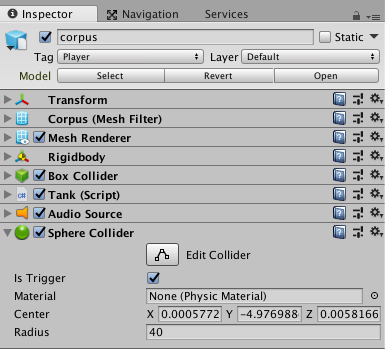
Лабораторная работа №4\* (дополнительная). Создание ботов, обработка их поведения, добавление эффектов

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



1. Создать новый скрипт **bot.cs**. Объявить в нем **float** переменные **moveSpeed** (скорость передвижения), **rotSeedTank** (скорость поворота), **rotSpeedBash** (скорость поворота башни), **speedCore** (скорость вражеского снаряда). Переменные типа **Transform (bash и stvol)** для управления его башней и стволом. Переменную типа **GameObjec**t core для ссылки на префаб снаряда, **bool canShoot = true**, для определения, может ли в данный момент танк произвести выстрел. И **int life = 3**. Для определения кол-ва попаданий в танк, после который он уничтожится.
2. Проинициализировать переменные в методе **Start()** или через инспектор.
3. 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.tag == "Player")**, предварительно добавив к игроку данный тэг.
4. После определения, что танк находится в триггере игрока, нужно вычислить некоторые значения:
5. дистанцию до игрока:

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

1. Определить вектор направления:

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

1. И по вектору направления определить поворот:

**Quaternion newRot = Quaternion.LookRotation(relativePos);**

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

**if (distance > 20)  
{** // поворачиваем танк с помощью интерполяции между текущим поворотом и требуемым

// направлением в сторону игрока с заданной скоростью **transform.rotation = Quaternion.Slerp(transform.rotation, newRot, Time.deltaTime \* rotSeedTank);**

//также с помощью интерполяции двигаем танк в сторону игрока, меняя значения по х и z осям  
**transform.position = new Vector3(Mathf.Lerp(transform.position.x, other.transform.position.x, Time.deltaTime \* moveSpeed), transform.position.y, Mathf.Lerp(transform.position.z, other.transform.position.z, 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 игрока: **bash.rotation = Quaternion.Slerp(bash.rotation, newRot, Time.deltaTime \* rotSpeedBash);**
2. Для определения, повернута ли уже башня на игрока, используем метод бросания лучей:  
   **RaycastHit hit;** //переменная для записи объекта попадания  
   //выпускаем луч из башни в направлении относительно ее – вперед.  
   **if (Physics.Raycast(bash.position, bash.TransformDirection(Vector3.forward), out hit)) {**

**if ((hit.transform.tag == "Player") && canShoot)**

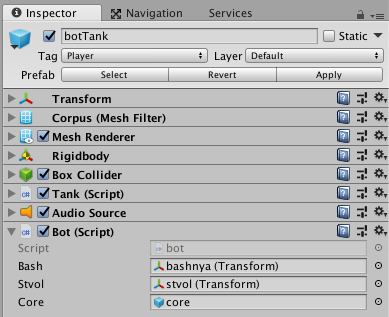
//если луч попал в коллайдер игрока и можно выстрелить  
**StartCoroutine(botShoot());** //запускаем корутину для выстрела вражеского танка

**}**

1. Написать корутину для выстрела:

**IEnumerator botShoot() {  
        canShoot = false;** //указываем, что теперь этот танк стрелять не может  
                          //определяем координту для создания снаряда  
**Vector3 forwardofstvol = stvol.position + stvol.TransformDirection(Vector3.forward \* 4f);**//создаем снаряд из префаба в требуемой координате  
**GameObject newcore = Instantiate (core, forwardofstvol, stvol.rotation);**        //поворачиваем ядро в сторону поворота ствола  
**newcore.transform.LookAt(forwardofstvol);  
        yield return new WaitForSeconds(3f);** //ждем 3 секунды (время перезарядки) **canShoot = true;** //указываем, что можем сделать новый выстрел  
**}**

1. Добавить возможность уничтожения вражеского танка, добавив метод **OnCollisionEnter (Collision collision)**. Внутри этого метода, определить по тэгу коллизию со снарядом: **if (collision.gameObject.tag == "core")**, а в этом условном блоке отнять счетчик жизни: **life--;** и в случае утраты всех жизней, уничтожить вражеский танк: **if (life<1) Destroy(gameObject);**
2. Поместите скрипт **bot.cs** на префаб бота-танка, предварительно разместив его на сцене. Перетяните нужные переменные на скрипт и нажмите кнопку **Apply**. Не забудьте удалить с бота все старые скрипты для управления танком, оставив только на корпусе танка новый скрипт **bot.cs**.



1. Добавить взрыв ядра при попадании. Для этого открыть скрипт ядра, добавить в него ссылку на префаб взрыва (префаб взрыва можно найти в стандартных ассетах) **GameObject explosion**; А в метод **OnCollisionEnter** добавить строчку для создания экземпляра взрыва в точке, где находится ядро при попадании  **Instantiate(explosion, gameObject.transform.position, Quaternion.identity);**
2. Добавить на сцену один или несколько экземпляров бота и проверить, 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.