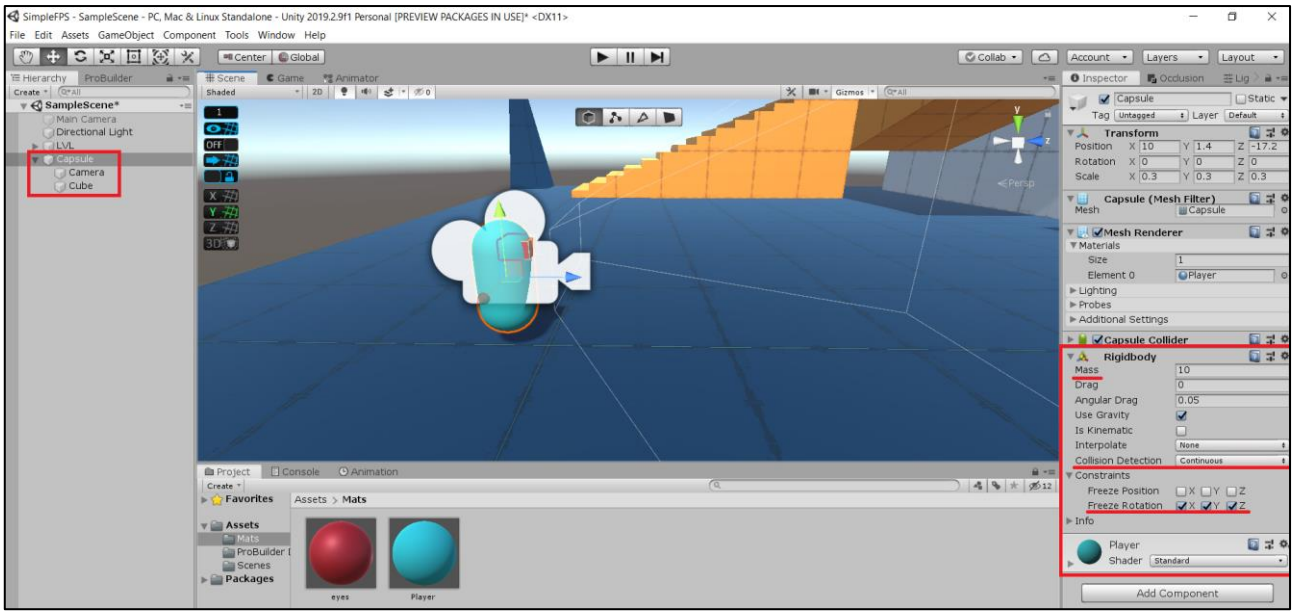
Лабораторная работа №3:
Динамическое создание объектов.

Цель: Целью лабораторной работы является знакомство с методом динамического создания и уничтожения объектов в среде Unity.

Перемещение

Существует два концептуальных подхода к реализации перемещения персонажей: с учётом физической модели и без. В данном документе рассматривается вариант, когда персонаж имеет физическую модель.

Для удобства позиционирования игрока, добавьте в сцену капсулу, в капсулу добавьте куб и новую камеру. Отмасштабируйте и разместите куб так, чтобы обозначить зону, где будет голова/глаза игрока, а затем, разместите в этой зоне камеру. Добавьте к капсуле компонент “Rigidbody” и настройте его параметры:



Создайте и назначьте материалы для капсулы и куба, а затем, отключите объект Main Camera.

Добавьте к капсуле скрипт следующего вида:

```
public class Movement : MonoBehaviour
{
    [Range(50f, 200f)]
    public float mSpeed = 100f; //скорость перемещения

    Rigidbody rb; //ссылка на физическую модель объекта

    void Start()
    {
        rb = GetComponent<Rigidbody>(); //получение ссылки на физическую модель
    }

    void FixedUpdate()
    {
        float xMove = Input.GetAxisRaw("Horizontal");  //проверка нажатия кнопок a, d, left, right
        float zMove = Input.GetAxisRaw("Vertical");    //проверка нажатия кнопок w, s, up, down

        Vector3 dir = new Vector3(xMove, 0, zMove); //получение направления движения в плоскости X|Z
        dir.Normalize();                            //нормализация вектора движения
        //вектор скорости объекта = направление движения * скорость * время с прошедшего вызова этой функции
        Vector3 v = transform.TransformDirection(dir) * mSpeed * Time.fixedDeltaTime;
        v.y = rb.velocity.y; //восстановление смещения по оси Y
        rb.velocity = v;
    }
}
```

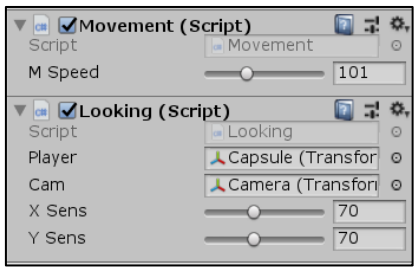
Если всё было сделано правильно, то при запуске, объект будет перемещаться по сцене в плоскости X|Z.

