TecnoCampus
Mataró-Maresme

DESENVOLUPAMENT DE JOCS 3D

Plan docente

https://pladocent.tecnocampus.cat/pdfs/2019/24/106312_es.pdf

Prácticas

Práctica 1 - https://youtu.be/n6faxMVvVgw

Práctica 2 - https://youtu.be/MRsd0ptJxV8

Práctica 3 - https://youtu.be/D0PSJy2thnM

Character controller

¿qué vemos?

https://www.youtube.com/watch?v=cAbIzG-aSDY

Character controller

¿qué jugamos?

https://www.youtube.com/watch?v=LJp2NnKj1X4

Character controller

Documentación

https://docs.unity3d.com//Manual/class-CharacterController.html

FPS Controller

Ejemplos

Doom

https://www.youtube.com/watch?v=K0nlO87evhY

Forsaken

https://www.youtube.com/watch?v=cs8UXvNX8W8

FPS Controller

Creación de un controlador FPS

-Necesitamos un objeto que controlará el personaje con un script de controlador FPS.

-Introducimos el modelo animado que se pintará en caso de que nos veamos reflejados.

-Introducimos un game object que controlará el aim (pitch) del FPS.

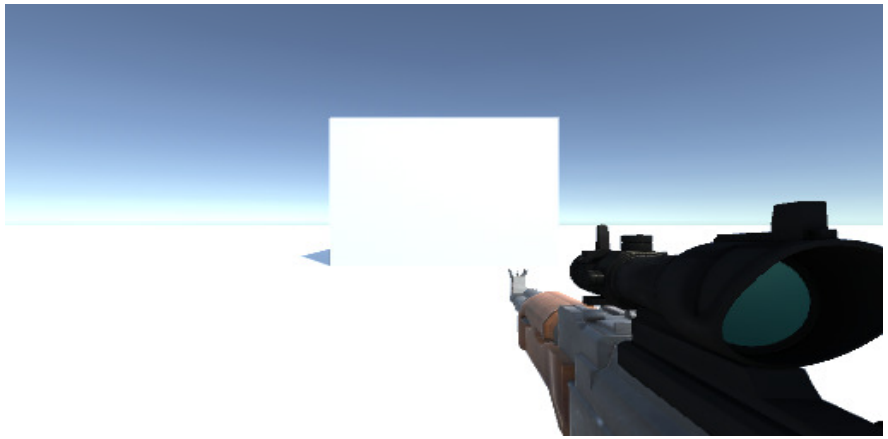
-Introducimos en este nuevo game object la cámara que representa los ojos del player y el arma en la posición que deseemos.

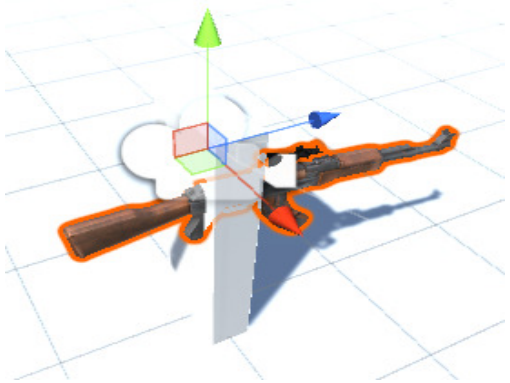
FPS Controller

Creación de un controlador FPS

FPS Controller

Creación de un controlador FPS

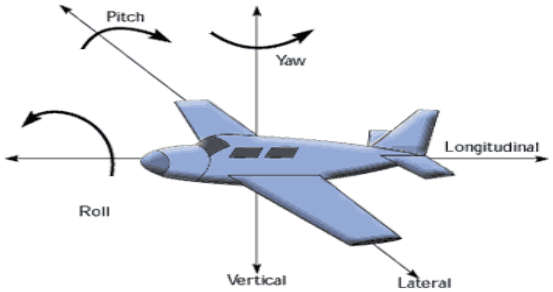




FPS Controller

Controlando la rotación

-En un controlador FPS debemos tener en cuenta dos tipos de rotación. El YAW y el PITCH, en caso de realizar un controlador FPS totalmente libre añadiríamos el ángulo ROLL.



FPS Controller

Controlando la rotación

En un controlador FPS debemos tener en cuenta dos tipos de rotación. El YAW y el PITCH, en caso de realizar un controlador FPS totalmente libre añadiríamos el ángulo ROLL.

El ángulo YAW será controlado en coordenadas globales, mientras que el PITCH lo gestionaremos en coordenadas locales respecto a su padre.

FPS Controller

Controlando la rotación - Implementación

```
float m_Yaw;
float m_Pitch;
public float m_YawRotationalSpeed=360.0f;
public float m_PitchRotationalSpeed=180.0f;
public float m_MinPitch=-80.0f;
public float m_MaxPitch=50.0f;
public Transform m_PitchControllerTransform;
public bool m_InvertedYaw=false;
public bool m_InvertedPitch=true;
```

FPS Controller

Controlando la rotación - Implementación

```
void Awake()
{
      m_Yaw=transform.rotation.eulerAngles.y;
      m_Pitch=m_PitchControllerTransform.localRotation.eulerAngles.x;
}
void Update()
{
      //…
      float l_MouseAxisY=Input.GetAxis("Mouse Y");
      m_Pitch+=l_MouseAxisY*m_PitchRotationalSpeed*Time.deltaTime;
      m_Pitch=Mathf.Clamp(m_Pitch, m_MinPitch, m_MaxPitch);
      //…
      transform.rotation=Quaternion.Euler(0.0f, m_Yaw, 0.0f);
      m_PitchControllerTransform.localRotation=Quaternion.Euler(m_Pitch, 0.0f, 0.0f);
      //…
}
```

FPS Controller

Controlando el movimiento

Para controlar el movimiento de nuestro controlador FPS utilizaremos el componente CharacterController.

Añadiremos el componente en el gameobject Player y modificaremos sus propiedades según las necesidades de nuestro personaje y juego.

FPS Controller

Controlando el movimiento

Para mover el personaje utilizaremos las teclas WASD, dónde las WS moverán el personaje según el vector forward y las teclas AD moverán al personaje según el vector right.

FPS Controller

Controlando el movimiento – Implementación

```
CharacterController m_CharacterController;
public float m_Speed=10.0f;
public KeyCode m_LeftKeyCode=KeyCode.A;
public KeyCode m_RightKeyCode=KeyCode.D;
public KeyCode m_UpKeyCode=KeyCode.W;
public KeyCode m_DownKeyCode=KeyCode.S;
```

FPS Controller

Controlando el movimiento – Implementación

```
void Awake()
{
        m_CharacterController=GetComponent<CharacterController>();
}
void Update()
{
        //…
        float l_YawInRadians=m_Yaw*Mathf.Deg2Rad;
        float l_Yaw90InRadians=(m_Yaw+90.0f)*Mathf.Deg2Rad;
        Vector3 l_Forward=new Vector3(Mathf.Sin(l_YawInRadians), 0.0f, Mathf.Cos(l_YawInRadians));
        Vector3 l_Right=new Vector3(Mathf.Sin(l_Yaw90InRadians), 0.0f, Mathf.Cos(l_Yaw90InRadians));
        if(Input.GetKey(m_UpKeyCode))
                l_Movement=l_Forward;
        else if(Input.GetKey(m_DownKeyCode))
                l_Movement=-l_Forward;

        if(Input.GetKey(m_RightKeyCode))
                l_Movement+=l_Right;
        else if(Input.GetKey(m_LeftKeyCode))
        l_Movement-=l_Right;

        l_Movement.Normalize();
        l_Movement=l_Movement*Time.deltaTime*m_Speed;
        //…
        CollisionFlags l_CollisionFlags=m_CharacterController.Move(l_Movement);
}
```

FPS Controller

Añadiendo gravedad

Para implementar gravedad a nuestro personaje utilizaremos una variable que irá acumulando la velocidad vertical del personaje según la gravedad.

Cuando el personaje colisione contra el suelo o contra un techo y su velocidad vertical sea positivia asignaremos un valor de 0 a la velocidad vertical.

Para utilizar el motor de física de Unity utilizaremos el método Move del character controller, modificando el vector movimiento que hemos usado previamente.

