**ПРОГРАММИРОВАНИЕ**

**АВТОНОМНЫХ ОБЪЕКТОВ**

**(БОТов)**

**БОТ - автономная копия объекта, созданная на основе префаба, которая управляется программно без участия пользователя.**

**Алгоритм создания бота на сцене**

1. **СОЗДАТЬ ПРЕФАБ** из уже имеющегося на сцене объекта танка с его скриптом.

При этом все экземпляры префаба, размещаемые на сцене (**боты** – танкипротивника), будут обладать функциональностью, определяемую исходным танком-игроком**.**

1. **ДОБАВИТЬ НА ИСХОДНЫЙ ТАНК-ИГРОКА СФЕРИЧЕСКИЙ ТРИГГЕРНЫЙ КОЛЛАЙДЕР** с радиусом, соответствующим расстоянию, на котором будет обнаруживаться танк-игрока другими танками-противника (ботами) при попадании их в этот коллайдер при движении по сцене танка-игрока и начнется «оживление» танка-бота с выполнением действий, предусмотренных созданным для него скриптом.
2. **СОЗДАТЬ НОВЫЙ СКРИПТ ДЛЯ ПРЕФАБА** – источника экземпляров танков-ботов (танков противника), при этом для экземпляров префаба на сцене старые скрипты танка-игрока нужно удалять.

***Алгоритм выполнения действий скрипта танка-бота:***

1. *Обнаружить попадание в коллайдер танка-игрока танка-бота*
2. *При попадании в коллайдер повернуть башню танка-бота в сторону танка-игрока*
3. *Определить, нацелен ли ствол танка-бота на танк-игрока*
4. *Произвести выстрел в танк-игрока с заданной с задержкой (на «перезарядку»)*
5. *Определить минимальную дистанцию, при которой начинается движение танка-бота в сторону танка-игрока*
6. *Начать движение танка-бота в сторону танка-игрока*
7. *Уничтожить танк-бот при попадании в него нескольких снарядов*

ПРИ ДОБАВЛЕНИИ СКРИПТА **«BOT»** НА ЭКЗЕМПЛЯР ПРЕФАБА БОТА-ТАНКА РЕКОМЕНДУЕТСЯ СОБЛЮДАТЬ ДЛЯ ИЗБЕЖАНИЯ ВОЗМОЖНЫХ «ГЛЮКОВ» СЛЕДУЮЩУЮ ОЧЕРЕДНОСТЬ ДЕЙСТВИЙ:

* 1. разместите экземпляр префаба танка-бота на сцене;
  2. «перетяните» в **Assets** скрипт **Bot** на префаб танка;
  3. добавьте на экземпляр префаба тег **«goal»;**
  4. свяжите объектные переменные для башни, ствола и снаряда в скрипте **Bot** танка-бота с соответсвующими объектами на сцене (выбирайте башню и ствол из списка правильно, поскольку названия этих объектов у танка-игрока и танка-бота совпадают и нужно подобрать правильное соответсвие) и нажмите кнопку **Apply;**
  5. удалите у танка-бота на сцене камеру (если она была привязана к исходному танку=игроку);
  6. удалите у танка-бота на сцене все старые скрипты для управления исходным танком (оставшиеся от танка-игрока) и его стволом (для запуска снаряда танком-игроком), оставив на танке-боте только новый скрипт **«Bot»**.

**ОПРЕДЕЛИТЬ ПЕРЕМЕННЫЕ**

**using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;**

**public class bot : MonoBehaviour {  
    float movespeed = 0.25f; //** скорость передвижения танка-бота **float rotspeedtank = 0.1f; //** скорость поворота танка-бота **float rotspeedbash = 0.5f; //** скорость поворота башни танка-бота **float speedcore = 3f; //** скорость снаряда танка-бота **public Transform bash; //** для управления башней **public Transform stvol; //** для управления стволом **public GameObject core; //** для ссылки на префаб снаряда **bool canshoot = true; //** для определения, может ли танк-бот произвести выстрел **int life = 3; //** для определения максимального количества попаданий в танк-бот

**...**

*!!! НЕОБХОДИМО ПРОИНИЦИАЛИЗИРОВАТЬ ОБЪЕКТНЫЕ ПЕРЕМЕННЫЕ* через **Inspector –** указать на соответствующие объекты на сцене.

