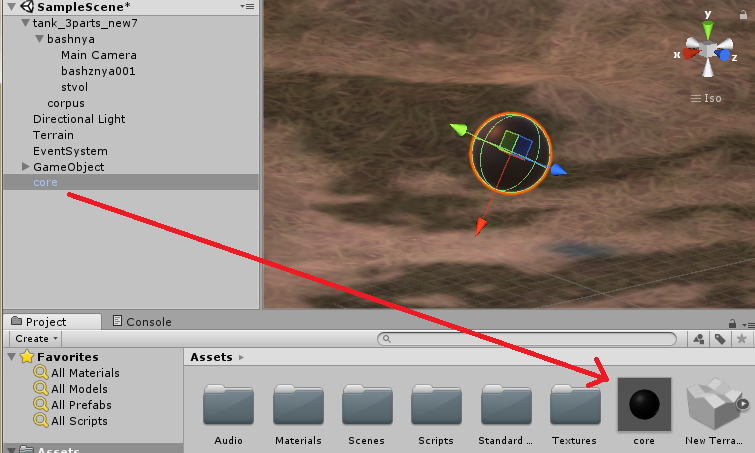
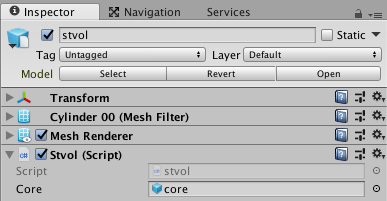
Лабораторная работа №3. Реализация выстрелов снарядами, обработка попадания элементов в цель

1. В созданном ранее проекте Unity3D с управлением движения танка добавить новый объект – снаряд **core** в форме сферы **Sphere** соответствующих размеров для ствола танка.
2. Добавить к ядру *core* «физику» командой **Add Component/Physics/Rigidbody**. При необходимости добавить ядру массу в инспекторе в свойствах **Rigidbody**, а также поставить галочку **isKinematic**, чтобы снаряд мог воздействовать на объекты на сцене, а на него воздействие могло оказываться только из кода.
3. Создать **prefab** для данного снаряда, перетянув его со сцены в папку Assets, и удалить его со сцены.



1. Открыть компонент **Script** *ствола* и добавить в его программный код публичную переменную: типа **GameObject** для ссылки на **Prefub** cнаряда ***core***. Перетащите префаб снаряда на скрипт ствола на сцене.



1. В метод **Update()** добавить обработку условия нажатия определенной клавиши на клавиатуре (например, пробел) или клик мыши для выполнения выстрела. В данном условии сначала высчитать точку, в которой нужно создать снаряд (в стволе со смещением вперед): **Vector3 forwardOfStvol = transform.position + transform.TransformDirection(Vector3.forward \*4f)**, где 4f – длина ствола (необходимо выставить нужную величину в соответствии с вашей моделью танка). Далее создаем сам снаряд из префаба в этой точке: **GameObject newcore = Instantiate(core, forwardOfStvol, transform.rotation)**; Если снаряд летит в ненужную сторону, добавьте принудительный поворот так, как повернут ствол **newcore.transform.LookAt(forwardOfStvol);.**
2. Запустить проект и проверить, что при нажатии пробела создаются снаряды, в зависимости от поворота ствола, но пока эти снаряды не летят и не взрываются.
3. Создать новый скрипт для управления поведением снаряда (**core.cs**) и поместить данный скрипт компонентом на *префаб* ядра в ассетах.
4. В методе **Update()** добавить постоянно перемещение снаряда вперед: **transform.position += transform.TransformDirection(Vector3.forward \* coreSpeed);**, где coreSpeed – переменная скорости, которую нужно объявить выше в данном скрипте
5. Запустить игру и проконтролировать соответствие полета снарядов из разных точек сцены, различных поворотах танка и его башни. Можно заметить, что снаряды постоянно находятся на сцене и не удаляются, даже если отлетели уже на большое расстояние и по-прежнему остаются в памяти, что плохо для производительности. Чтобы исправить данный факт, в код снаряда добавим корутину вне метода **Update()**, которая будет уничтожать объект, на котором находится через 5 секунд:

**IEnumerator coreLife()** // объявляется как метод

**{**

**yield return new WaitForSeconds(5f);** // ждем 5 секунд

**Destroy(gameObject);** // уничтожаем объект

**}**

А вызывать ее нужно в методе **Start(),** т.е. сразу после создания ядра**:** **StartCoroutine(coreLife());**

1. Добавить на сцену 3 различных цели-объекта (например, куб, сфера), наложить на них материалы и текстуры, задать им соответствующие размеры и расположить в разных местах сцены на ее поверхности. Добавить объектам-целям «физику», с помощью добавки компонента **Rigidbody**.
2. Добавьте скрытие снаряда при столкновении с объектами добавлением метода обработки коллизии:

**private void OnCollisionEnter(Collision collision)**

**{**

**gameObject.GetComponent<MeshRenderer>().enabled = false;**

**}**

1. Обработка попадания элементов в цель происходит при добавлении хотя бы на один объект компонента **Physics/Rigidbody** и на оба объекта коллайдера. Добавим новый скрипт с именем **trigger** на объекты-цели.
2. В программном коде нового скрипта вместо пустых функций **Start()** и **Update()** определить новую функцию обработки попадания одного объекта в коллайдер другого, например, **void** **OnCollisionEnter(Collision col)**, где переменная **col** содержит ссылку на коллайдер другого объекта, с которым произошло столкновение.
3. Задать в функции **OnCollisionEnter** проверяемое условие столкновения, например, **col.gameObject.tag == "core" || col.gameObject.tag == "Player",** где **core** или **Player** – тэг объектов, над которыми нужно производить действия. Добавьте теги **core** к ядру и **Player** к танку.
4. Задать значение какого-либо свойства для цели, которое должно поменяться после столкновения, например, цели становится окрашены в разные оттенки красного: **Color color = new Color(Random.Range(0f,1f), 0,0)**, где **Random.Range** возвращает случайное значение в заданном диапазоне от 0 (белый) до 1 (черный) с плавающей точкой.
5. Также при попадании присваиваем объектам-целям заданный цвет **gameObject.GetComponent<Renderer>().material.color = color;**
6. Добавьте возврат цвета к белому через 3 секунды. Для этого необходимо создать корутину в скрипте **trigger.cs**: **IEnumerator wait3seconds ()**. В теле корутины написать **yield return new WaitForSeconds(3f)**, где 3 – это время ожидания в секундах. За этой строчкой добавить **gameObject.GetComponent<Renderer>().material.color = new Color(1,1,1);** – окрашиваем переданный в метод объект в белый цвет. Осталось добавить вызов корутины из условия столкновения в методе *OnCollisionEnter*: **StartCoroutine(wait3seconds());**
7. Запустить игру и проконтролируйте реакцию целей на попадание снарядов и «наезд» на них танка (на корпусе танка должен быть коллайдер).