FPS Controller

Añadiendo gravedad - Implementación

```
float m_VerticalSpeed=0.0f;
bool m_OnGround=false;

void Update()
{
	//…
	m_VerticalSpeed+=Physics.gravity.y*Time.deltaTime;
	l_Movement.y=m_VerticalSpeed*Time.deltaTime;

	CollisionFlags l_CollisionFlags=m_CharacterController.Move(l_Movement);
	if((l_CollisionFlags & CollisionFlags.Below)!=0)
	{
		m_OnGround=true;
		m_VerticalSpeed=0.0f;
	}
	else
		m_OnGround=false;

	if((l_CollisionFlags & CollisionFlags.Above)!=0 &&  m_VerticalSpeed>0.0f)
		m_VerticalSpeed=0.0f;
}
```

FPS Controller

Salto y correr

Para añadir la mecánica de salto modificaremos el código de tal manera que al pulsar la tecla de espacio asignaremos una velocidad vertical inicial de salto.

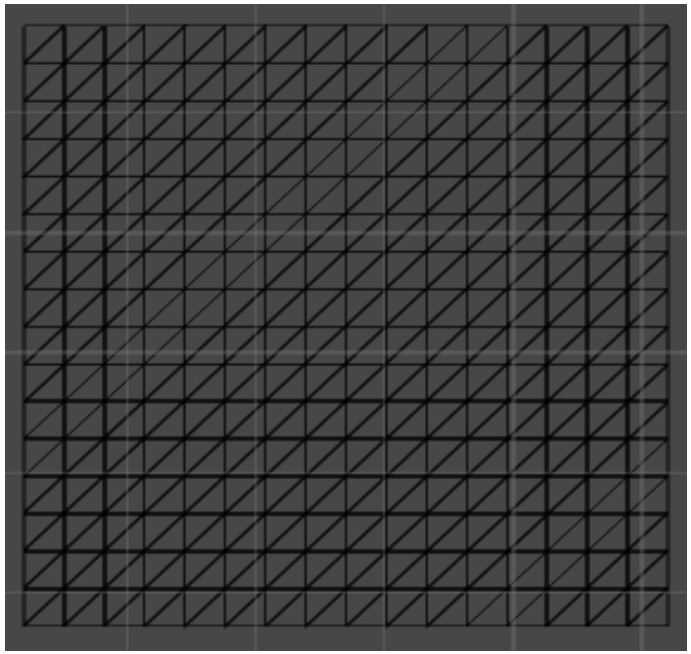
Para avanzar más rápido al pulsar la tecla de shift utilizaremos una variable multiplicadora sobre la velocidad de desplazamiento.

FPS Controller

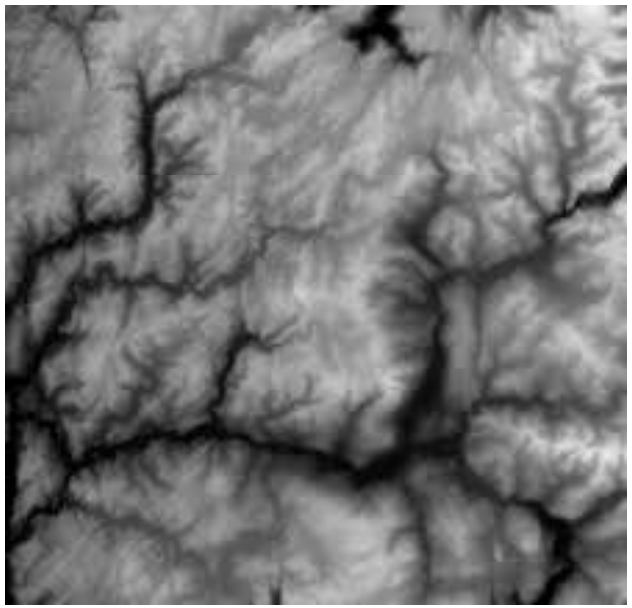
Salto y correr - Implementación

```
float m_VerticalSpeed=0.0f;
public KeyCode m_RunKeyCode=KeyCode.LeftShift;
public KeyCode m_JumpKeyCode=KeyCode.Space;
public float m_FastSpeedMultiplier=1.2f;
public float m_JumpSpeed=10.0f;

void Update()
{
	//…
	float l_SpeedMultiplier=1.0f;
	if(Input.GetKey(m_RunKeyCode))
		l_SpeedMultiplier=m_FastSpeedMultiplier;
	//…
	l_Movement*=Time.deltaTime*m_Speed*l_SpeedMultiplier;

	//…
	if(m_OnGround && Input.GetKeyDown(m_JumpKeyCode))
		m_VerticalSpeed=m_JumpSpeed;
}
```

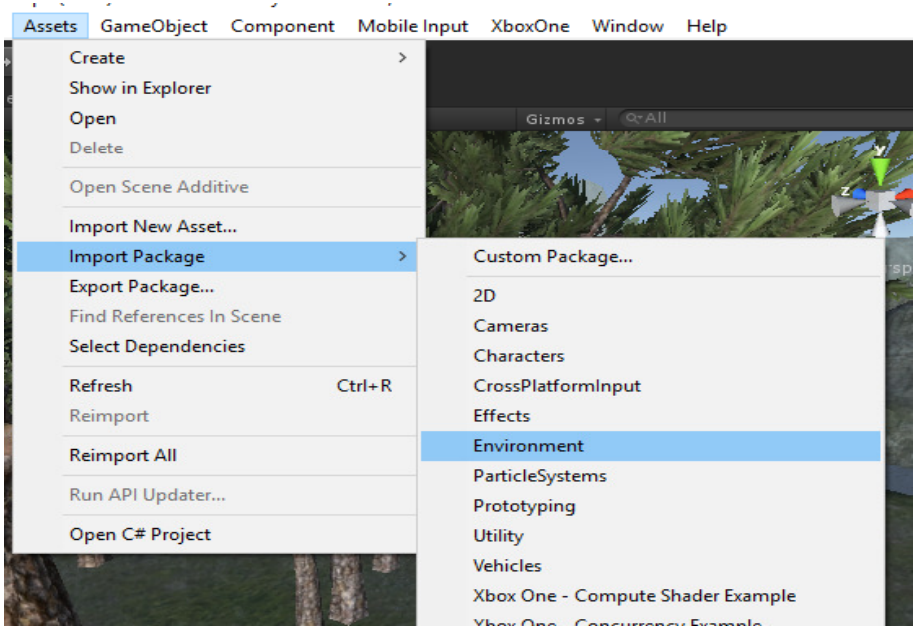
Terrain



Terrain (heightmap)



Terrain (Importing Environment)



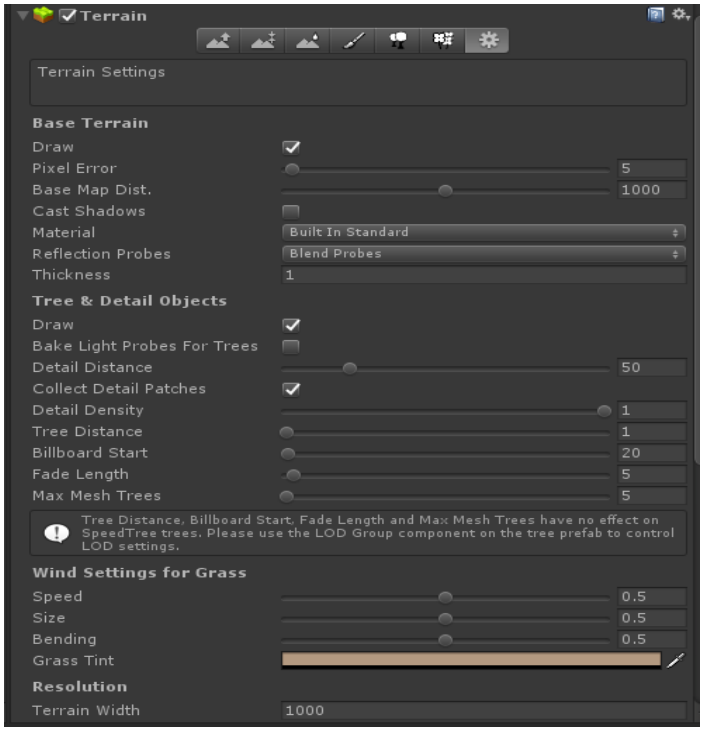
Terrain (Propiedades)

