Лабораторная работа №13. Создание ботов, обработка их поведения, добавление эффектов

Раздел I. Создание бота

1. Подготовить префаб для вражеского танка, из уже имеющегося танка.
2. Добавить на первый танк на сцене (не на префаб) большой (радиус ~40) сферический триггерный коллайдер, для обнаружения игрока другими танками (рисунок 1).

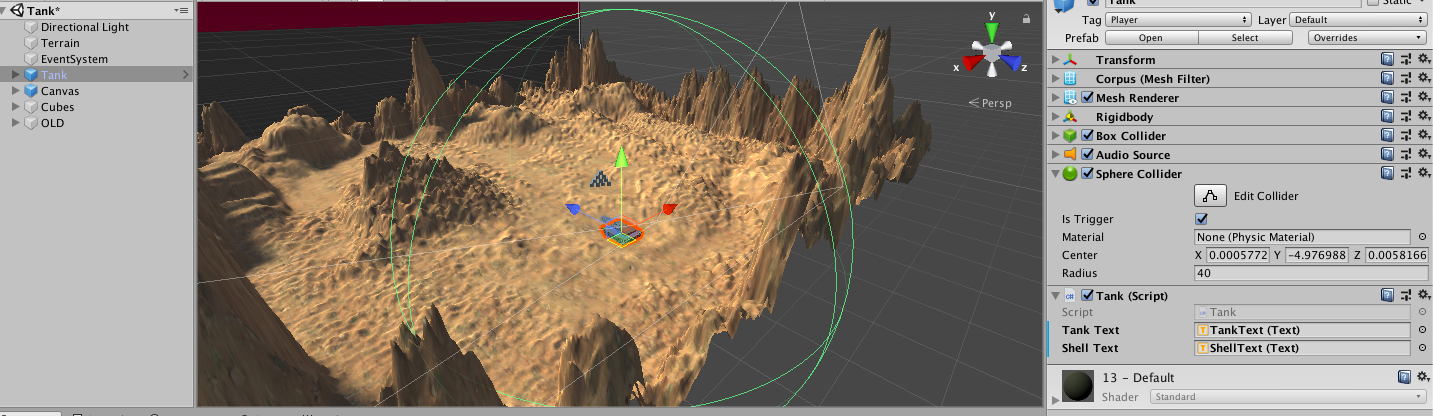


Рисунок 1 – Сферический триггерный коллайдер главного танка

1. Создайте новый скрипт **«Bot»**. Объявить в нем:
   * **float** переменные **moveSpeed** (скорость передвижения), **rotSpeedTank** (скорость поворота) и **rotSpeedTurret** (скорость поворота башни);
   * переменные типа **Transform** для управления его башней **turret** и стволом **cannon** и переменную типа **GameObject** core для ссылки на префаб снаряда, эти переменные необходимо проинициализировать с помощью **[SerializeField]**;
   * булеву переменную **bool canShoot = true**, для определения, может ли в данный момент танк произвести выстрел.
2. Добавить в скрипт бота метод **OnTriggerStay(Collider other)**, т.к. выполнять какие-либо действия он должен при нахождении в большом триггере игрока. Внутри этого метода определить, что танк находится именно в триггере игрока: **if (other.CompareTag("Player"))**.
3. После определения, что танк находится в триггере игрока, необходимо вычислить некоторые значения:
   * дистанцию до игрока:

float distance = Vector3.Distance(other.transform.position, transform.position);

* + определить вектор направления:

Vector3 relativePos = (other.transform.position - transform.position);

* + по вектору направления определить поворот:

Quaternion newRot = Quaternion.LookRotation(relativePos);

1. При дистанции больше 20, бот должен двигаться и поворачиваться в направлении к игроку:

if (distance > 20)

{  
            transform.rotation = Quaternion.Slerp(transform.rotation, newRot, Time.deltaTime \* rotSeedTank);  
            transform.position = Vector3.Lerp(transform.position, other.transform.position, Time.deltaTime \* moveSpeed);  
         }

1. Для определения, повернута ли уже башня на игрока, используем метод бросания лучей. В независимости от расстояния, но при нахождении в триггере поворачиваем башню в сторону нахождения игрока:

turret.rotation = Quaternion.Slerp(turret.rotation, newRot, Time.deltaTime \* rotSpeedTurret);

1. Для определения, повернута ли уже башня на игрока, используем метод бросания лучей и если луч попал в коллайдер игрока производиться выстрел, а именно запускается корутина:

RaycastHit hit;  
            if (Physics.Raycast(turret.position, turret.TransformDirection(Vector3.forward), out hit))  
            {  
                if (hit.transform.CompareTag("Player") && canShoot)  
                    StartCoroutine(botShoot());  
            }