**ДОБАВИТЬ В СКРИПТ БОТА МЕТОД OnTriggerStay(Collider other)**

В методе определяется, какие действия должен выполнять танк-бот при нахождении в триггере танка-игрока (*танку-игроку присвоен тэг* ***Player)*.**

**private void OnTriggerStay(Collider other)  
    {  
   if (other.tag == "Player")** // если бот попал в триггер танка-игрока выполняем действия **А, Б, В** и **Г**  
**{**

А. ОПРЕДЕЛЯЕМКВАТЕРНИОН ДЛЯ ПОВОРОТА БОТА В СТОРОНУ ИГРОКА

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

// определяем вектор направления между игроком и ботом

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

// по найденному вектору определяем кватернион для поворота

*Метод* ***LookRotation()*** *вычисляет кватернион, проворачивающий объект в направлении, указанном вектором в аргументе метода.*

Б. ПЛАВНО ПОВОРАЧИВАЕМ БАШНЮ ТАНКА-БОТА В СТОРОНУ ТАНКА-ИГРОКА

**bash.rotation = Quaternion.Slerp(bash.rotation, newrot, Time.deltaTime \* rotspeedbash);**

*Метод* ***Slerp()*** *задает плавный поворот объекта от начального положения вектора поворота в его конечное положение (сферическая линейная интерполяция) с заданной скоростью.*

В. МЕТОД «БРОСАНИЯ ЛУЧЕЙ» для определения момента, повернута ли башня на игрока

**RaycastHit hit;**

//переменная для определения объекта попадания «луча»

**If(Physics.Raycast(bash.position,bash.TransformDirection(Vector3.forward), out hit))**

//если выпущен луч из башни в направлении относительно нее – вперед

**{**

**If((hit.transform.tag==”Player”) && canshoot)**

//если луч попал в коллайдер игрока и можно выстрелить

**StartCoroutine(botshoot());**

//запускаем короутину для выстрела танка-бота

**}**

*Метод* ***Physics.Raycast*** *создает «луч» в заданном направлении.*

Г. ПЛАВНЫЕ ДВИЖЕНИЕ И ПОВОРОТ ТАНКА-БОТА В НАПРАВЛЕНИИ К ТАНКУ-ИГРОКУ

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

// вычисляем дистанцию бота до игрока

**If(distance<20)**

**{**

**transform.rotation = Quaternion.Slerp(transform.rotation, newrot, Time.deltaTime \* rotspeedtank);**

//плавно поворачиваем танк-бот в сторону танка-игрока с заданной скоростью

**transform.position = new Vector3.Lerp(transform.position,other.transform.position,Time.deltaTime\*movespeed),**

// плавно двигаем танк-бота в сторону игрока, меняя значения по осям X и Z

**}**

*Метод* ***Lerp()*** *задает плавный переход от одной позиции в 3D-пространстве к другой с заданной скоростью (линейная интерполяция – linear interpolation).*

1. КОРОУТИНА ДЛЯ ЗАПУСКА ВЫСТРЕЛА ТАНКА-БОТА:

**IEnumerator botshoot()**

**{**

**canshoot = false;** //указываем, что танк-бот стрелять пока не может

//определяем координату для положения снаряда танка-бота

**Vector3 forwardofstvol = stvol.transform.position + stvol.TransformDirection(Vector3.forward\*4f);**

//создаем снаряд из префаба снаряда в требуемой координате относительно ствола

**GameObject newcore = Instantiane(core, forwardofstvol, stvol.rotation);**

**yield return new WaitForSeconds(3f);** //ждем 3 секунды (время «перезарядки»)

**canshoot = true;** //указываем, что танк может сделать выстрел

**}**

*Сопрограммы* ***– короутины (coroutines)*** *в Unity выполняются параллельно программе в течение некоторого времени. Этим они отличаются от большинства функций, заставляющих программу ждать окончания своей работы. Короутины обычно используются для задержки выполнения какого-либо действия во время выполнения основной программы.*

*Короутины для своего выполнения используют встроенную функцию* ***IEnumerator.*** *Главным компонентом в теле короутины является ключевое слово* ***yield****, временно прерывающее ее работу, возвращающее управление основной программе и в следующем кадре возобновляющее сопрограмму с прерванной точки с заданным интервалом ожидания в секундах функцией* ***WaitForSeconds().***

1. КОД ДЛЯ УНИЧТОЖЕНИЯ ТАНКА-БОТА

Добавить в скрипт префаба метод **OnCollisionEnter (Collision collision)**. Внутри этого метода, определить по тэгу коллизию со снарядом:

**private void OnCollisionEnter(Collision collision)  
    {  
        if (collision.gameObject.tag == "core")  
        {  
            life--;  
            if (life < 1) Destroy(gameObject);  
        }  
     }**