Terrain width

Terrain length

Terrain height

Detail resolution per patch

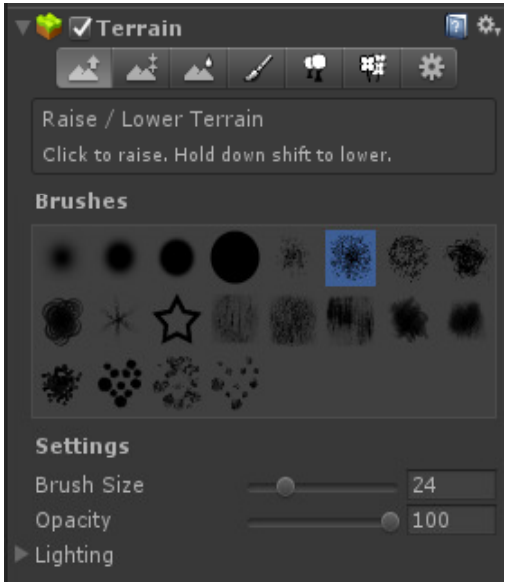


Terrain (Raise/Lower)

Brushes

Brush Size

Opacity



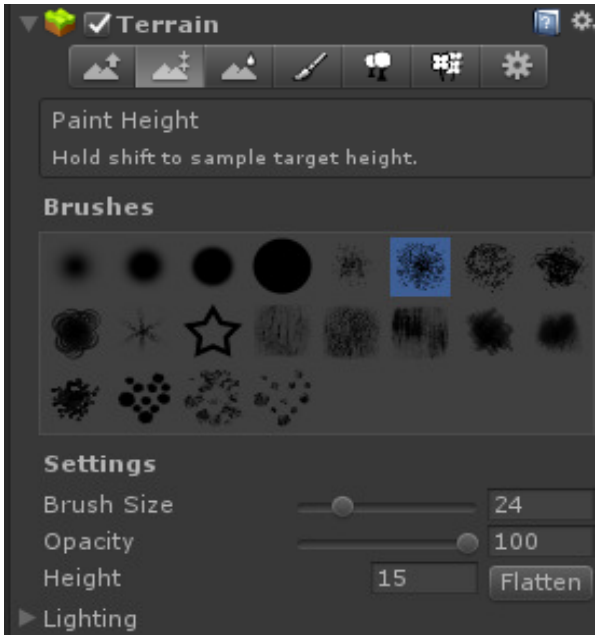
Terrain (Paint Height)

Brushes

Brush Size

Opacity

Height

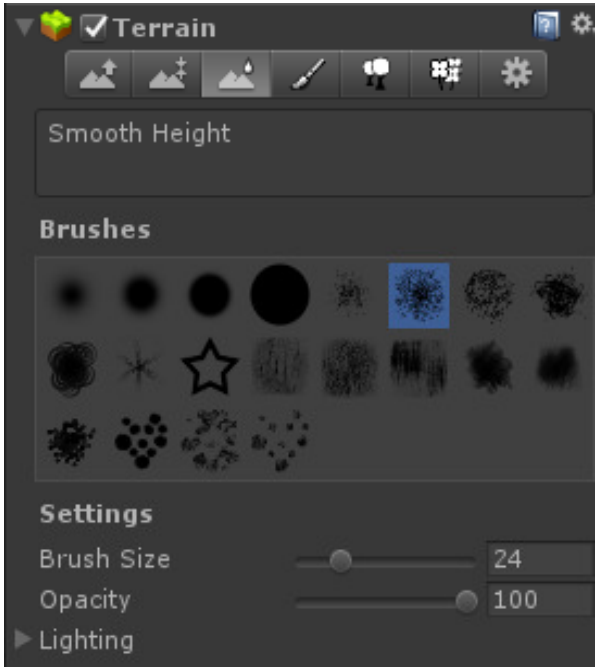


Terrain (Smooth Height)

Brushes

Brush Size

Opacity



Terrain (Paint Texture)

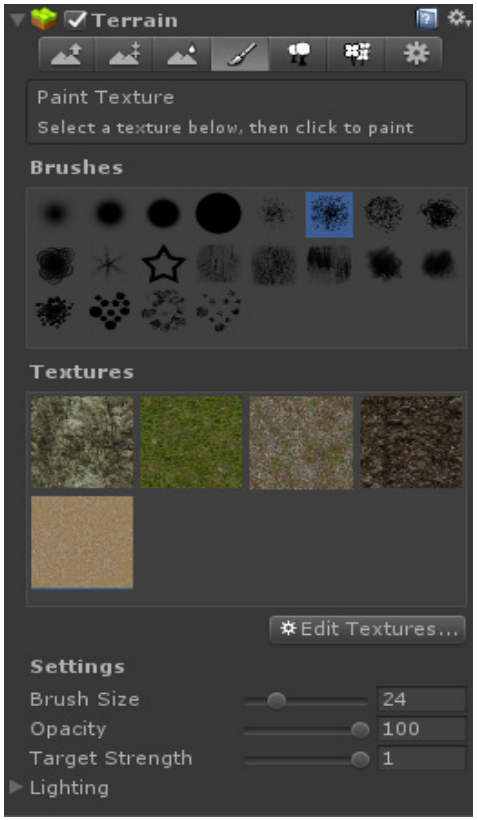
Brushes

Textures

Brush size

Opacity

Target Strength



Terrain (Place Trees)

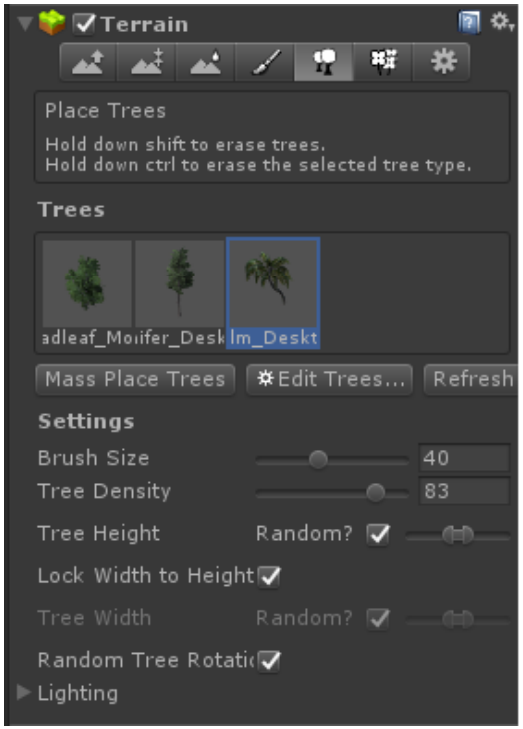
Trees

Brush size

Tree Density

Tree height

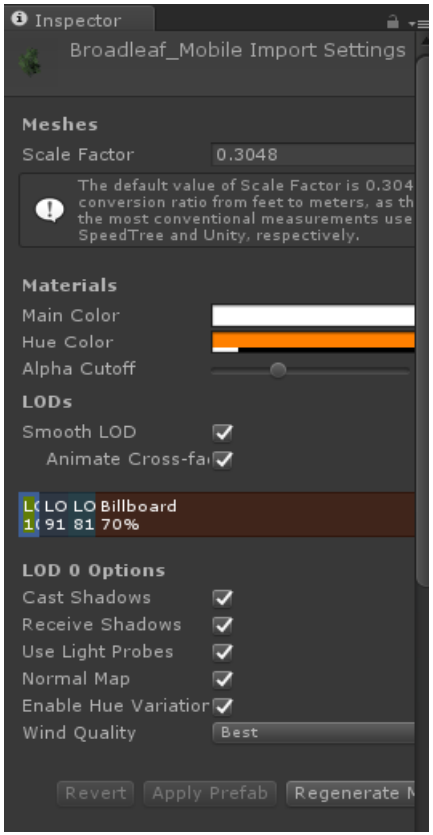
Random tree rotation



Speedtree

Materials

Lods



Terrain (Detail)

Brushes

Details

Brush size

Opacity

Target Strength

FPS Controller

Disparo - Implementación

```
public int m_MouseShootButton=0;
int m_CurrentAmmoCount=0;

if(Input.GetMouseButtonDown(m_MouseShootButton) && CanShoot())
      Shoot();

void Shoot()
{
      Ray l_CameraRay=Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.0f));
      RaycastHit l_RaycastHit;
      if(Physics.Raycast(l_CameraRay, out l_RaycastHit, 200.0f, m_ShootLayerMask.value))
                CreateShootHitParticles(l_RaycastHit.point, l_RaycastHit.normal);
}
void CreateShootHitParticles(Vector3 Position, Vector3 Normal)
{
      GameObject.Instantiate(m_ShootHitParticles, Position, Quaternion.identity, m_DestroyObjects);
}
```

