

Лабораторная работа №1: Минигольф

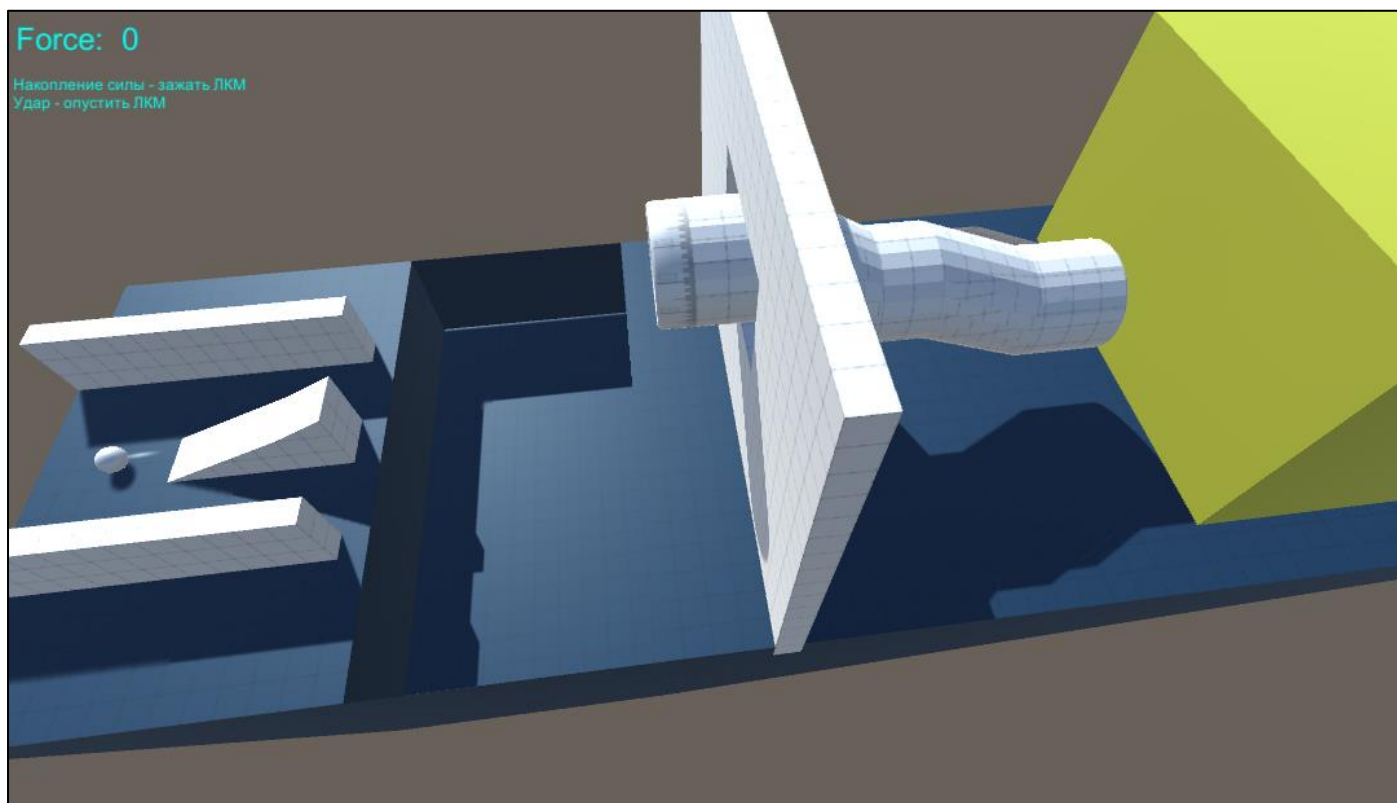
Цель работы: знакомство с расширениями и закрепление навыков работы в Unity

Задание:

Реализовать приложение – минигольф. Приложение должно содержать:

1. Шар, управляемый игроком. Управление осуществляется путём выбора направления и силы удара, после чего, шар отправляется в полёт. Во время полёта, управление шаром должно быть не доступно.
2. Как минимум два игровых уровня. Переход на второй игровой уровень должен осуществляться при достижении победы на первом.
3. При попадании шара за пределы игрового уровня, должно появляться сообщение о поражении и предложение перезапустить уровень. При достижении шаром окончания уровня, должно появляться сообщение о победе и предложение запустить следующий уровень.
4. Хотя бы один игровой уровень должен содержать препятствия следующего типа:
 - а. Мельница. Вращающийся объект, имеющий лопасти, препятствующие перемещению шара.
 - б. Дверь. Арка и заслонка, перекрывающая арку с определённой периодичностью.

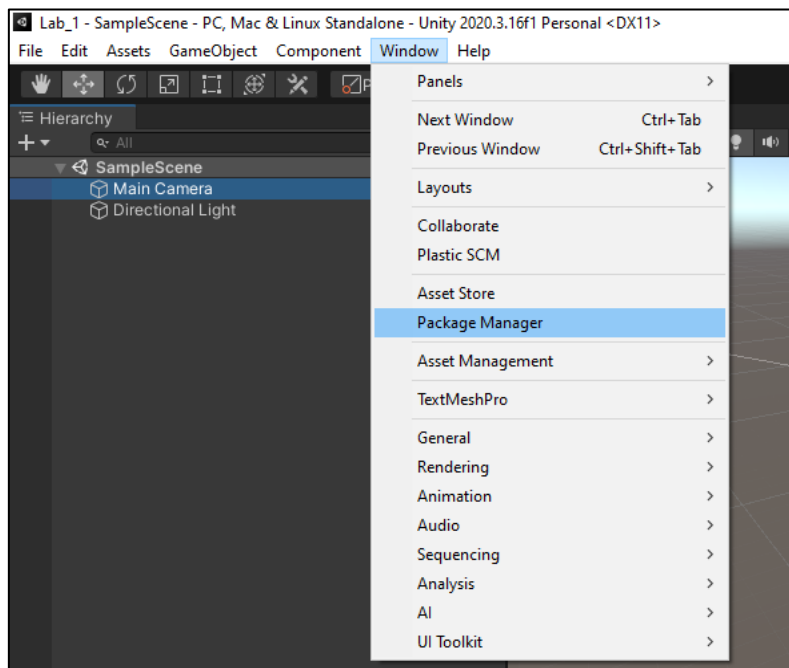
Пример простого уровня: <https://h-anim.github.io/Golf/>



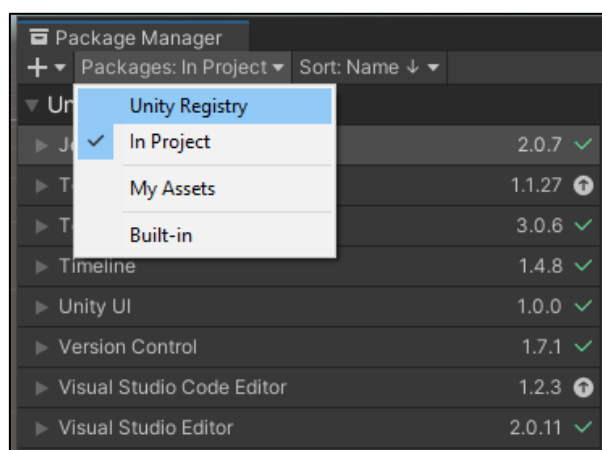
Пример препятствий: <https://h-anim.github.io/trials/>