Поворот камеры

Добавьте к объекту игрока новый скрипт для описания поворотов камеры:

```
public class Looking : MonoBehaviour
{
    public Transform player; //ссылка на модель игрока
    public Transform cam;    //ссылка на камеру игрока

    [Range(50f, 100f)]
    public float xSens = 70f; //чувствительность мыши по оси X
    [Range(50f, 100f)]
    public float ySens = 70f; //чувствительность мыши по оси Y

    Quaternion center; //начальный поворот камеры

    void Start()
    {
        center = cam.localRotation; //получение начального поворота камеры
    }

    void Update()
    {
        //получение поворота вокруг оси X
        float mouseY = Input.GetAxis("Mouse Y") * ySens * Time.deltaTime;
        Quaternion yRot = cam.localRotation * Quaternion.AngleAxis(mouseY, -Vector3.right);
        //запрет поворота на более чем 90 градусов
        if (Quaternion.Angle(center, yRot) < 90f)
            cam.localRotation = yRot; //поворот камеры

        //получение поворота вокруг оси Y
        float mouseX = Input.GetAxis("Mouse X") * xSens * Time.deltaTime;
        Quaternion xRot = player.localRotation * Quaternion.AngleAxis(mouseX, Vector3.up);

        player.localRotation = xRot; //поворот игрока
    }
}
```

Передайте в скрипт ссылки на объект игрока и камеру игрока:



Для того, чтобы зафиксировать курсор мыши в центре экрана, создайте глобальную переменную:

```
public class Looking : MonoBehaviour
{
    public static bool cursorLock = true; //включение/выключение курсора
```

А затем, в методе Update, опишите следующие условия:

```
        if (cursorLock) //если курсор заблокирован
        {
            Cursor.lockState = CursorLockMode.Locked; //блокирование перемещения курсора
            Cursor.visible = false;                   //скрытие курсора
            if (Input.GetKeyDown(KeyCode.Escape))
                cursorLock = false;
        } else
        {
            Cursor.lockState = CursorLockMode.None; //разблокирование курсора
            Cursor.visible = true;                  //раскрытие курсора
            if (Input.GetKeyDown(KeyCode.Escape))
                cursorLock = true;
        }
    }
```

Бег

Для реализации “бега”, модифицируйте скрипт перемещения следующим образом:

```
public class Movement : MonoBehaviour
{
    [Range(50f, 200f)]
    public float walkSpeed = 100f; //скорость хотьбы
    [Range(100f, 500f)]
    public float runSpeed = 200f;  //скорость бега

    private float mSpeed; //скорость перемещения

    Rigidbody rb; //ссылка на физическую модель объекта

    void Start()
    {
        mSpeed = walkSpeed;
        rb = GetComponent<Rigidbody>(); //получение ссылки на физическую модель
    }

    void FixedUpdate()
    {
        //sprint = true если зажат левый или правый Shift
        bool sprint = (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.LeftShift));

        float xMove = Input.GetAxisRaw("Horizontal");  //проверка нажатия кнопок a, d, left, right
        float zMove = Input.GetAxisRaw("Vertical");    //проверка нажатия кнопок w, s, up, down

        if (sprint == true && zMove > 0) mSpeed = runSpeed; //если движение вперёд и зажат Shift - скорость бега
        else mSpeed = walkSpeed; //иначе - скорость ходьбы

        Vector3 dir = new Vector3(xMove, 0, zMove); //получение направления движения в плоскости X|Z
        dir.Normalize();                            //нормализация вектора движения
        //вектор скорости объекта = направление движения * скорость * время с прошедшего вызова этой функции
        Vector3 v = transform.TransformDirection(dir) * mSpeed * Time.fixedDeltaTime;
        v.y = rb.velocity.y; //восстановление смещения по оси Y
        rb.velocity = v;
    }
}
```

В дополнении к скорости перемещения, можно добавить эффект камеры для бега. Для этого, измените скрипт как показано ниже:

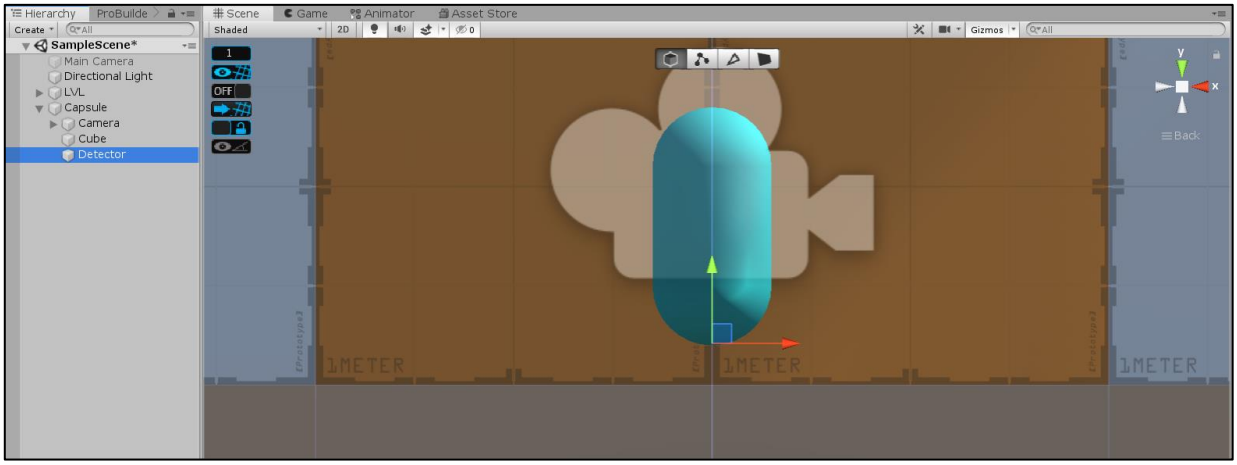
```
public Camera cam;
float baseFOV;                  //начальный угол обзора камеры
public float sprintFOV = 1.25f; //изменение угла обзора при беге


void Start()...

void FixedUpdate()
{
    //sprint = true если зажат левый или правый Shift
    bool sprint = (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.LeftShift));

    float xMove = Input.GetAxisRaw("Horizontal");  //проверка нажатия кнопок a, d, left, right
    float zMove = Input.GetAxisRaw("Vertical");    //проверка нажатия кнопок w, s, up, down

    if (sprint == true && zMove > 0) //если движение вперёд и зажат Shift - скорость бега
    {
        mSpeed = runSpeed;
        //плавное изменение угла обзора с начального, до начального * на изменение с течением времени
        cam.fieldOfView = Mathf.Lerp(cam.fieldOfView, baseFOV * sprintFOV, Time.fixedDeltaTime * 8f);
    }
    else //иначе - скорость ходьбы
    {
        mSpeed = walkSpeed;
        //плавное изменение угла обзора с модифицированного, до начального с течением времени
        cam.fieldOfView = Mathf.Lerp(cam.fieldOfView, baseFOV, Time.fixedDeltaTime * 8f);
    }
```

Прыжок

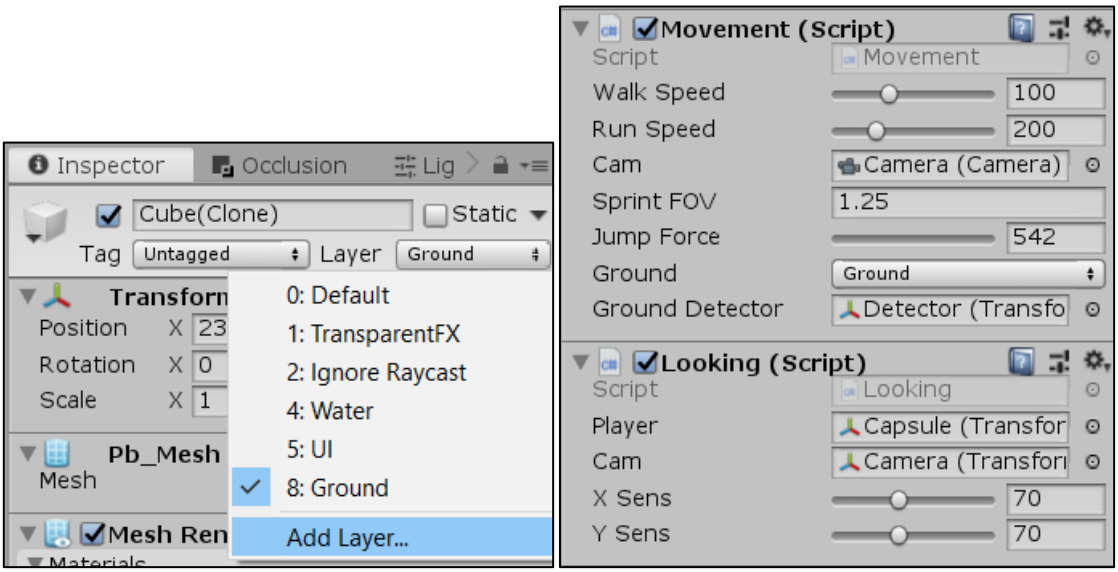
Реализовать прыжок можно проверив, находится ли игрок на поверхности и применив к нему силу по оси Y:

```
public float sprintFOV = 1.25f; //изменение угла обзора при беге

[Range(1000f, 2000f)]
public float jumpForce = 1000f;  //сила прыжка

public LayerMask ground;         //ссылка на слой, который будет считаться землёй
public Transform groundDetector; //объект, отвечающий за определения расстояния до земли

void Start()...

void FixedUpdate()
{
    //проверка расстояния до земли
    bool groundCheck = Physics.Raycast(groundDetector.position, Vector3.down, 0.1f, ground);
    bool jump = Input.GetKey(KeyCode.Space) && groundCheck; //прыжок, только если игрок находится на земле

    if (jump == true) rb.AddForce(Vector3.up * jumpForce); //применение силы по вектору вверх

    //sprint = true если зажат левый или правый Shift
    bool sprint = (Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.LeftShift));
```