Character controller

Disparo – Posibles mejoras

Recoil

Dispersion

Reload

https://www.youtube.com/watch?v=ju88Yv3OlO4

FPS Controller

Destroy object on time - Implementación

```
public float m_DestroyOnTime=3.0f;

StartCoroutine(DestroyOnTimeFn());

IEnumerator DestroyOnTimeFn()
{
     yield return new WaitForSeconds(m_DestroyOnTime);
     GameObject.Destroy(gameObject);
}
```

FPS Controller

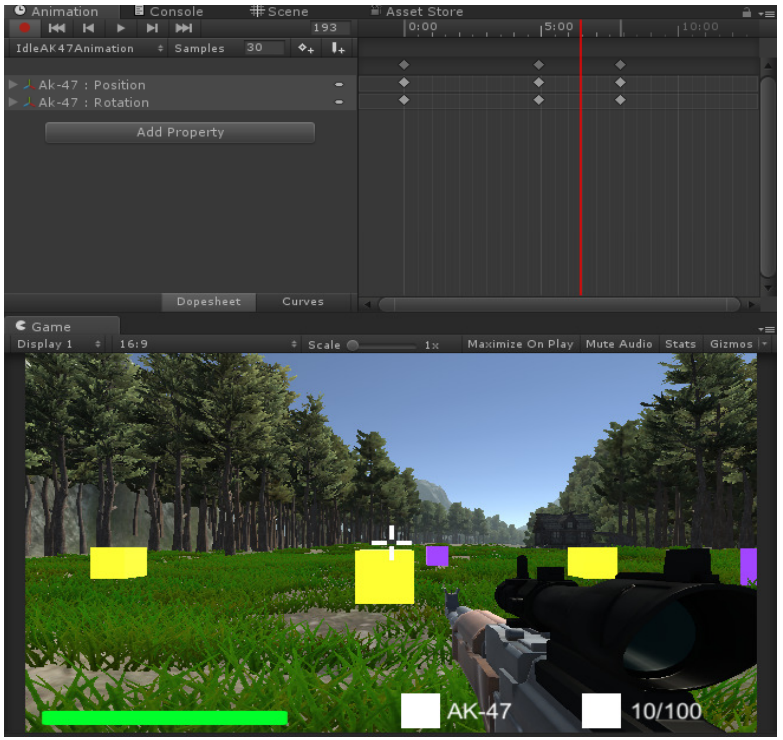
Lock Cursor/Angle - Implementación

```
public KeyCode m_DebugLockAngleKeyCode=KeyCode.I;
public KeyCode m_DebugLockKeyCode=KeyCode.O;

if(Input.GetKeyDown(m_DebugLockAngleKeyCode))
      m_AngleLocked=!m_AngleLocked;
if(Input.GetKeyDown(m_DebugLockKeyCode))
{
      if(Cursor.lockState==CursorLockMode.Locked)
                Cursor.lockState=CursorLockMode.None;
      else
                Cursor.lockState=CursorLockMode.Locked;
      m_AimLocked=Cursor.lockState==CursorLockMode.Locked;
}
```

FPS Controller

Animaciones

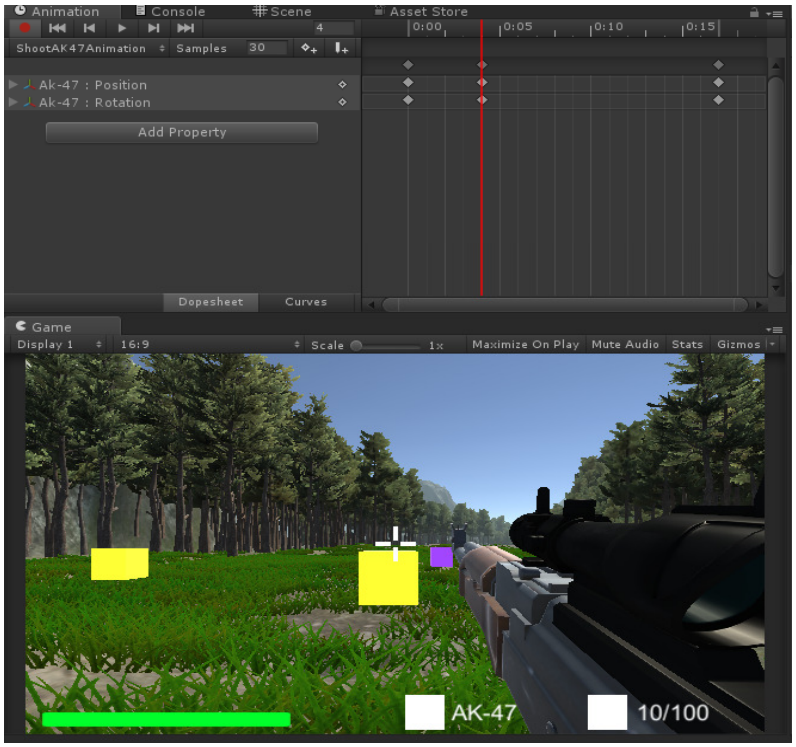
Idle (loop)



FPS Controller

Animaciones

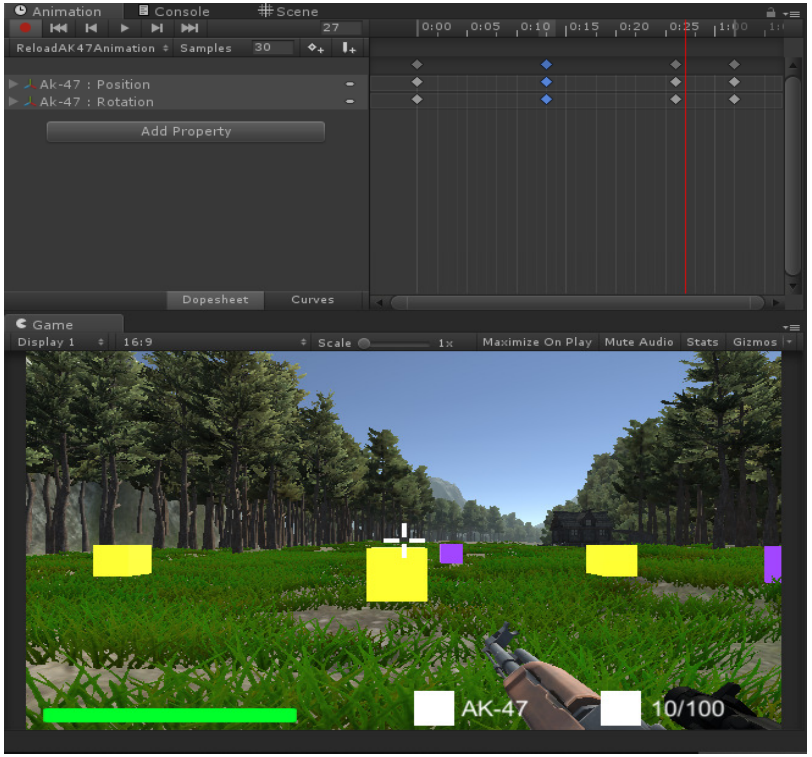
Shoot (once)



FPS Controller

Animaciones

Reload (once)



FPS Controller

Animaciones - Implementación

```
public Animation m_WeaponAnimation;
public AnimationClip m_IdleWeaponAnimationClip;
public AnimationClip m_ShootWeaponAnimationClip;
public AnimationClip m_ReloadWeaponAnimationClip;

void SetIdleWeaponAnimation()
{
      m_WeaponAnimation.CrossFade(m_IdleWeaponAnimationClip.name);
}
void SetShootWeaponAnimation()
{
      m_WeaponAnimation.CrossFade(m_ShootWeaponAnimationClip.name);
      m_WeaponAnimation.CrossFadeQueued(m_IdleWeaponAnimationClip.name);
}
```

FPS Controller

GameController - Implementación

```
[Header("Game")]
public FPSPlayerController m_PlayerController;
public Transform m_DestroyObjects;
```