Дополнительные модули

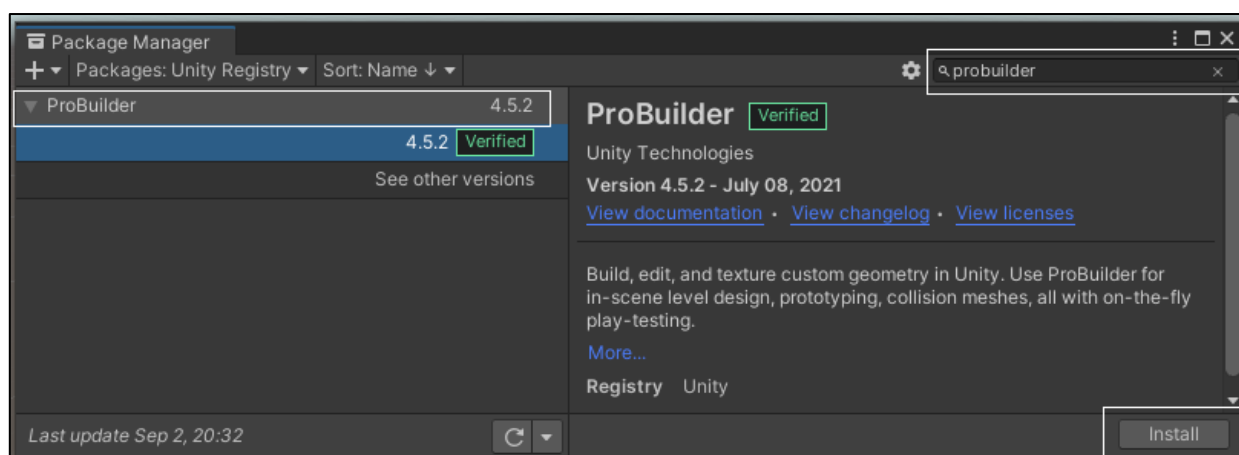
Установить расширения для Unity можно кликнув по Window и выбрав Package Manager:



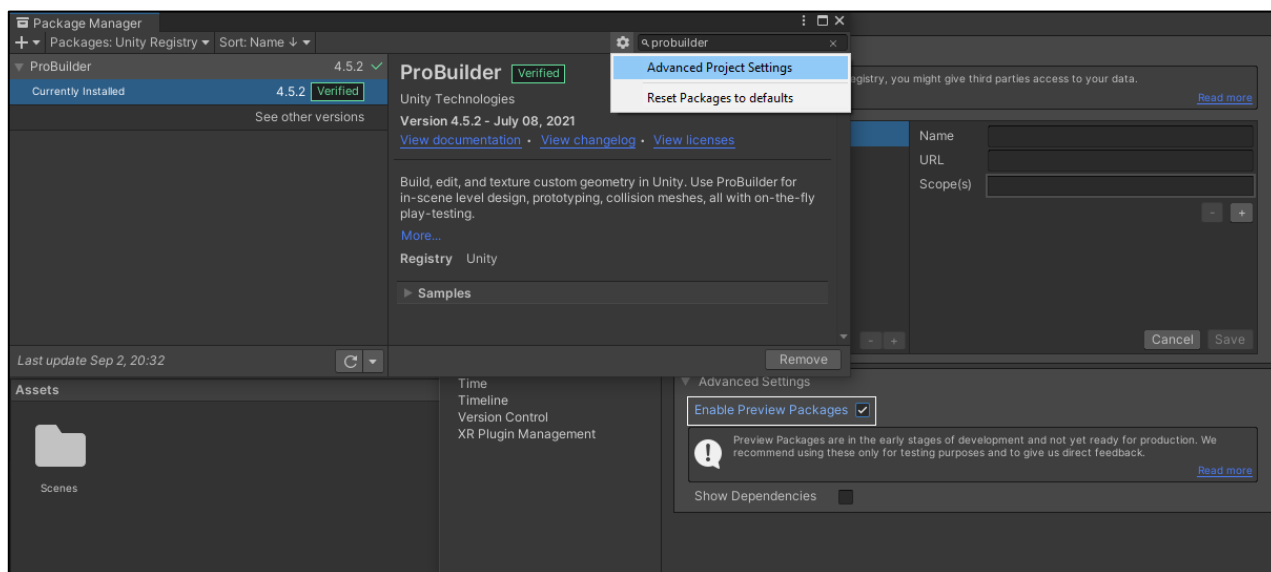
По умолчанию, в списке отображаются пакеты, уже добавленные в проект, чтобы переключиться к общему списку, кликните на Packages и отметьте Unity Registry:



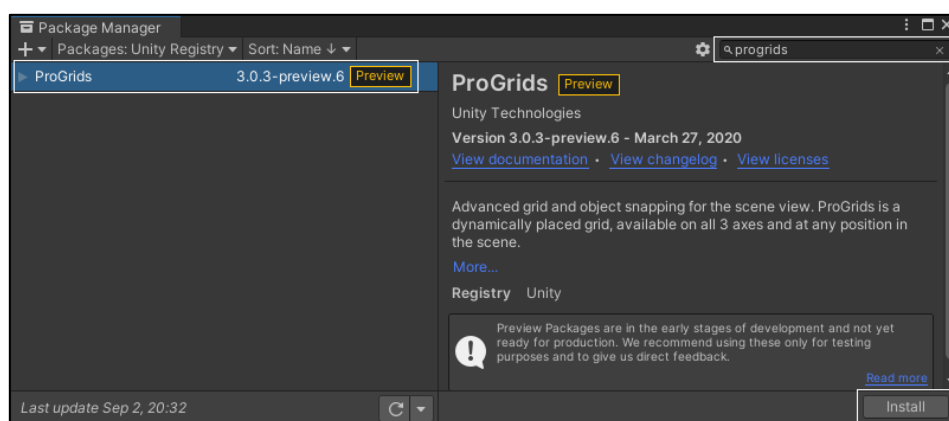
В рамках работы над лабораторной работой, вам могут пригодиться два расширения, ProBuilder и ProGrids. Установить ProBuilder можно введя название в строку поиска, выбрав пакет и нажав Install:



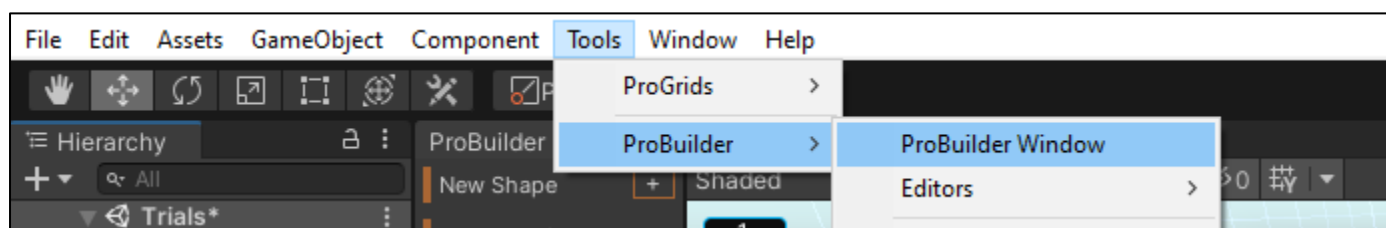
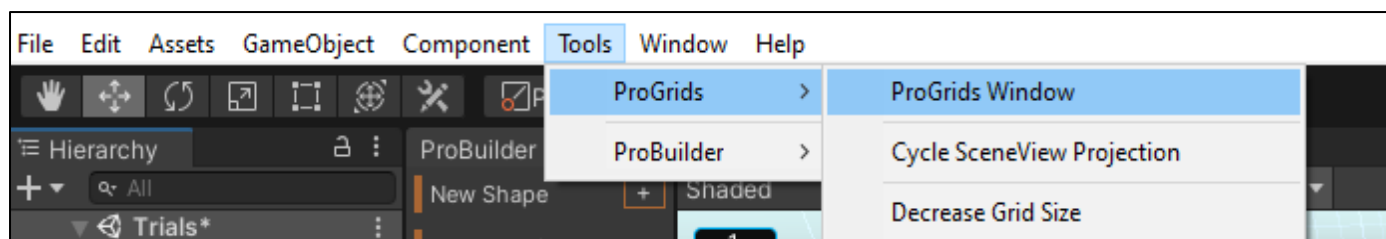
ProGrids, является пакетом находящимся в разработке, поэтому для того, что бы он появился в списке доступных расширений, необходимо кликнуть на знак шестерни рядом со строкой поиска, выбрать Advanced Project Settings, а затем, в появившемся окне, отметить поле Enable Preview Packages:



Если всё было сделано правильно, то воспользовавшись строкой поиска, можно найти и установить расширение ProGrids:

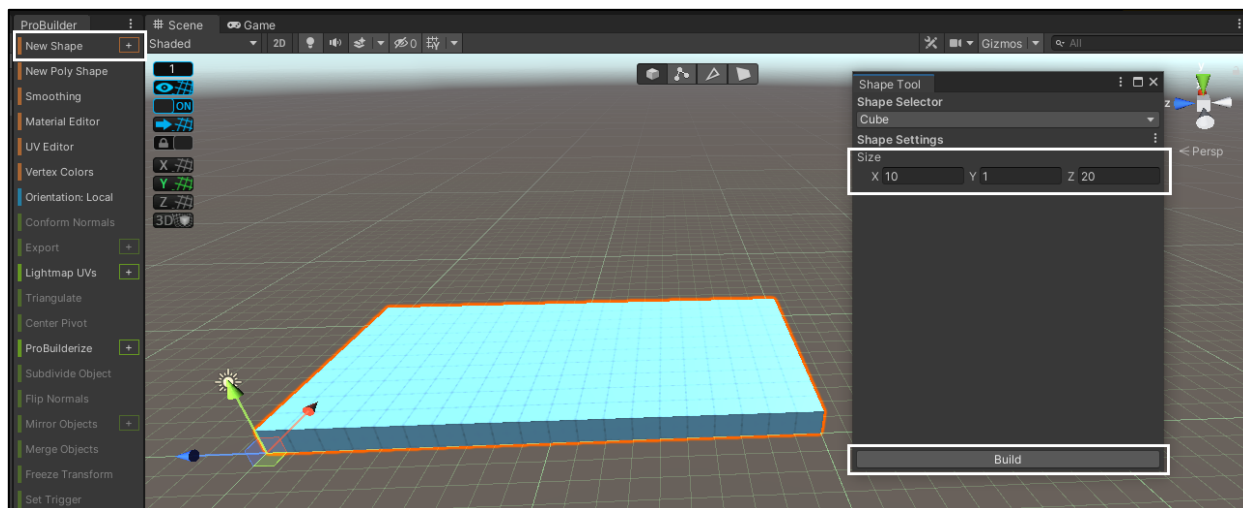


Активировать окна расширений можно кликнув Tools, выбрав нужное расширение и нажав кнопку с окончанием Window:

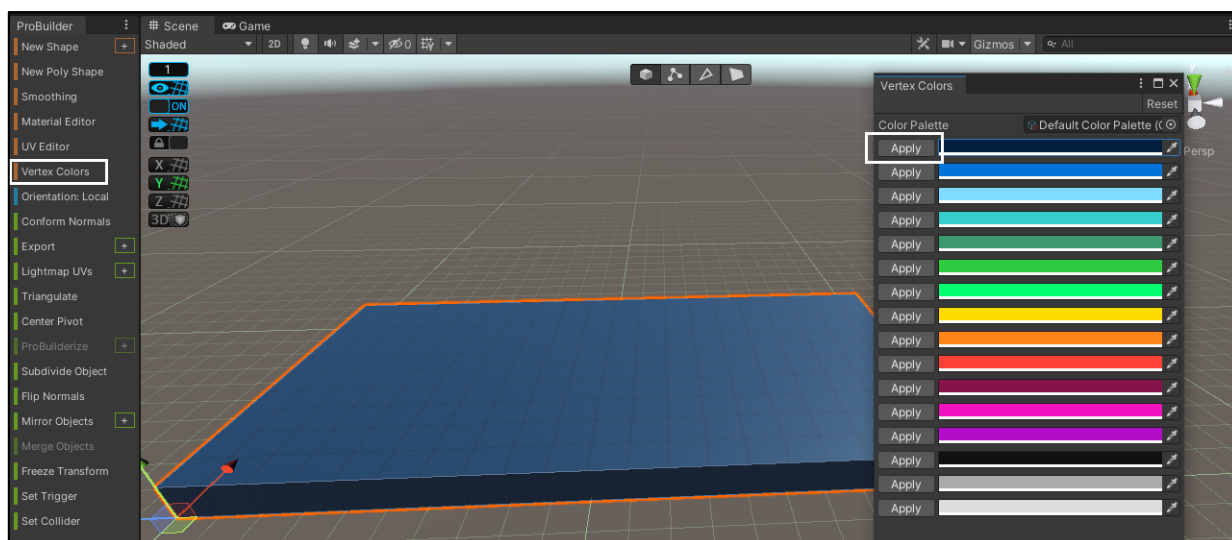


ProBuilder является простым 3D редактором, а ProGrids – инструментом позиционирования объектов.

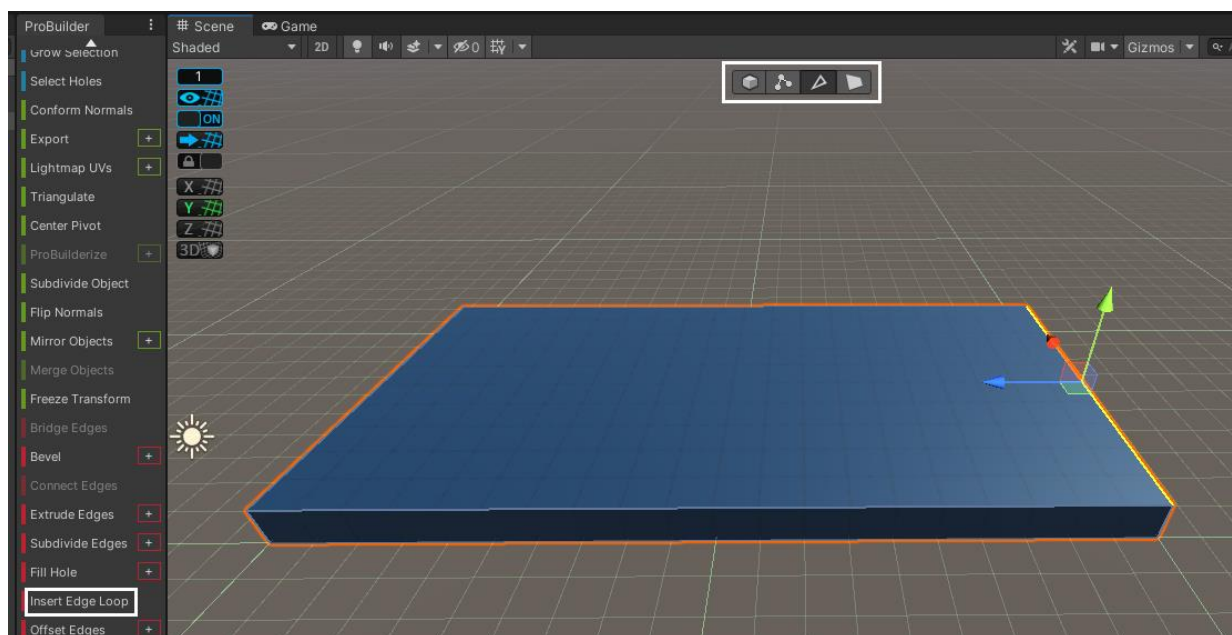
Добавить в сцену объект, используя ProBuilder, можно кликнув на знак “+” рядом с New Shape, выбрав тип объекта, его начальные параметры и кликнув Build:



При желании, можно сменить цвет объекта, для этого выберите объект, затем Vertex Colors, выберите желаемый цвет в появившемся меню, а затем Apply:

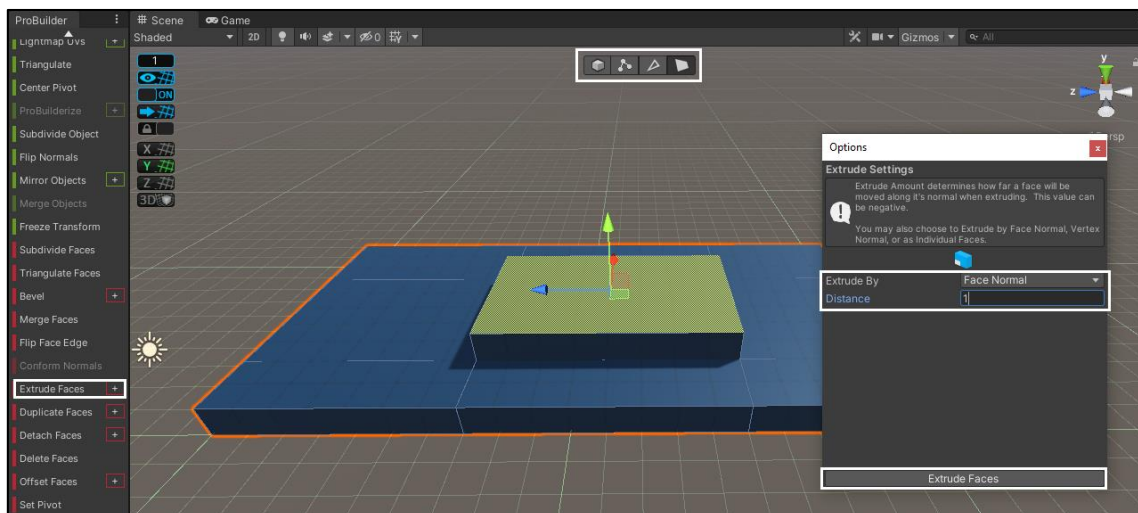


Обратите внимание, что при выборе объекта, на верху экрана доступно меню режима выбора:

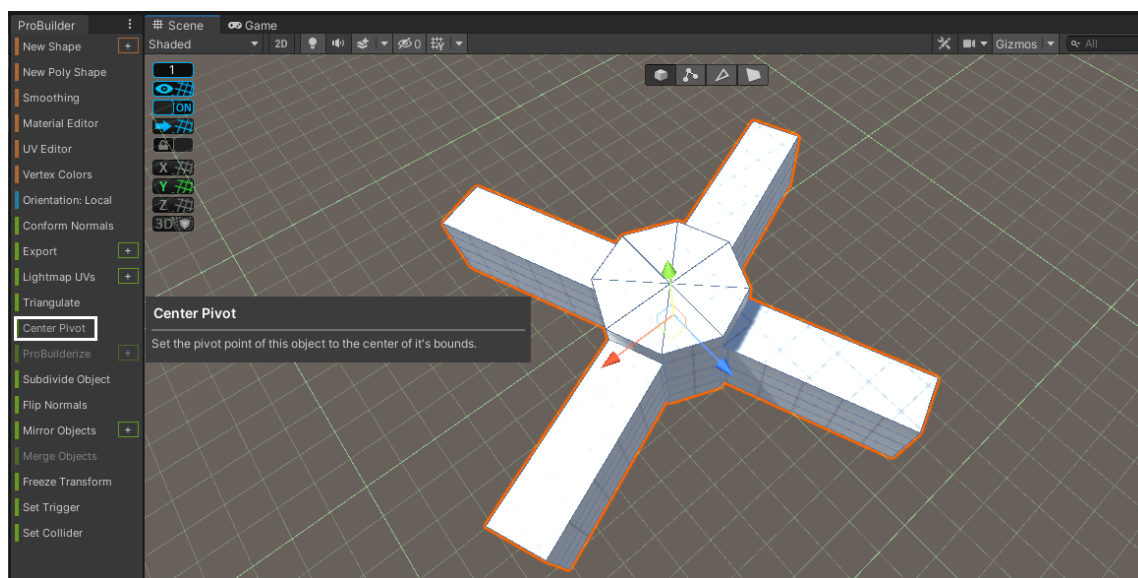


Используя это меню, можно, к примеру, выбрать отдельную грань объекта и выполнить его рассечение, используя инструмент Insert Edge Loop.

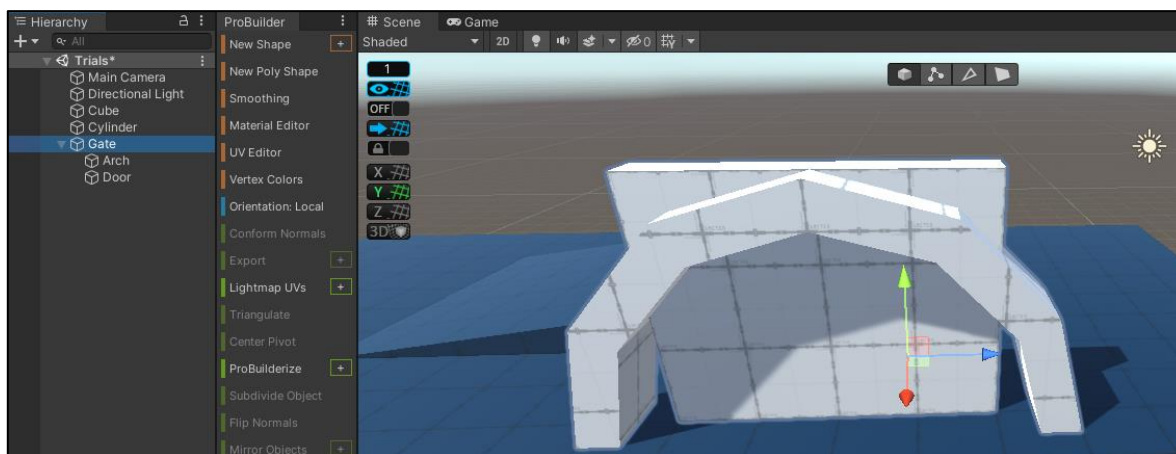
Добавив новую геометрию путём рассечения каждого ребра объекта, можно, используя режим выбора поверхностей, выполнить вытягивание части объекта, используя инструмент Extrude Faces:



Ниже приведён пример объекта, созданного из цилиндра, при помощи инструмента Extrude Faces. Основание объекта можно переместить в его геометрический центр используя команду Center Pivot:

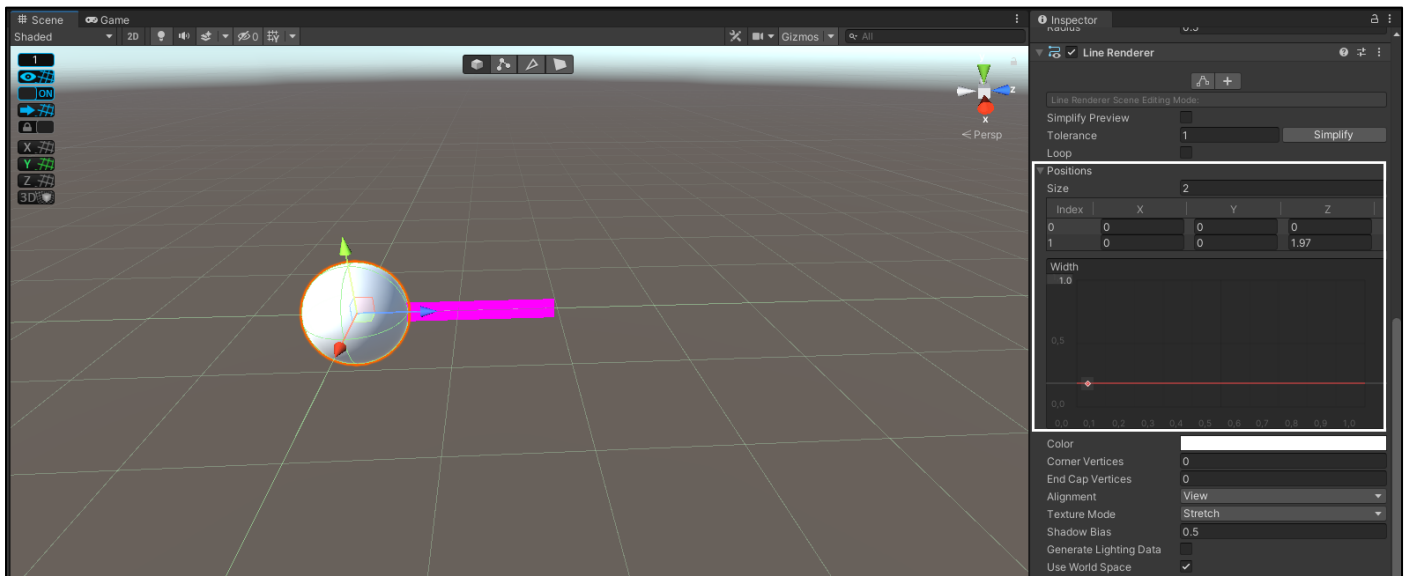


В случае, если необходимо создать объект, состоящий из подобъектов, удобно использовать Empty Object в качестве контейнера для хранения этих подобъектов:

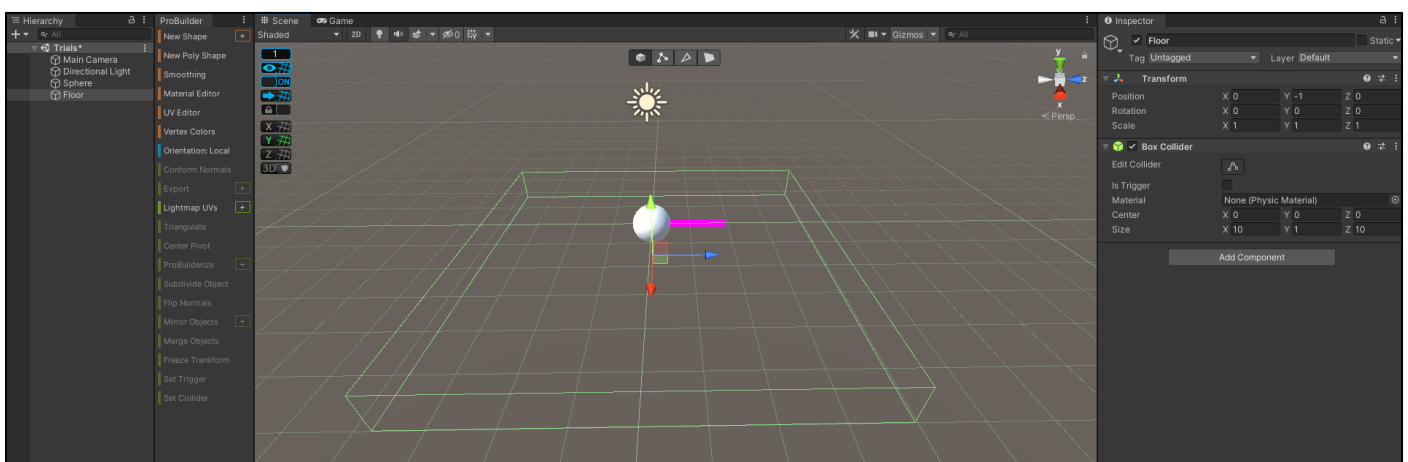


Line Renderer

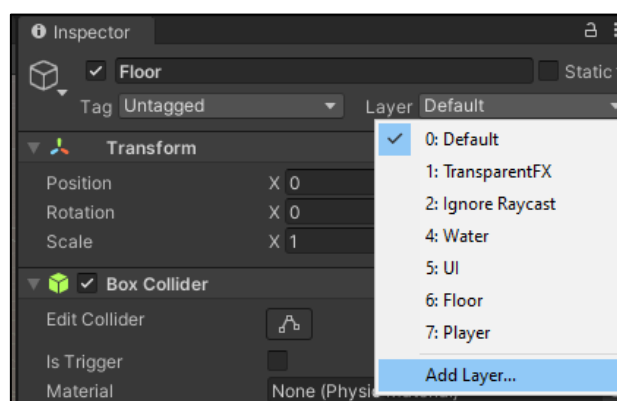
Отобразить вектор между курсором мыши и объектом можно, например, при помощи компонента LineRenderer. Добавить его к объекту можно при помощи меню Add Component. В инспекторе объектов можно настроить ширину линии и количество точек из которых она состоит:



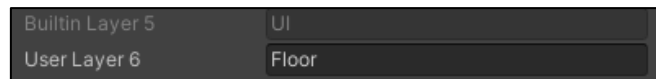
Для отслеживания перемещений курсора мыши в сцене можно создать пустой объект и добавить к нему Collider. Объект должен располагаться ниже сферы и иметь достаточно большие размеры:



Созданный коллайдер будет использоваться для поиска пересечения проекции курсора мыши с объектами сцены. Для того, чтобы поиск пересечения осуществлялся только с данным коллайдером, имеет смысл вынести его на отдельный слой. Для этого кликните на список Layer и нажмите Add Layer:



В появившемся списке, добавьте название для нового слоя:



Не забудьте выбрать это новое название для объекта, с которым будет осуществляться поиск пересечения.