1. Напишите корутину **botShoot()** для реализации выстрела ботом:

IEnumerator botShoot()  
    {  
        canShoot = false;  
        Vector3 forwardofstvol = cannon.position + cannon.TransformDirection(Vector3.forward \* 4f);  
        GameObject newShell = Instantiate(shell, forwardofstvol, cannon.rotation);  
        newShell.transform.LookAt(forwardofstvol);  
        yield return new WaitForSeconds(3f);   
        canShoot = true;   
    }

1. Поместите скрипт **«Bot»** на префаб бота-танка, предварительно разместив его на сцене. Добавьте на него тег **«bot»**. Перетяните нужные переменные на скрипт и нажмите кнопку **Apply**. Не забудьте удалить с бота камеру, все старые скрипты для управления танком, его башней и стволом, оставив только на корпусе танка новый скрипт **«Bot»** (рисунок 2).

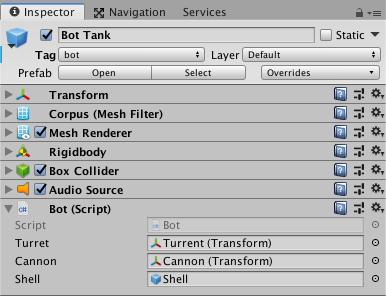


Рисунок 2 – Размещение скрипта на боте

1. Добавьте возможность уничтожения вражеского танка или танка игрока, создав новый скрипт **«Heаlth»**, который необходимо разместить как на боте, так и на главном танке. Создайте в нём метод **OnCollisionEnter (Collision collision)**. Внутри этого метода определите по тэгу коллизию со снарядом, и если текущий объект бот, то необходимо отнять жизнь, а если все жизни использованы – удалить бота со сцены. А также если все боты на сцене удалены, вывести сообщение о победе игрока:

if (gameObject.CompareTag("bot"))  
            {  
                if (--life < 1)   
                {  
                    if (GameObject.FindGameObjectsWithTag("bot").Length <= 1)  
                    {  
                        // сообщение о победе  
                    }  
                    Destroy(gameObject);  
                }   
            }

1. Аналогично, если все жизни игрока были истрачены, вывести сообщение о проигрыше и убрать тег **Player** с персонажа, задав ему любой другой, чтобы у ботов пропала цель и они остановились.

 if (gameObject.CompareTag("Player"))  
            {  
                if (--life < 1)  
                {  
                    gameObject.tag = "Untagged";  
                    // сообщение о проигрыше  
                }  
            }

Раздел II. Добавление эффектов

1. Добавьте взрыв ядра при попадании. Для этого в скрипте **«Shell»** добавьте публичную переменную **explosion** типа **GameObject** на префаб взрыва (префаб взрыва находится в стандартных ассетах, но можно скачать и из дополнительных источников).
2. В методе **OnCollisionEnter()** скрипта **«Shell»** добавьте код для создания экземпляра взрыва в точке, где находится ядро при попадании.

Instantiate(explosion, gameObject.transform.position, Quaternion.identity);

1. Добавьте сам взрыв на префаб снаряда в инспекторе (рисунок 3):

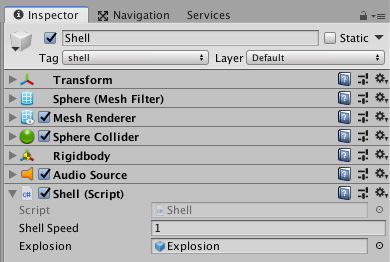


Рисунок 3 – Размещение взрыва на снаряд

1. Добавьте на сцену один или несколько экземпляров бота и проверьте, что при попадании в триггер игрока он движется и поворачивается в его направлении, также вращается башня и он стреляет в направлении игрока. При попадании во вражеский танк 3 раза, он должен удалиться со сцены. Ядро при попадании должно взрываться.
2. Создайте пустой объект на сцене **GameObject/Create Empty** и измените имя на **Laser**. Расположите его на конце ствола вашего танка и сделайте дочерним элементом ствола (рисунок 4). Добавьте на него текстуру **laser.jpg** и измените её **Shader** выбрав **Particles/Priority Addictive**. Добавьте на данный объект компонент **Line Render** (можете изменить Width и попробовать другие настройки). Данный компонент будет отображаться только в режиме **Game**.