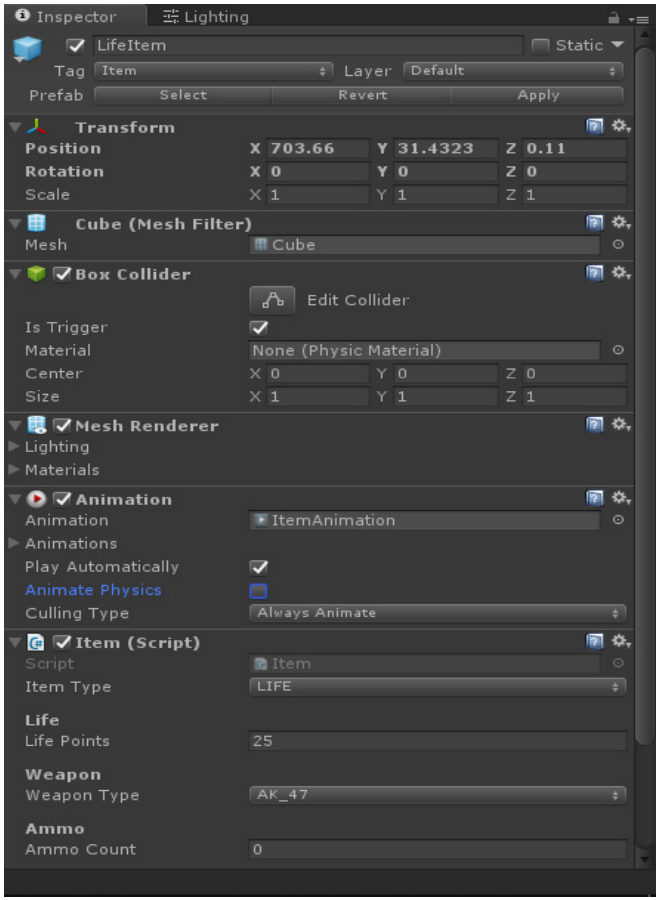
Items

Tag

Item

Animation

Box collider (Is trigger)



Items - Implementación

```
public enum TItemType
{
     LIFE,
     WEAPON,
     AMMO
}
GameController m_GameController;

public TItemType m_ItemType;

[Header("Life")]
public float m_LifePoints;
[Header("Ammo")]
public int m_AmmoCount;
```

Items - Implementación

```
void TakeLifeItem()
{
     if(m_GameController.m_PlayerController.GetLife()<m_GameController.m_PlayerController.m_MaxLife)
     {
              m_GameController.m_PlayerController.AddLife(m_LifePoints);
              DestroyItem();
     }
}
void DestroyItem()
{
     GameController.Destroy(gameObject);
}
```

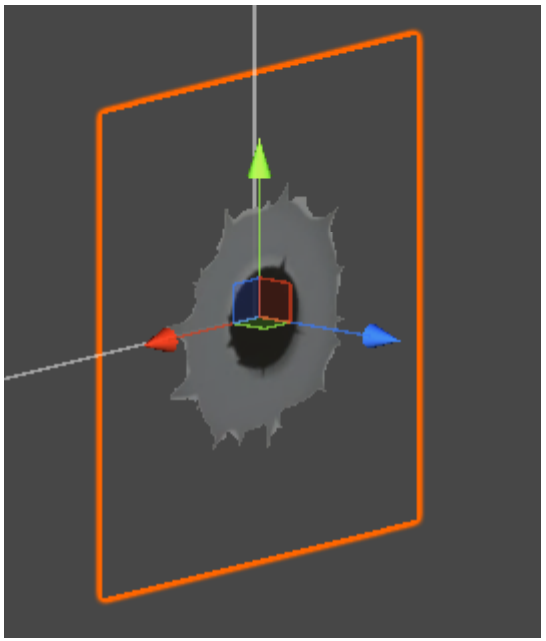
Items - Implementación

```
//En clase FPSPlayerController
void OnTriggerEnter(Collider _Collider)
{
     if(_Collider.tag=="Item")
     {
              Item l_Item=_Collider.GetComponent<Item>();
              l_Item.TakeItem();
     }
}
```

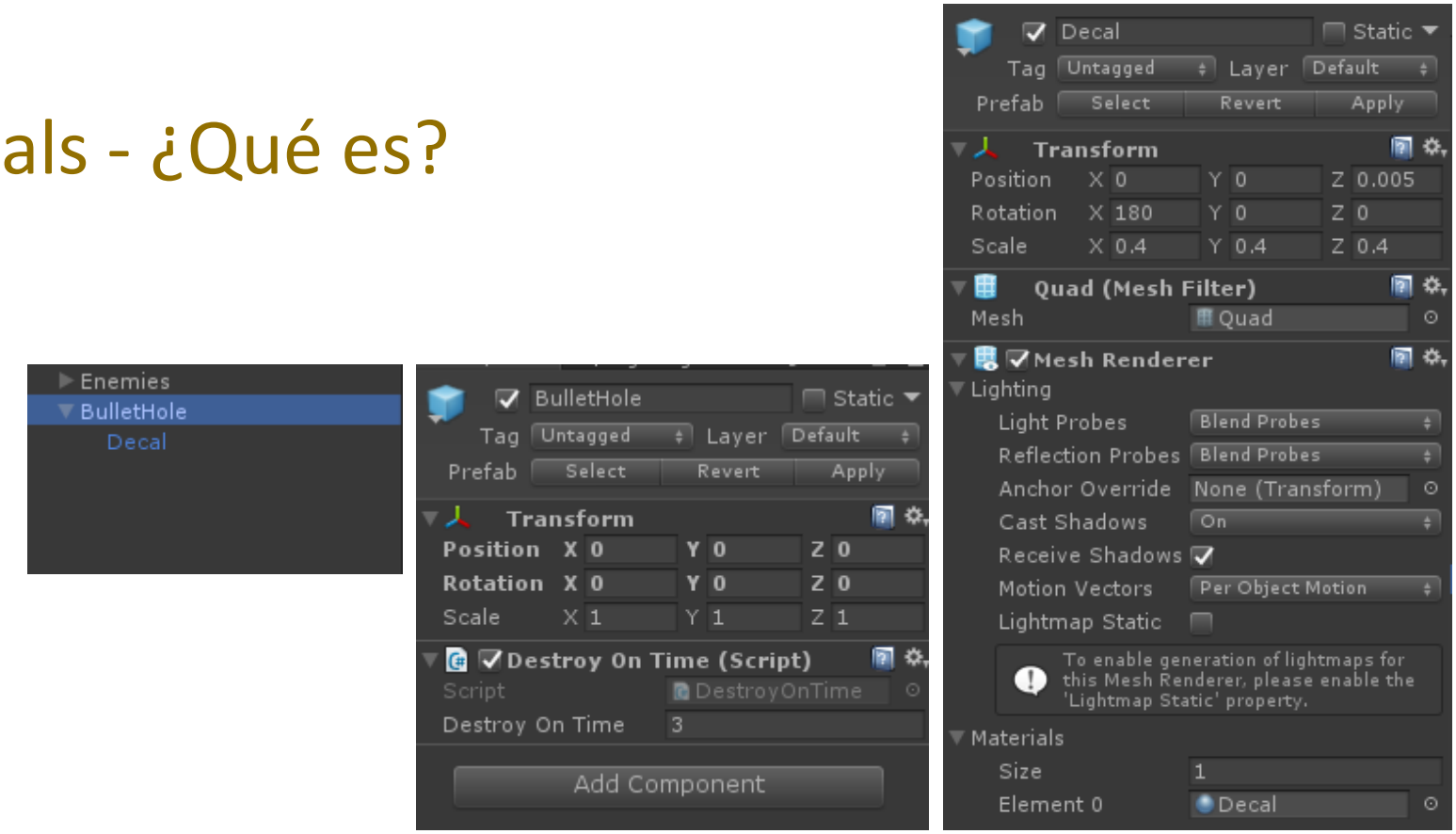
Decals - ¿Qué es?

Un decal es una pegatina que se pone sobre el escenario.

Para ello creamos un prefab con un Quad orientado al eje Z, al ponerlo sobre el escenario dónde colisione nuestra bala mirando en la normal del plano conseguiremos el efecto buscado.



Decals - ¿Qué es?