В качестве GroundDetector, можно использовать пустой объект, расположенный в нижней точке модели игрока:



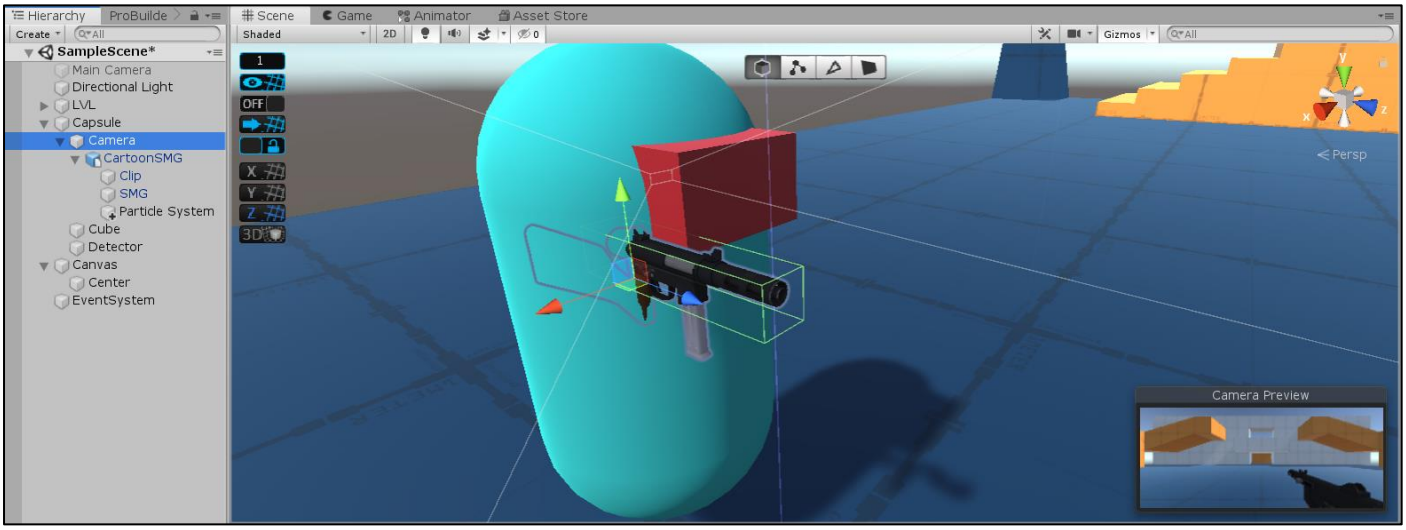
Для того, чтобы обозначить поверхность как “землю”, добавьте новый слой и назначьте его для всех объектов, с поверхности которых можно осуществлять прыжок:



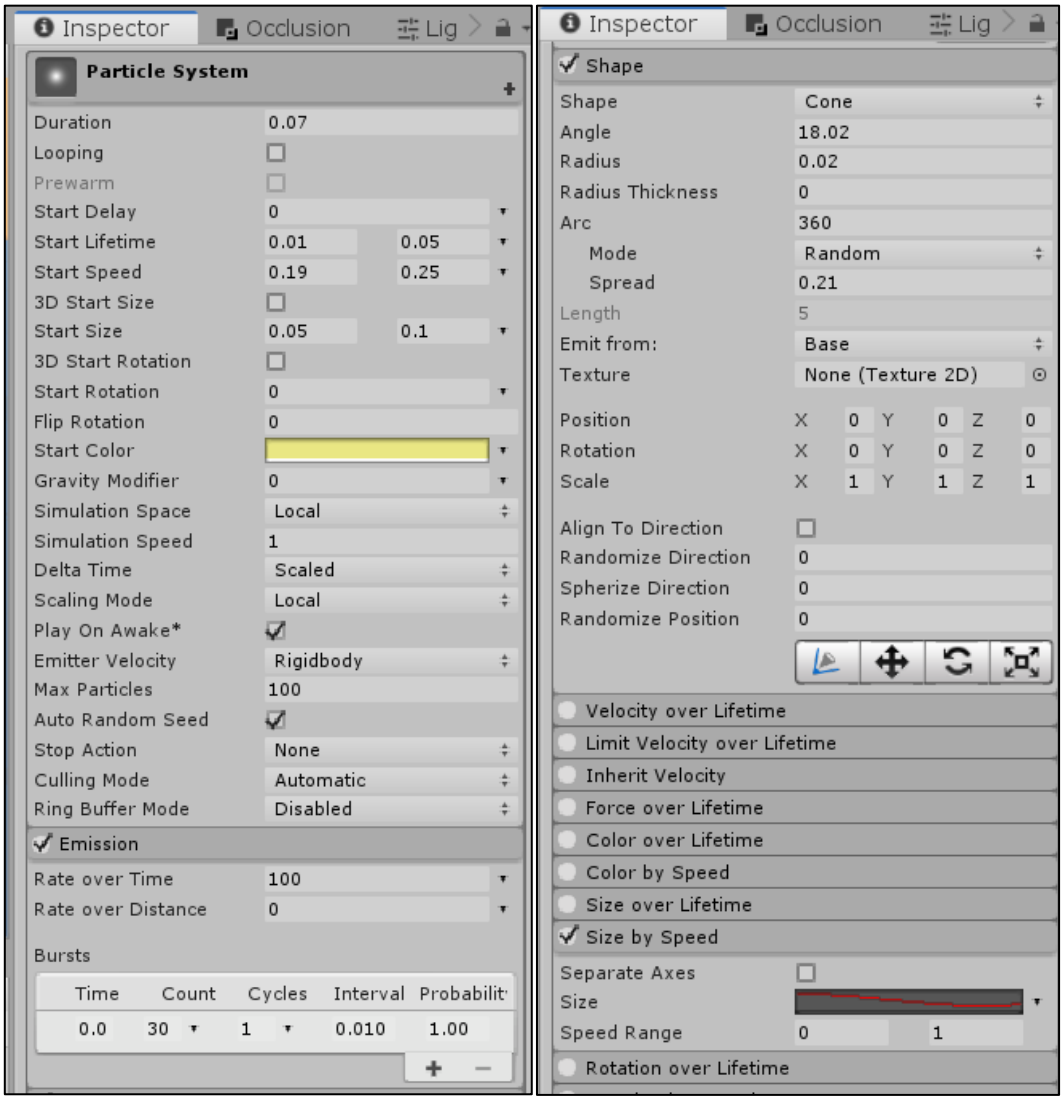
Не забудьте передать все необходимые параметры в скрипты движения и поворотов камеры.

Стрельба

Перед началом работ над механикой стрельбы, добавьте к объекту камеры игрока модель оружия. Прикрепите к модели систему частиц, которая будет отвечать за выстрел. Добавьте в сцену изображение, разместите его в центре экрана и загрузите картинку прицела:

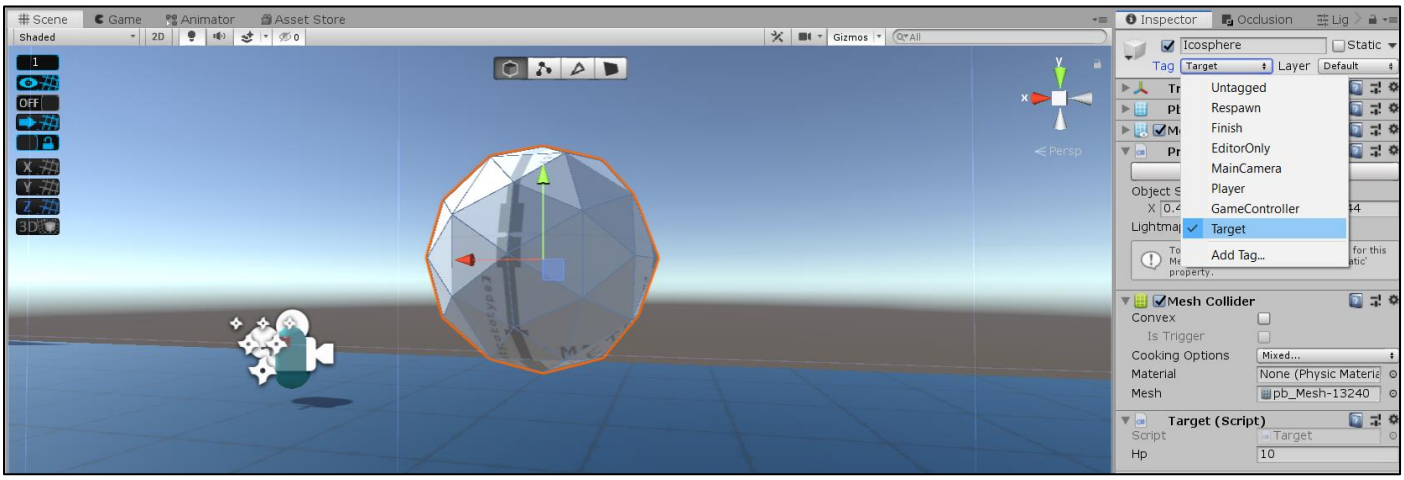


Настройки системы частиц могут выглядеть следующим образом:



Ключевыми моментами являются настройки формы и стартовые параметры частиц.

Перед реализацией стрельбы, добавьте в сцену объект “Цель” и создайте для него специальный тэг:



Добавьте к объекту скрипт, описывающий его здоровье, метод получения урона и уничтожения:

```
public class Target : MonoBehaviour
{
    public float hp = 10f; //здоровье цели

    public void Hit(float damage) //функция, вызываемая при попадании по цели
    {
        hp -= damage;           //уменьшение здоровья
        if (hp <= 0) Death();
    }

    void Death() //описание действий при уничтожении цели
    {
        Destroy(gameObject);
    }
}
```

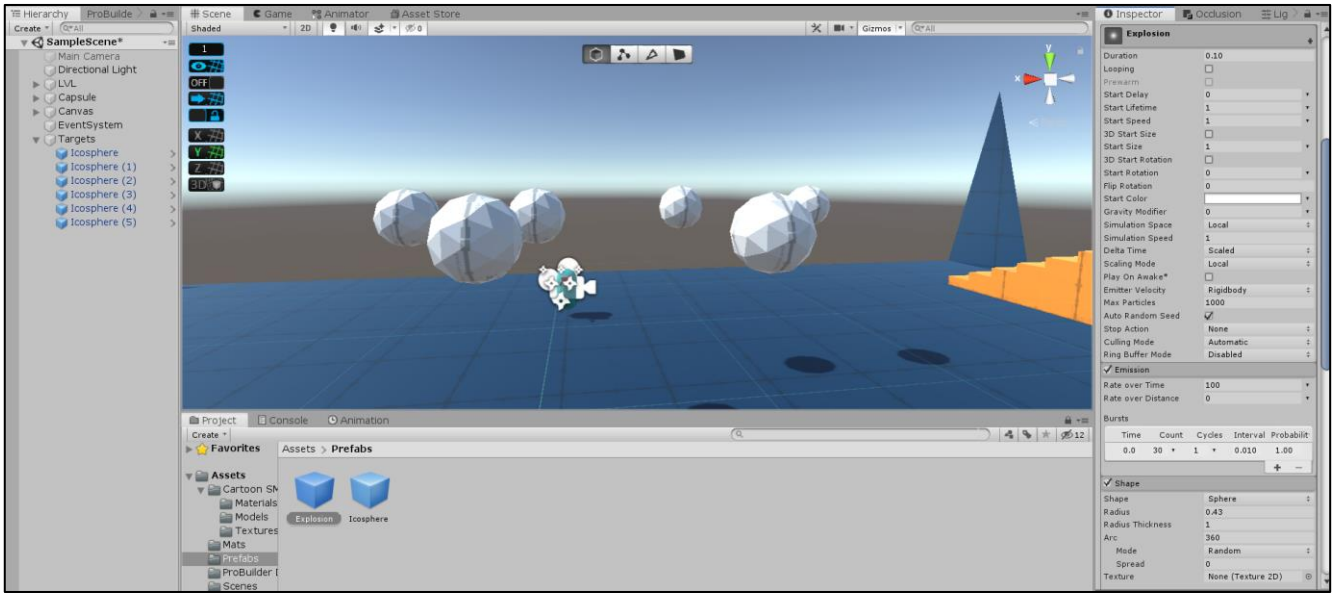
После чего, добавьте к модели оружия скрипт, описывающий механику стрельбы:

```
public class Gun : MonoBehaviour
{
    public float damage = 10f;   //урон оружия
    public float range = 1000f;  //дальность действия оружия

    public Camera cam;           //ссылка на камеру игрока
    public ParticleSystem flash; //ссылка на систему частиц отвечающую за "вспышку"

    void Update()
    {
        if (Input.GetButton("Fire1")) Shoot();  //если нажата кнопка стрельбы - вызвать метод стрельбы
    }

    void Shoot()    //метод стрельбы
    {
        flash.Play();   //воспроизвести вспышку

        RaycastHit hit;
        if (Physics.Raycast(cam.transform.position, cam.transform.forward, out hit, range)) //если луч из камеры во что-то попал
        {
            if (hit.transform.CompareTag("Target")) //если это что-то имеет тэг "Цель"
            {
                Target t = hit.transform.GetComponent<Target>();    //получить доступ к скрипту цели
                t.Hit(damage);  //вызвать метод получения урона
            }
        }
    }
}
```

В результате, при попадании в объект луча из камеры, он будет уничтожаться.