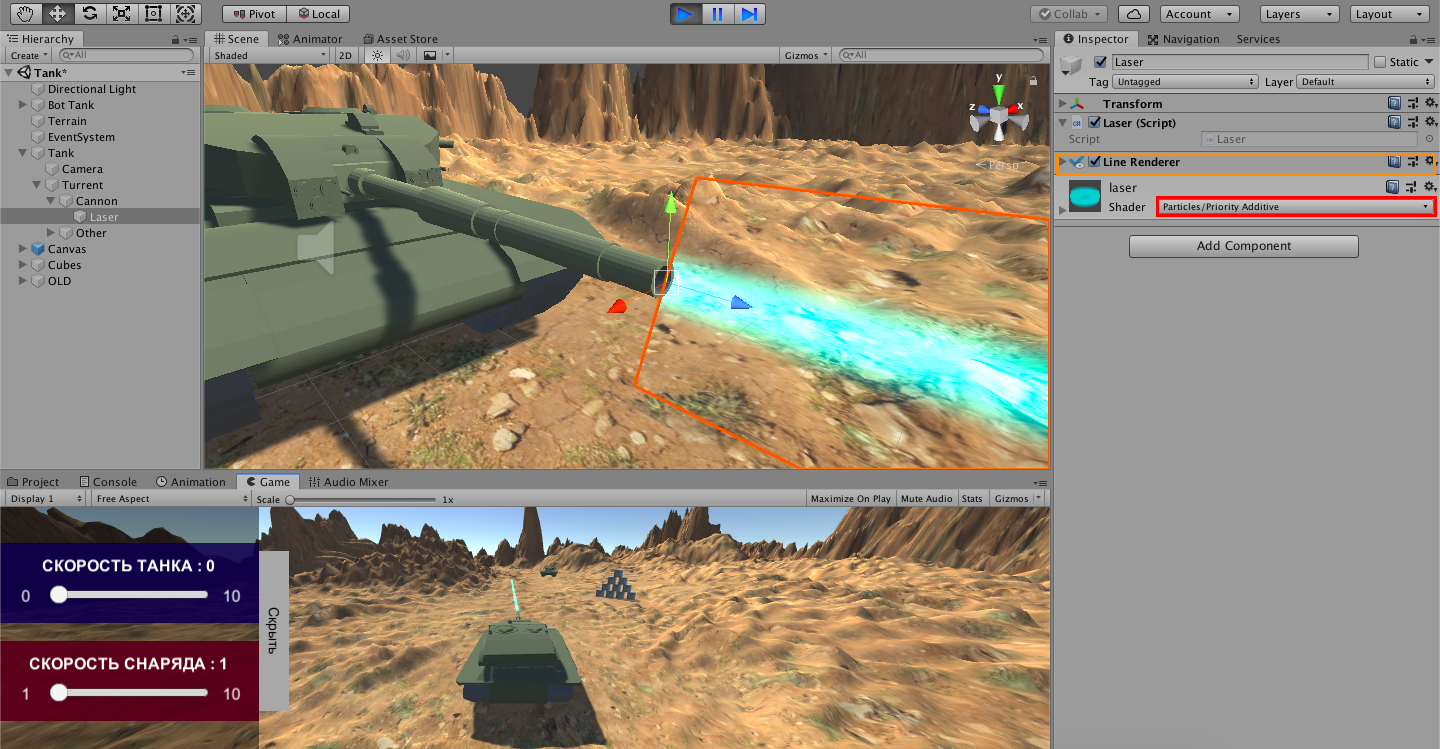


Рисунок 4 – Размещение объекта Laser

1. Создайте скрипт **«Laser»** и разместите его на объекте **«Laser».** В нёмсоздайте переменную **lr** типа **LineRender**. И в методе **Start()** проинициализируйте данную переменную.
2. В методе **Update()** необходимо**:**
   * установить начальную точку;
   * создать переменную для хранения информации о столкновении;
   * создать луч из начальной точки и выпустить его вперёд, а также записать в переменную информацию об объекте, с которым было столкновение;
   * если произошло столкновение с каким либо коллайдером, установить конечную точку луча в место столкновения;
   * если не происходило столкновение то выпускать луч длиной 5000 единиц.

lr.SetPosition(0, transform.position);    
        RaycastHit hit;  
        if (Physics.Raycast(transform.position, transform.forward, out hit))  
        {  
            if (hit.collider)    
            {  
                lr.SetPosition(1, hit.point);    
            }  
        }  
        else lr.SetPosition(1, transform.forward \* 5000);

**20. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ.** Напишите скрипт **«Timer»**, который будет отображать оставшееся время (например, минута) за которое игроку необходимо победить всех вражеских танков. В случае успешного выполнения задания за выданное время или в случае проигрыша (время закончилось или главного игрока убили) должно появиться соответствующее текстовое сообщение.