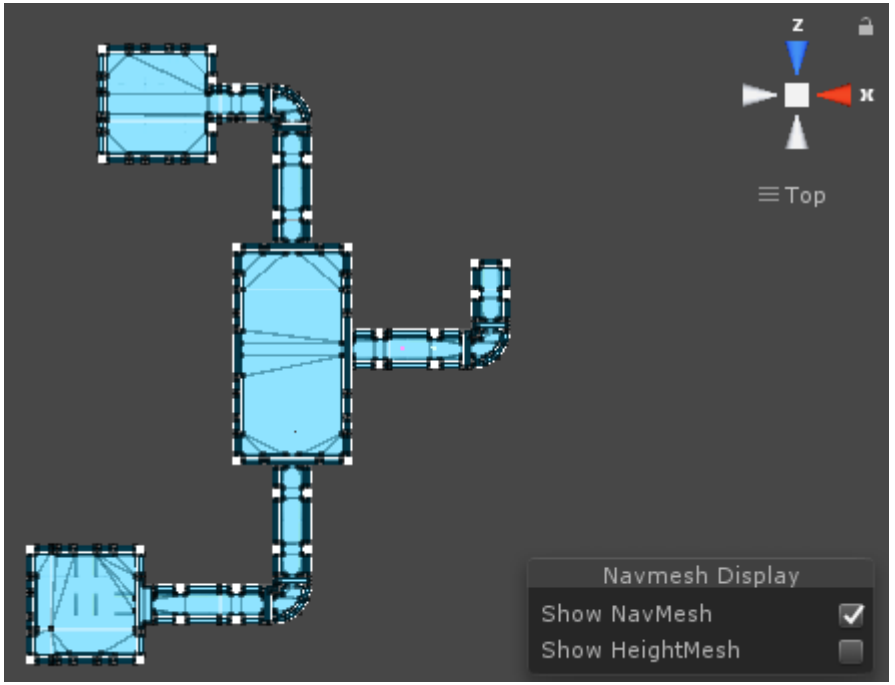
Decals - Implementación

```
void CreateShootHitParticles(Vector3 Position, Vector3 Normal)
{
     //…
     GameObject.Instantiate(m_BulletHoleDecal, Position, Quaternion.LookRotation(Normal),
     m_GameController.m_DestroyObjects);
}
```

Navigation

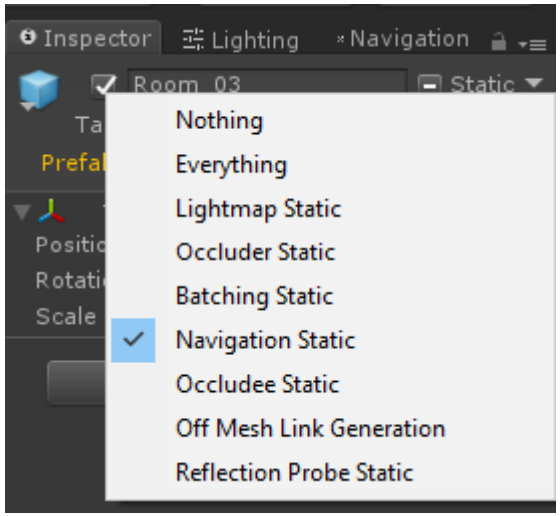
Para la implementación de los enemigos vamos a utilizar la funcionalidad de Navigation de Unity.

Esta funcionalidad nos va a permitir navegar personajes por el escenario de forma sencilla utilizando la clase NavMeshAgent.



Navigation

Para delimitar la zona de navegación debemos activar la propiedad navigation static.

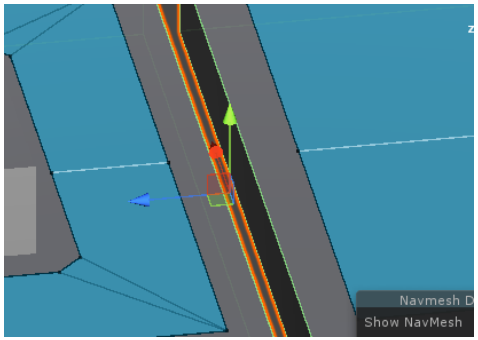


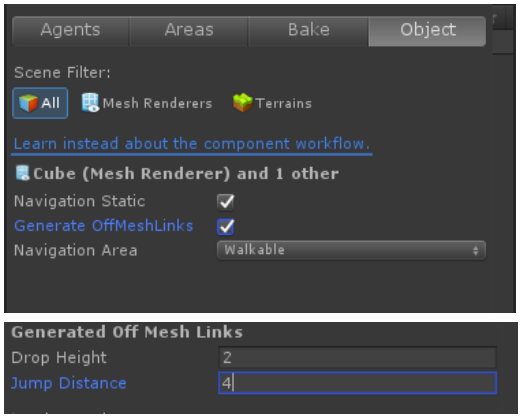
Navigation - OffMeshLinks

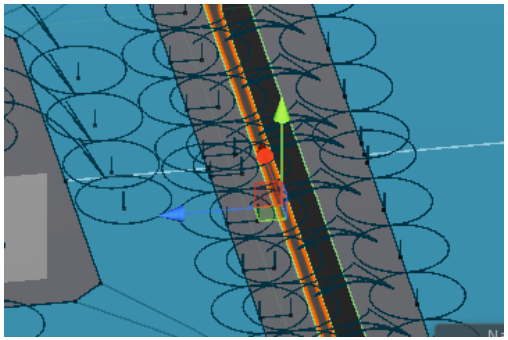
Podemos crear uniones entre diferentes zonas transitables.

Para ello seleccionaremos las zonas transitables y marcamos el checkbox de Generate OffMeshLinks en Object.

Por último para realizar el bake debemos ponerle la distancia de salto que somos capaces de superar y la distancia de caída para bajar una altura.

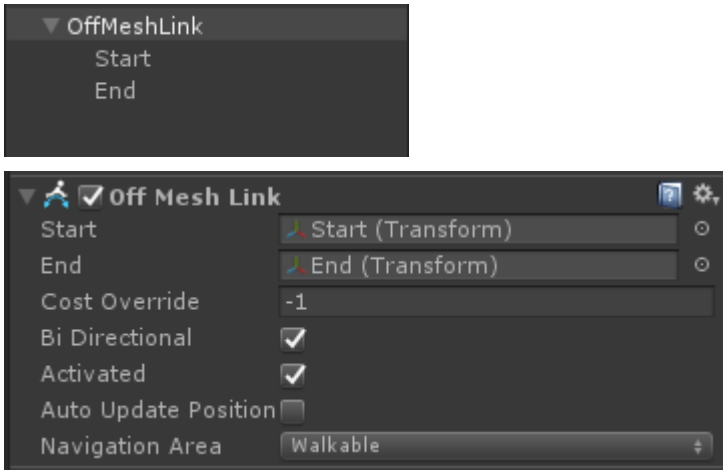


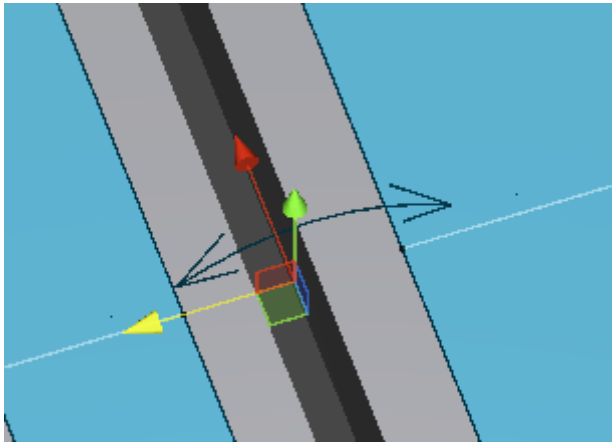




Navigation - OffMeshLinks

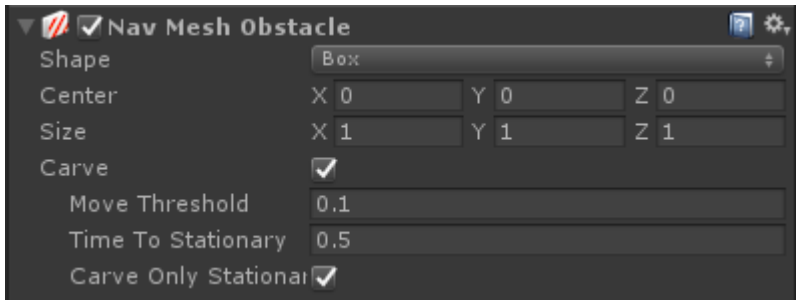
También podemos crear uniones de forma manual utilizando el componente OffMeshLink.