Уничтожение цели

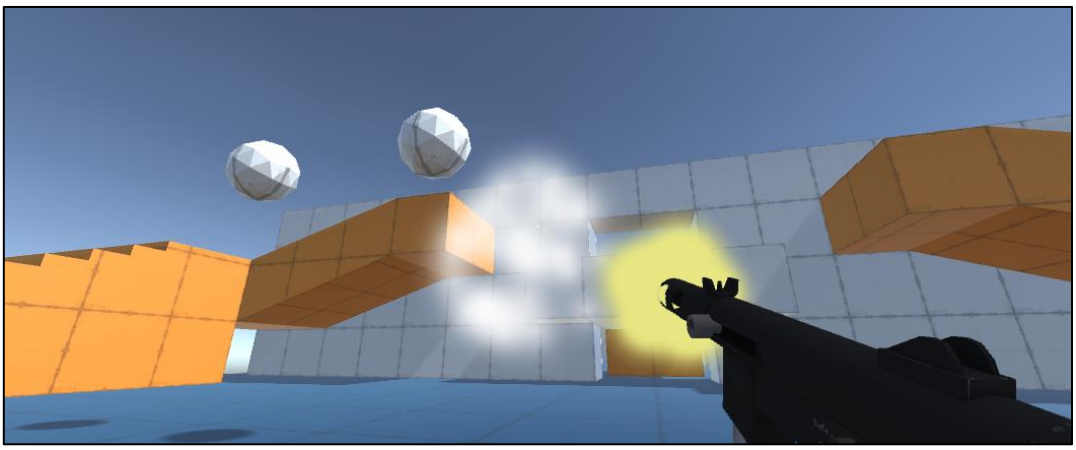
Как правило, уничтожаемые объекты оформляются в виде префаба, а в момент их уничтожения проигрывается какая-нибудь анимация. Создайте префаб для системы частиц, реализующей взрыв:



А затем, дополните скрипт объекта “Цель”:

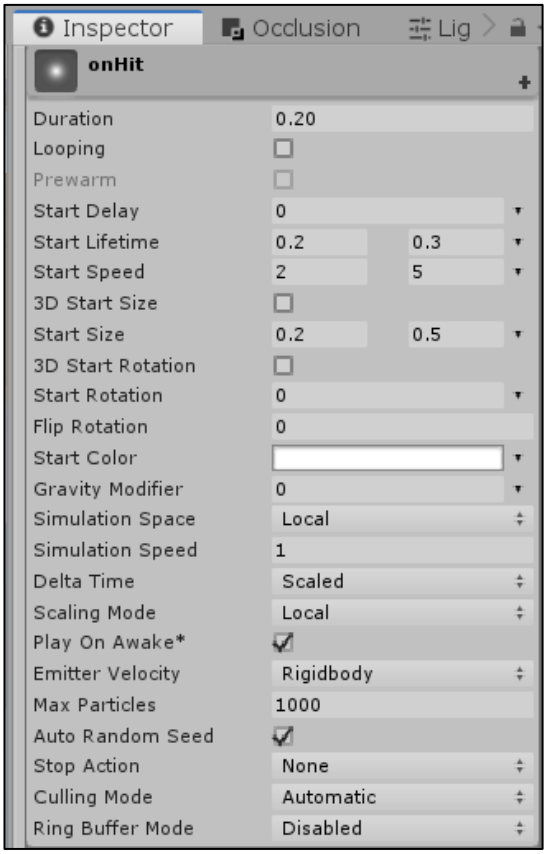
```
public class Target : MonoBehaviour
{
    public float hp = 10f;            //здоровье цели
    public ParticleSystem explosion; //ссылка на эффект взрыва

    public void Hit(float damage) //функция, вызываемая при попадании по цели
    {
        hp -= damage;            //уменьшение здоровья
        if (hp <= 0) Death();
    }

    void Death() //описание действий при уничтожении цели
    {
        //создание копии эффекта взрыва и размещение её по координатам цели
        ParticleSystem exp = Instantiate(explosion, transform.position, transform.rotation);
        exp.Play(); //воспроизведение анимации взрыва

        Destroy(exp.gameObject, 1f); //уничтожение системы частиц через одну секунду
        Destroy(gameObject);         //уничтожение цели
    }
}
```