Скрипт проверки пересечения и расчёта направления между объектом проекцией курсора мыши может выглядеть следующим образом:

```
public class PScript : MonoBehaviour
{
    public Camera cam; //ссылка на камеру
    public LayerMask floor; //номер слоя, на котором будут происходить поиск пересечений
    LineRenderer lr; //ссылка на Line Renderer

    Ссылка: 0
    void Start()
    {
        lr = GetComponent<LineRenderer>(); //получение ссылки на Line Renderer
    }

    Ссылка: 0
    void LateUpdate()
    {
        RaycastHit hit;
        Ray ray = cam.ScreenPointToRay(Input.mousePosition); //проекция луча из позиции курсора мыши

        lr.SetPosition(0, transform.position); //установка изначальных позиций точек линии
        lr.SetPosition(1, transform.position);

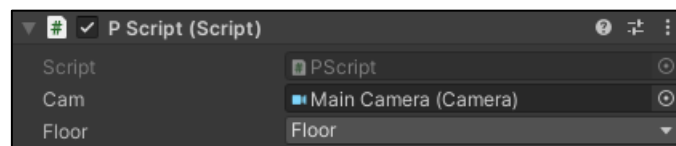
        if (Physics.Raycast(ray, out hit, 1000, floor)) //если произошло пересечение с объектом на слое Floor не далее чем в 1000 единиц
        {
            Vector3 pnt = hit.point; //получение координат точки пересечения
            pnt.y = transform.position.y; //выравнивание точки пересечения по высоте сферы

            Vector3 dir = (pnt - transform.position); //вычисление направления до точки пересечения
            dir.Normalize(); //нормализация вектора направления
            dir *= 2; //установка длины вектора направления

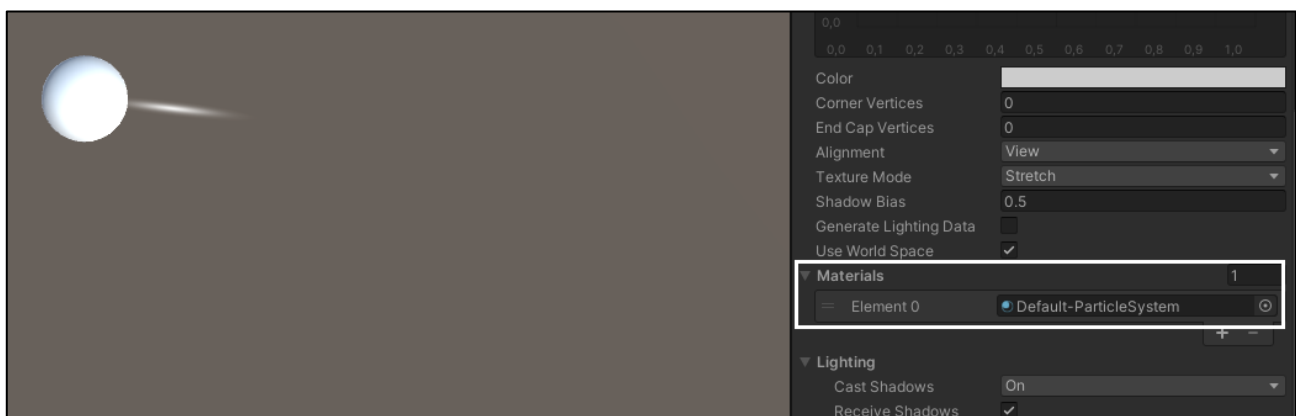
            lr.SetPosition(1, (transform.position + dir)); //установка точки окончания линии
        }
    }
}
```

Обратите внимание, что в данной реализации, направление рассчитывается только в плоскости X | Z.

Заполнение параметров скрипта будет выглядеть следующим образом:

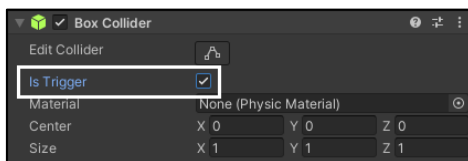


В качестве индикатора направления можно использовать любую текстуру. Для этого, выберите объект, перейдите в раздел Line Renderer в инспекторе и установите желаемую текстуру в качестве материала:



Триггеры

В случае, если необходимо обнаружить достижение объектом определённой позиции, можно использовать механизм триггеров. Для этого, создайте объект с коллайдером (пустой или 3D модель), а затем отметьте в поле Is Trigger:



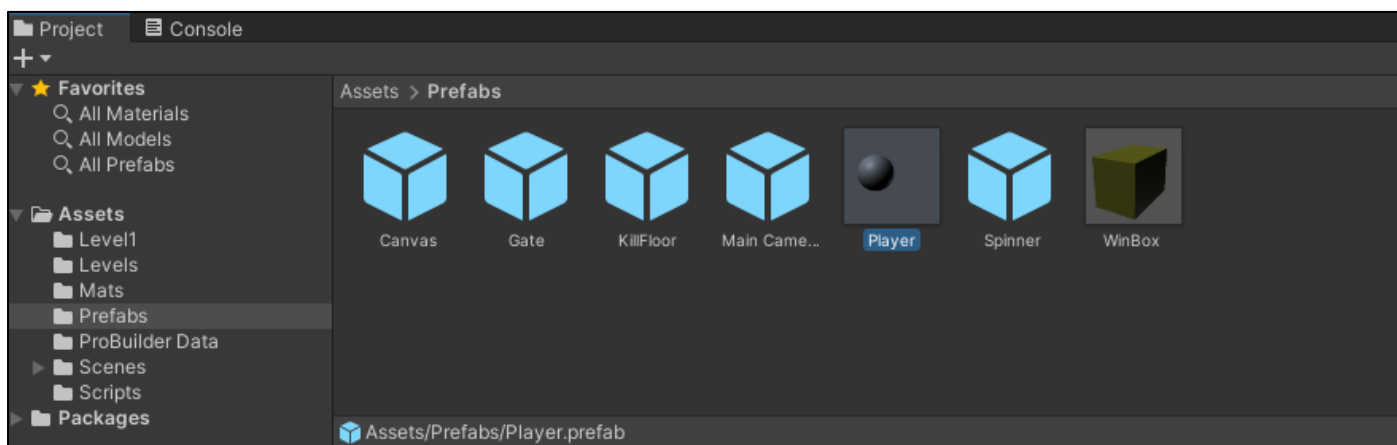
Такой объект не будет препятствовать перемещению других объектов, но при этом, при пересечении, будет генерировать событие OnTriggerEnter. Ниже приведён пример использования триггера для обнаружения пересечения с объектом, имеющим тэг (tag) Player:

```
private void OnTriggerEnter(Collider other)
{
    if (other.transform.CompareTag("Player") == true)
    {
        //действия при пересечении игрока с триггером
    }
}
```

Префабы

В каждой новой сцене, вам придётся заново размещать объекты камера, интерфейс и игрок. Кроме того, существуют объекты (например, препятствия), единственным отличием которых являются позиция, поворот, параметры ассоциированных скриптов и т.д. Для быстрой работы с такими объектами существует механизм Prefabs, который позволяет создавать шаблоны объектов.

Для создания Prefab, добавьте в сцену объект, добавьте к объекту все необходимые компоненты, а затем, перетащите его в раздел Assets:



Если всё было сделано правильно, можно создавать копии объекта из шаблонов, просто перетаскивая их в сцену, или в иерархию объектов.

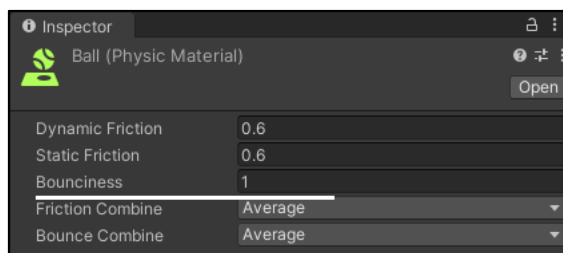
TimeScale

В случае необходимости, можно остановить время в рамках проекта, замедлить, ускорить или восстановить его нормальное течение. Осуществить это можно при помощи изменения значений переменной timeScale:

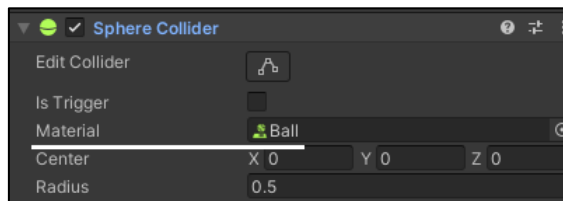
```
Time.timeScale = 0; //полная остановка времени
Time.timeScale = 1; //нормальное течение времени
```


Физический материал

Настроить прыгучесть сферы, управляемой игроком, можно создав физический материал, задав параметр Bounciness:

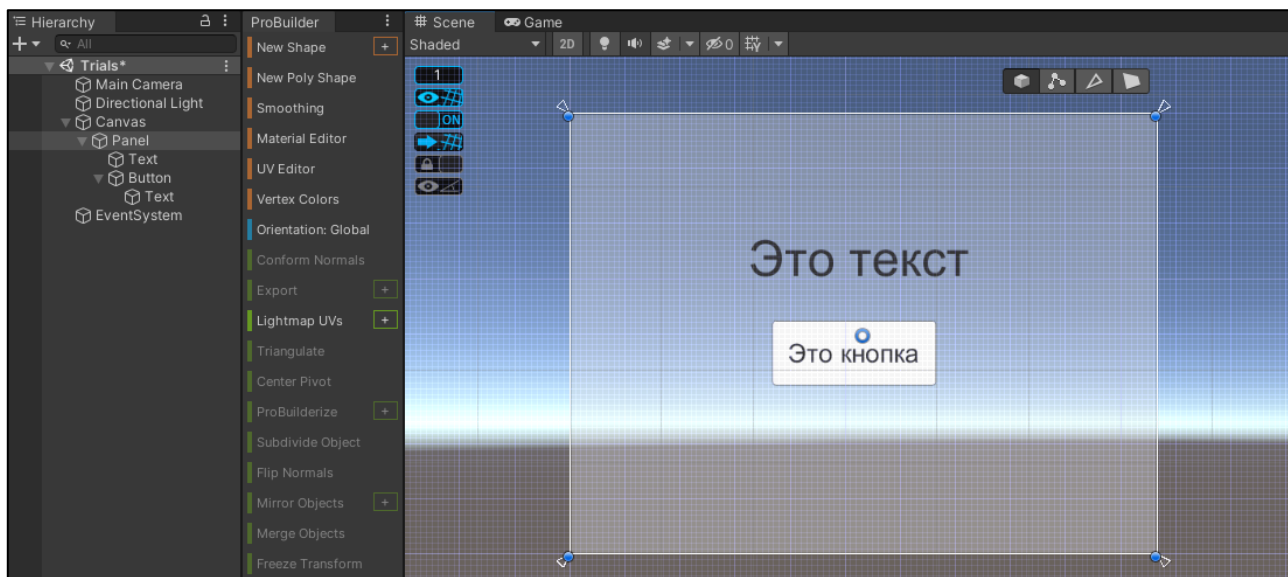


А затем установив этот материал в коллайдер сферы:



Панели

Сообщения о победе и поражении (а также разнообразные меню) удобно организовывать при помощи панелей. Панель, это элемент UI Unity, которую можно добавить кликнув правой кнопкой в области иерархии и выбрав UI->Panel:



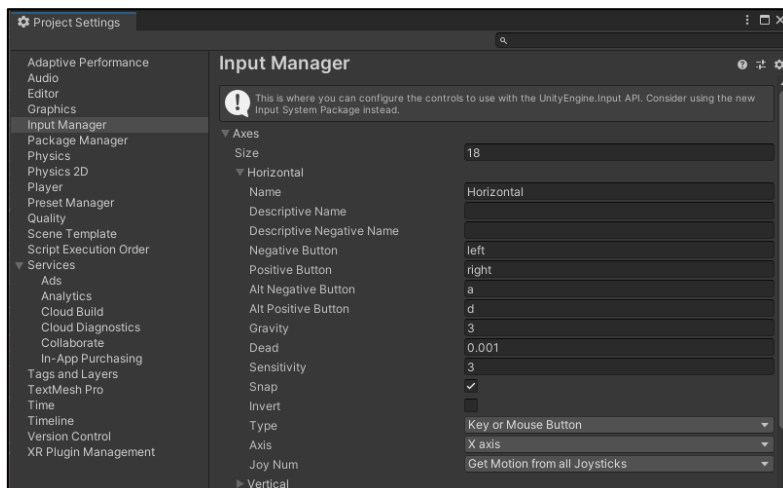
Показать или скрыть панель можно передав ссылку на неё в соответствующий скрипт и используя метод SetActive:

```
public class WinScript : MonoBehaviour
{
    public GameObject panel; //ссылка на панель

    Ссылка: 0
    private void OnTriggerEnter(Collider other) //игрок достиг финиша
    {
        panel.SetActive(true); //показать панель с сообщением о победе
        Time.timeScale = 0; //остановить время в сцене
    }
}
```

Перемещение камеры

Посмотреть информацию о вводе в приложении можно выбрав Edit->Project Settings->Input Manager:



Реализация перемещения и вращения камеры может быть выполнена следующим образом:

```
void FixedUpdate()
{
    float vert = Input.GetAxisRaw("Vertical"); //получение смещения по вертикали
    float hor = Input.GetAxisRaw("Horizontal"); //получение смещения по горизонтали

    Vector3 dir = new Vector3(hor, 0, vert); //получение вектора направления смещения камеры
    dir.Normalize(); //нормализация вектора направления смещения камеры

    dir = transform.TransformDirection(dir) * Time.fixedDeltaTime * moveSpeed; //вычисление скорости смещения камеры

    cam.transform.position += dir; //смещение камеры

    //повороты камеры в глобальных координатах (альтернативно, можно реализовать поворот вокруг объекта сцены)
    if (Input.GetKey(KeyCode.Q) == true)
    {
        cam.transform.Rotate(new Vector3(0, 1, 0), -angSpeed * Time.fixedDeltaTime, Space.World);
    }

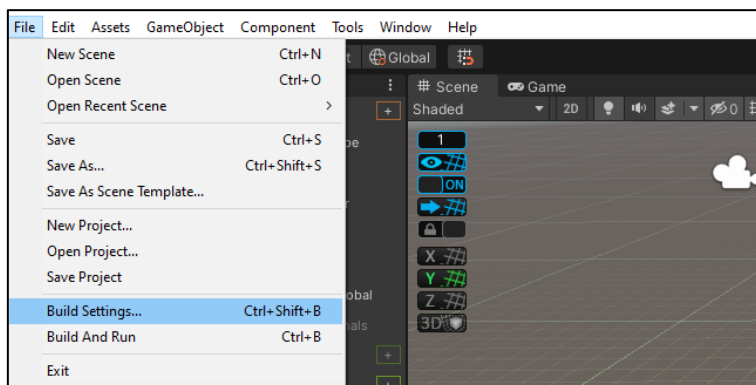
    if (Input.GetKey(KeyCode.E) == true)
    {
        cam.transform.Rotate(new Vector3(0, 1, 0), angSpeed * Time.fixedDeltaTime, Space.World);
    }
}
```

Загрузка уровней

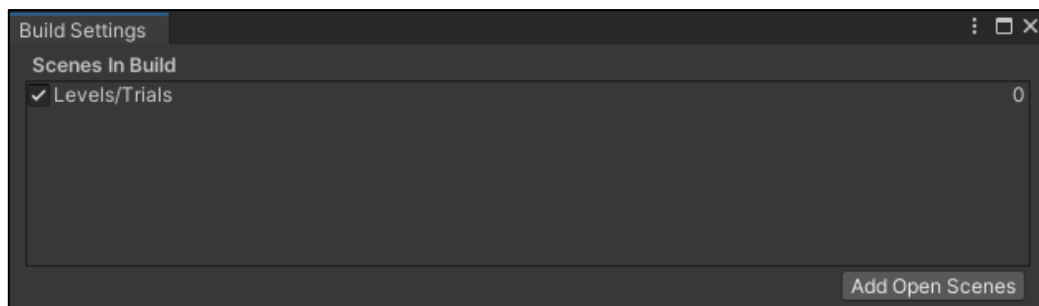
Манипуляции со сценами выполняются при помощи метода:

```
SceneManager.LoadScene()
```

Параметром метода может являться либо строка, содержащая название сцены, либо число, обозначающее индекс сцены. Чтобы узнать индекс сцены, загрузите все сцены вашего проекта, перейдите в Build Settings:



А затем кликните Add Open Scenes:



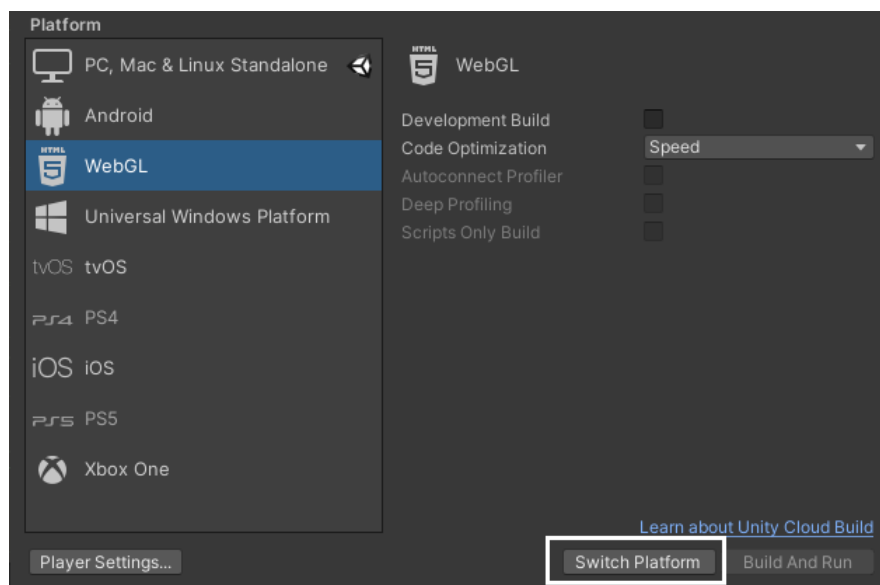
В список будут добавлены все открытые сцены, а справа будут отображены их индексы.

Метод загрузки сцены может быть так же использован для перезагрузки текущей сцены:

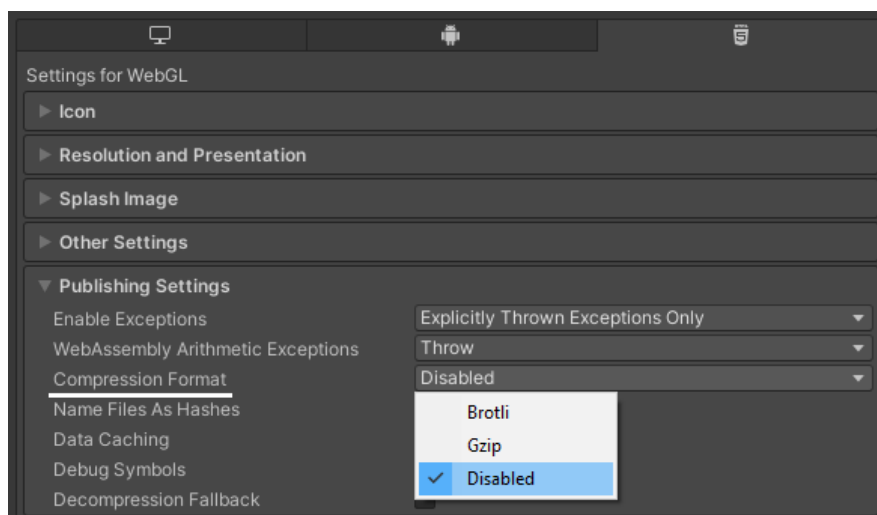
```
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
```

Сборка и размещение проекта WebGL на GitHub




Что бы разместить собрать проект для интеграции в веб страницу, необходимо в разделе Build Settings выбрать WebGL и кликнуть Switch Platform:



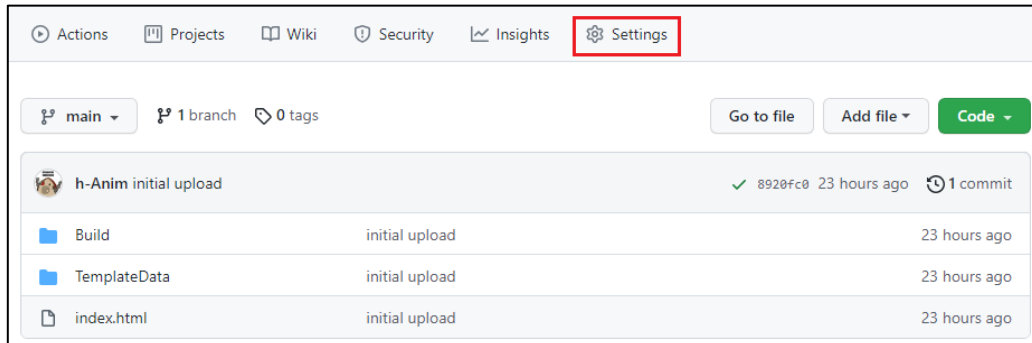
Затем в разделе Player Settings, выберите желаемые название проекта и настройки. Обратите внимание, что некоторые сервера (в частности github pages) не поддерживают сжатие, поэтому его можно отключить:



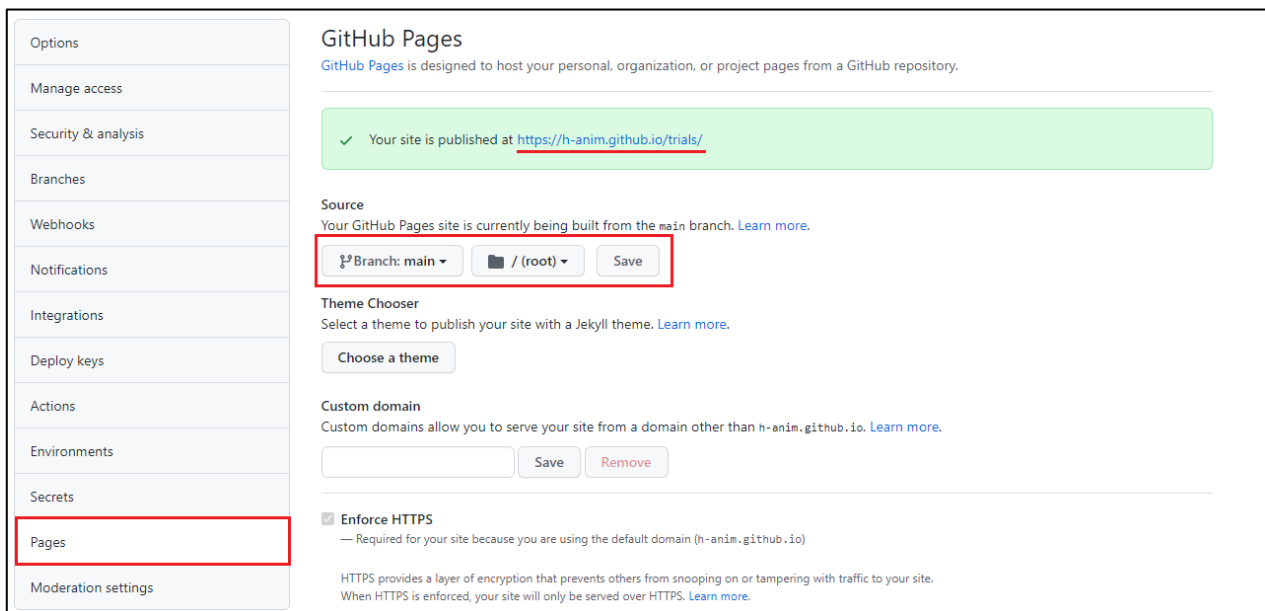
Если всё было сделано правильно, то после нажатия Build, выбранная папка будет выглядеть примерно так:

 Build	11.09.2021 21:35	Папка с файлами	
 TemplateData	11.09.2021 21:35	Папка с файлами	
 index.html	11.09.2021 21:35	Chrome HTML Do...	4 КБ

Загрузите собранный проект на github и перейдите в раздел Settings:



Перейдите в раздел Pages, выберите ветвь main и нажмите Save:



В случае если полученная ссылка будет вести на страницу с ошибкой 404, попробуйте переименовать репозиторий.