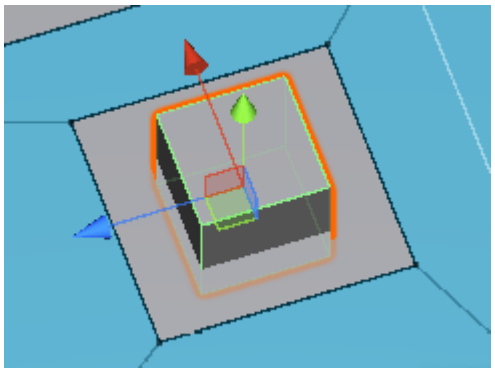




Navigation - NavMeshObstacle

Otra feature que nos ofrece Unity para la navegación es la de poner obstáculos dinámicos que cambian la malla de navegación. Para ello utilizamos los componentes NavMeshObstacle.

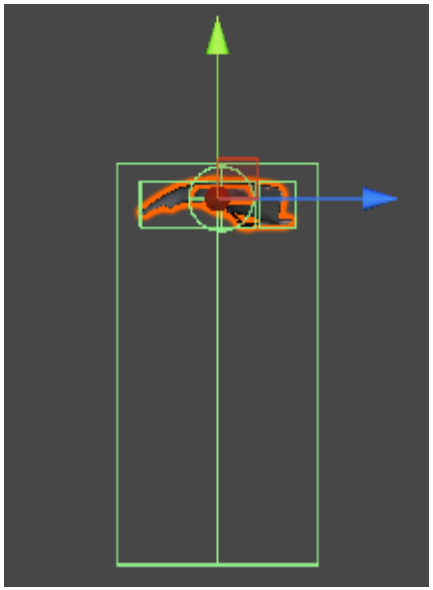


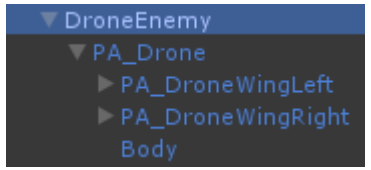


Enemigos

Utilizaremos un drone descargado de la Unity asset store.

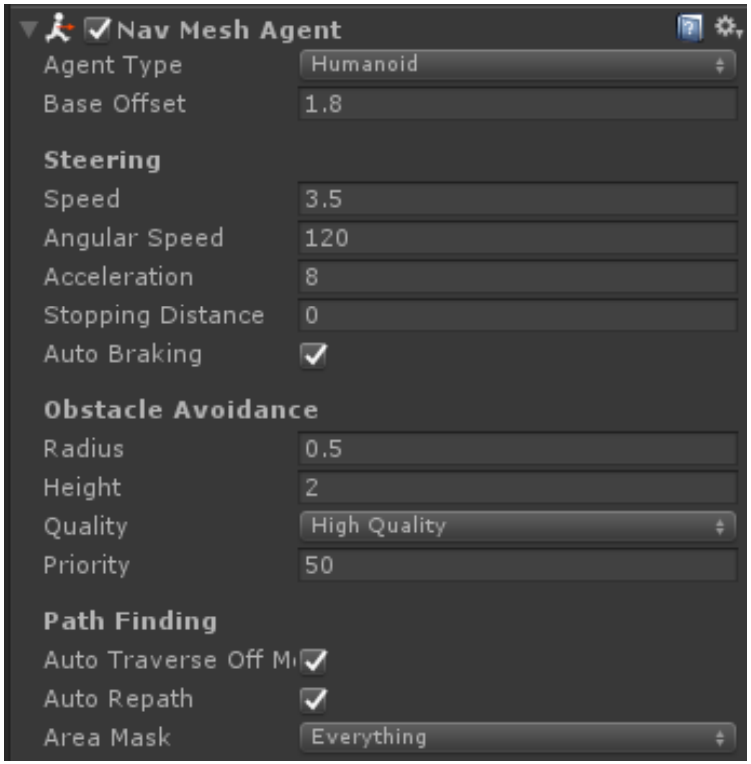
https://www.assetstore.unity3d.com/en/?stay#!/content/15159





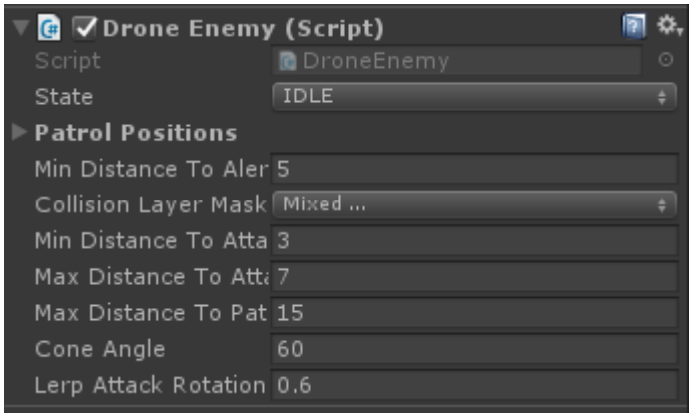
Enemigos

Para poder navegar con un enemigo por la zona navegable utilizaremos el componente NavMeshAgent.



Enemigos

Para crear el enemigo crearemos un script que implementará una máquina finita de estados (FSM).



Enemigos – Implementación

```
NavMeshAgent m_NavMeshAgent;
public enum TState
{
      IDLE=0,
      PATROL,
      ALERT,
      CHASE,
      ATTACK,
      HIT,
      DIE
}
public TState m_State;
public List<Transform> m_PatrolPositions;
float m_CurrentTime=0.0f;
int m_CurrentPatrolPositionId=-1;
float m_StartAlertRotation=0.0f;
float m_CurrentAlertRotation=0.0f;
GameController m_GameController;
public float m_MinDistanceToAlert=5.0f;
public LayerMask m_CollisionLayerMask;
public float m_MinDistanceToAttack=3.0f;
public float m_MaxDistanceToAttack=7.0f;
public float m_MaxDistanceToPatrol=15.0f;
public float m_ConeAngle=60.0f;
public float m_LerpAttackRotation=0.6f;
const float m_MaxLife=100.0f;
float m_Life=m_MaxLife;
[Range(0.0f, 1.0f)]
public float m_ShootAccuracy=0.3f;
```

Enemigos – Implementación

```
void Update()
{
    m_CurrentTime+=Time.deltaTime;
    switch(m_State)
    {
          case TState.IDLE:
                        UpdateIldeState();
                        break;
          //...
    }
}
void SetPatrolState()
{
    m_State=TState.PATROL;
    m_CurrentTime=0.0f;
    m_CurrentPatrolPositionId=GetClosestPatrolPositionId();
    m_NavMeshAgent.isStopped=false;
    m_NavMeshAgent.SetDestination(m_PatrolPositions[m_CurrentPatrolPositionId].position);
}
```

Enemigos – Implementación

```
void SetNextChasePosition()
{
    m_NavMeshAgent.isStopped=false;
    Vector3 l_Destination=m_GameController.m_PlayerController.transform.position-transform.position;
    float l_Distance=l_Destination.magnitude;
    l_Destination/=l_Distance;
    l_Destination=transform.position+l_Destination*(l_Distance-m_MinDistanceToAttack);
    m_NavMeshAgent.SetDestination(l_Destination);
}
void MoveToNextPatrolPosition()
{
    ++m_CurrentPatrolPositionId;
    if(m_CurrentPatrolPositionId>=m_PatrolPositions.Count)
         m_CurrentPatrolPositionId=0;
    m_NavMeshAgent.SetDestination(m_PatrolPositions[m_CurrentPatrolPositionId].position);
}
```

Enemigos – Implementación

```
bool SeesPlayer()
{
   Vector3
   l_Direction=(m_GameController.m_PlayerController.transform.position+Vector3.up*0.9f
   )-transform.position;
   Ray l_Ray=new Ray(transform.position, l_Direction);
   float l_Distance=l_Direction.magnitude;
   l_Direction/=l_Distance;
   bool l_Collides=Physics.Raycast(l_Ray, l_Distance, m_CollisionLayerMask.value);
   float l_DotAngle=Vector3.Dot(l_Direction, transform.forward);

   Debug.DrawRay(transform.position, l_Direction*l_Distance, l_Collides ? Color.red :
   Color.yellow);
   return !l_Collides && l_DotAngle>Mathf.Cos(m_ConeAngle*0.5f*Mathf.Deg2Rad);
}
```