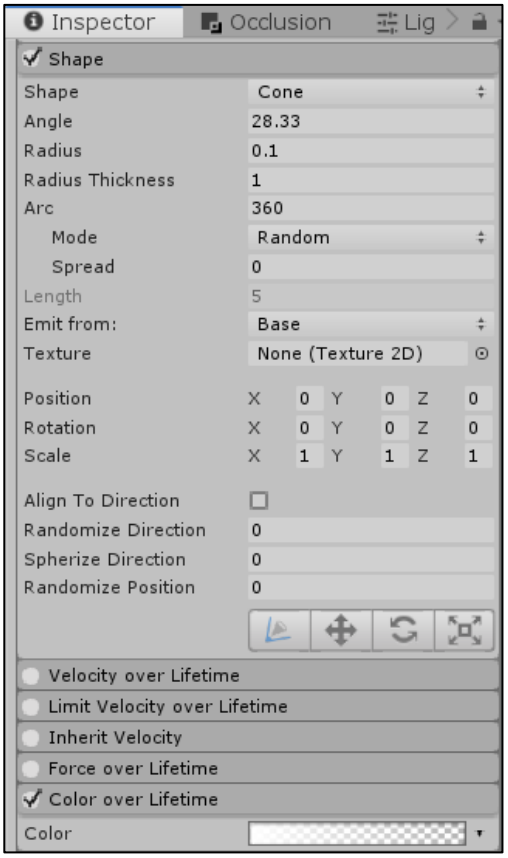
Результат выстрела и попадания по объекту:



Эффект попадания

Для обозначения мест попадания выстрелов, можно использовать систему частиц подобного вида:





После чего, добавьте следующий код в скрипт оружия:

```
public class Gun : MonoBehaviour
{
    public float damage = 10f;   //урон оружия
    public float range = 1000f;  //дальность действия оружия

    public Camera cam;           //ссылка на камеру игрока
    public ParticleSystem flash; //ссылка на систему частиц отвечающую за "вспышку"

    public ParticleSystem onHit; //ссылка на систему частиц отвечающую за "попадание"

    void Update()...

    void Shoot()    //метод стрельбы
    {
        flash.Play();   //воспроизвести вспышку

        RaycastHit hit;
        if (Physics.Raycast(cam.transform.position, cam.transform.forward, out hit, range)) //если луч из камеры во что-то попал
        {
            if (hit.transform.CompareTag("Target")) //если это что-то имеет тэг "Цель"
            {
                Target t = hit.transform.GetComponent<Target>();    //получить доступ к скрипту цели
                t.Hit(damage);  //вызвать метод получения урона
            }

            //создание и воспроизведение эффекта выстрела в точке попадания
            ParticleSystem hitEffect = Instantiate(onHit, hit.point, Quaternion.LookRotation(hit.normal));
            hitEffect.Play();

            Destroy(hitEffect.gameObject, 1f); //уничтожение эффекта через одну секунду
        }
    }
}
```

Ограничение частоты выстрелов

Для того, чтобы ограничить частоту выстрелов в секунду, добавьте следующий код к скрипту оружия:

```
public class Gun : MonoBehaviour
{
    public float damage = 10f;    //урон оружия
    public float range = 1000f;   //дальность действия оружия

    public float fireRate = 10f; //скорость стрельбы (10 выстрелов в секунду)
    public float nextShot = 0f;  //время до следующего выстрела

    public Camera cam;           //ссылка на камеру игрока
    public ParticleSystem flash; //ссылка на систему частиц отвечающую за "вспышку"

    public ParticleSystem onHit; //ссылка на систему частиц отвечающую за "попадание"

    void Update()
    {
        //если нажата кнопка стрельбы и пришло время следующего выстрела - вызвать метод стрельбы
        if (Input.GetButton("Fire1") && Time.time >= nextShot)
        {
            nextShot = Time.time + 1 / fireRate; //рассчитатать время до следующего выстрела
            Shoot();
        }
    }

    void Shoot()    //метод стрельбы...
}
```

Задание:

Создать интерактивное приложение трёхмерной графики, содержащее следующие механики:

1. Возможность перемещения по сцене в режиме от первого лица.
2. Возможность производить выстрел из оружия.
3. При выстреле, из оружия должна вылетать модель гильзы. Гильзы должны пропадать спустя некоторое время.
4. При пересечении определённых областей, в сцене должны появляться цели, случайного размера и в случайной позиции. Спустя случайный промежуток времени, цели должны исчезать.
5. При попадании в цель из оружия, должны начисляться очки, в зависимости от размера и времени жизни цели. Очки должны отображаться на экране.