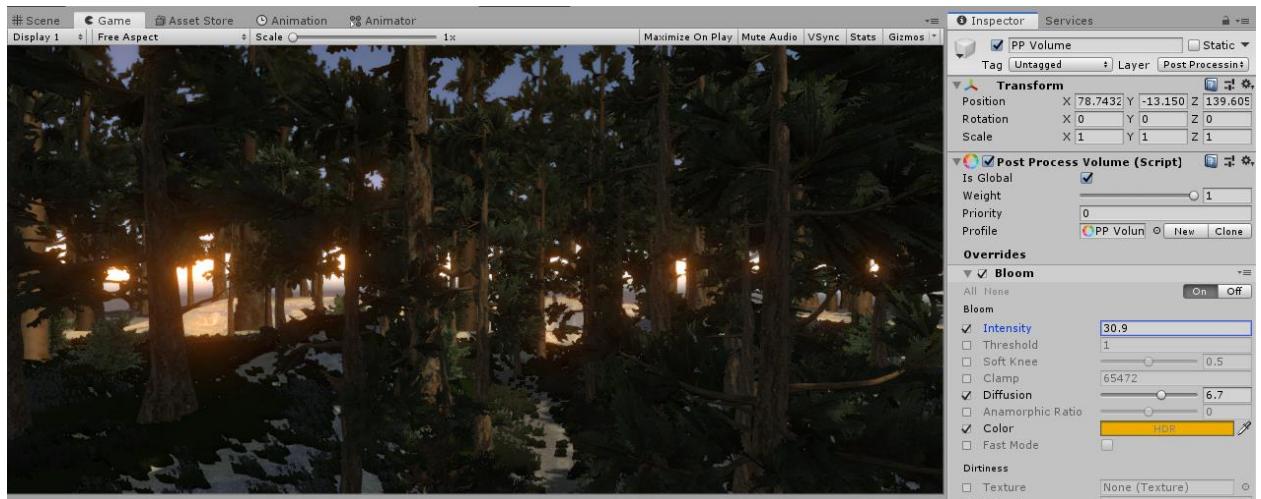


Рисунок 25 – Пример настроек для эффекта Bloom

Теперь добавим Ambient Occlusion для улучшения эффекта (Рисунок 26).

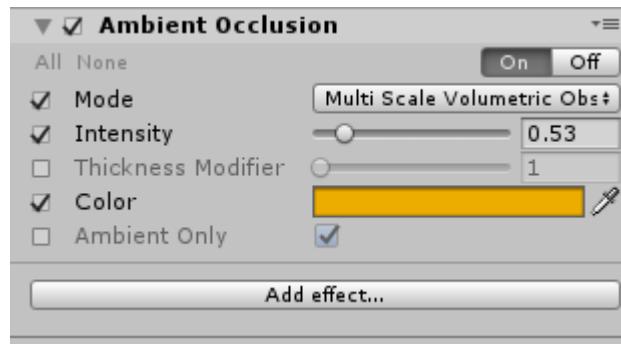


Рисунок 26 – Пример настроек для эффекта Ambient Occlusion.

Список литературы

- 1.Unity Documentation. Unity User Manual (2018.2) [Электронный ресурс] // <https://docs.unity3d.com/Manual/index.html>
2. Алан Торн. Искусство создания сценариев в Unity. – М.: ДМК Пресс, 2016, – 360 с.
3. Джонатан Линовес. Виртуальная реальность в Unity. – М.: ДМК Пресс, 2016, – 316 с.
4. Александр Горелик. Самоучитель 3ds Max. – СПб.: БХВ-Петербург, 2018, – 528 с.
5. Миловская О.С. 3ds Max 2018. Дизайн интерьеров и архитектуры. – СПб.: Питер, 2018, – 400 с.
6. Джейфри Рихтер. CLR via C#. Программирование на платформе microsoft .NET FRAMEWORK 4.5 на языке C#. - СПб.: Питер, 2020, - 895 с.

Сведения об авторах

Болбаков Роман Геннадьевич, кандидат технических наук, доцент, заведующий кафедрой инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА;

Синицын Анатолий Васильевич, старший преподаватель кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА;

Горбатов Григорий Витальевич, лаборант кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА;

Абрамов Артемий Алексеевич, студент кафедры практической и прикладной информатики Института информационных технологий РТУ МИРЭА.