Enemigos – Implementación

```
bool HearsPlayer()
{
   return
   GetSqrDistanceXZToPosition(m_GameController.m_PlayerController.transform.position)<
   (m_MinDistanceToAlert*m_MinDistanceToAlert);
}
void UpdatePatrolState()
{
   if(!m_NavMeshAgent.hasPath &&
   m_NavMeshAgent.pathStatus==NavMeshPathStatus.PathComplete)
       MoveToNextPatrolPosition();

   if(HearsPlayer())
       SetAlertState();
}
```

HitCollider – Implementación

```
public class HitCollider : MonoBehaviour
{
   public enum THitColliderType
   {
       HEAD,
       HELIX,
       BODY
   }
   public THitColliderType m_HitColliderType;
   public DroneEnemy m_DroneEnemy;
}
```

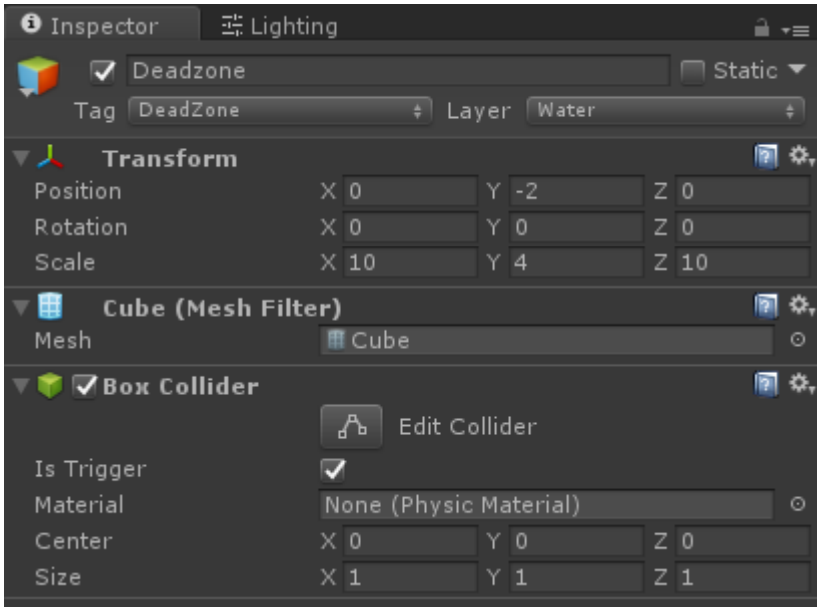
FPSPlayerController – Implementación

```
public void Shoot()
{
    Ray l_CameraRay=Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.0f));
    RaycastHit l_RaycastHit;
    if(Physics.Raycast(l_CameraRay, out l_RaycastHit, 200.0f, m_GameController.m_PlayerController.m_ShootLayerMask.value))
    {
          switch(l_RaycastHit.collider.tag)
          {
                case "Enemy":
                      HitCollider l_HitCollider=l_RaycastHit.collider.GetComponent<HitCollider>();
                      l_HitCollider.m_DroneEnemy.Hit(l_HitCollider.m_HitColliderType==HitCollider.THitColliderType.HEAD ?
    100.0f : 25.0f);
                      break;
          }
          CreateShootHitParticles(l_RaycastHit.point, l_RaycastHit.normal);
    }
}
```

Deadzone

Para crear una zona dónde el jugador muera nada más tocarlo utilizaremos una box collider y la estableceremos como trigger.

Por último usaremos el tag DeadZone para diferenciar el trigger.

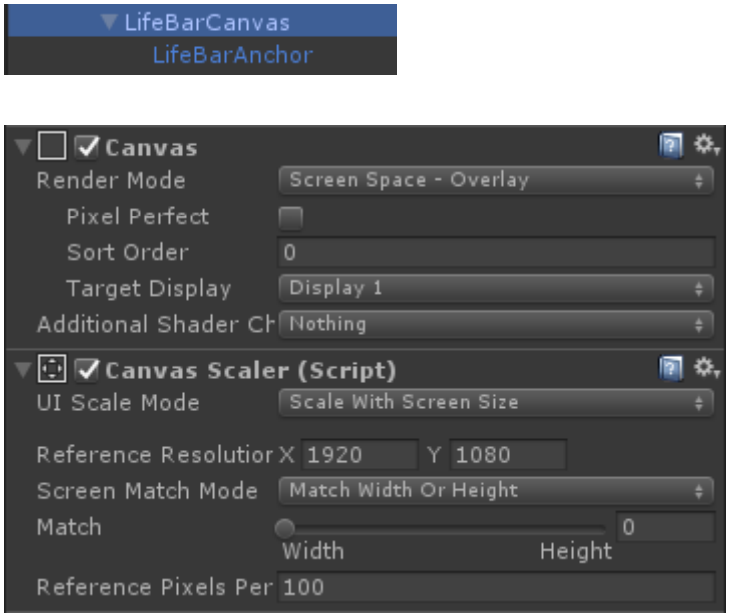


DeadZone – Implementación

```
void OnTriggerEnter(Collider _Collider)
{
   //..
   else if(_Collider.tag=="DeadZone")
       KillPlayer();
}
void KillPlayer()
{
   m_Life=0.0f;
   m_GameController.ShowLife(0.0f);
   m_GameController.RestartGame();
   //Podemo reiniciar el juego utilizando una corutina o una función lambda cuando
   terminemos de mostrar al jugador que ha muerto
}
```

Lifebar 2D

Para pintar la barra de vida en 2D, creamos un canvas con la resolución que nos interese y dentro añadimos la imagen (LifeBarAnchor) con el pivote en el top left y de Image type Filled y Fill method horizontal.



Lifebar 2D – Implementación

```
void UpdateLifeBarPosition()
{
   Vector3
   l_ViewportPoint=Camera.main.WorldToViewportPoint(transform.position+Vec
   tor3.up*m_LifeBarOffsetY);
   m_LifeBar.rectTransform.anchoredPosition=new
   Vector3(l_ViewportPoint.x*m_LifeBarCanvasRectTransform.sizeDelta.x, -
   (1.0f-l_ViewportPoint.y)*m_LifeBarCanvasRectTransform.sizeDelta.y,
   0.0f);
}
```

Cambio de escena

Para el cambio de nivel utilizaremos el método SceneManager.LoadScene() o SceneManager.LoadSceneAsync().

Cambio de escena – Implementación

```
using UnityEngine.SceneManagement;

void LoadLevel2()
{
   SceneManager.LoadSceneAsync("Level2");
}
```

Pool de elementos

Tendremos una clase que contendrá una lista (pool) de objetos. El constructor de la clase recibirá el número de elementos de la pool, el prefab del objeto a instanciar y el padre del transform dónde crearemos los objetos de la pool.

Por último tendremos un método que nos devolverá el siguiente objeto de la pool, al llegar al último nos devolverá el primer objeto de la pool.

Pool de elementos – Implementación

```
public class CPoolElements
{
   List<GameObject> m_Elements;
   int m_CurrentElementId=0;

   public CPoolElements(int ElementsCount, GameObject Prefab, Transform Parent)
   {
   }
   public GameObject GetNextElement()
   {
   }
}
```

Delegados y funciones lambda

Un delegado en c# define un tipo con la firma de un método o función, nos va a permitir pasar funciones por parámetro o guardarla en una variable.

Una función lambda es una función anónima que implementamos directamente en la llamada a una función sin necesidad de crear un método.

Delegados – Implementación

```
public delegate void MyFnType(int Parameter);

MyFnType m_MyFnType;
void SetMyFnType(MyFnType _MyFnType)
{
   m_MyFnType=_MyFnType;
}
public void CallFunction()
{
   m_MyFnType(3); // => HelloWorldDelegate(3);
}
void HelloWorldDelegate(int Parameter)
{
   //..
}
SetMyFnType(HelloWorldDelegate);
```

Función lambda – Implementación

```
public delegate void FadeOutFn();

IEnumerator FadeOut(FadeOutFn _FadeOutFn)
{
   yield return new WaitForSeconds(3.5);
   _FadeOutFn();
}

StartCoroutine(FadeOut(()=>
{
   SceneManager.LoadLevelAsync("NextLevel");
}));
```