Práctica 1

Máquina de Estados Finitos

Marcos Somoza Corral – 21711787

ÍNDICE

1.Descripción del proyecto:

El proyecto consiste en crear e implementar una máquina de estados finitos en Unity. En este caso, hemos partido de un proyecto vacío e importado un paquete de modelos 3D desde la asset store llamado “POLYDesert”. Se ha utilizado y decorado una escena de espacio limitado para el proyecto, utilizado un modelo como personaje y dos modelos distintos para las dos máquinas de estados implementadas

2.Resultados:

Se han establecido tres personajes en la escena, uno de ellos como “Player” controlado por teclado y cuya vista sigue al ratón (se ha modificado el script de PlayerController visto en clase con el tutorial de Unity). Uno de los enemigos cuenta con los distintos scripts para crear un sistema “inteligente” y está situado a nivel del suelo. El otro enemigo se encuentra en el aire con su propia máquina de estados.

Al enemigo que se mueve a nivel de suelo lo llamaremos EnemigoSuelo y al enemigo que se mueve por el aire EnemigoAire para una más fácil explicación en este documento.

El EnemigoSuelo tiene los siguientes comportamientos:

1. Patrullar por unos puntos seleccionados previamente.
2. Si el jugador se encuentra dentro de su rango de detección, rotará hasta que un raycast con la dirección de a donde esté mirando el EnemigoSuelo colisione con el jugador.
3. Si el raycast encuentra al jugador pasará a perseguirlo hasta que el jugador salga del rango de detección, en ese caso volverá a patrullar.
4. Si el jugador entra dentro del rango de persecución el enemigo en lugar de rotar para buscarle pasará directamente a perseguirlo.

Estos comportamientos se resumen en los siguientes estados:

1. Estado Patrulla
2. Estado Persecución
3. Estado Alerta

El EnemigoAire tiene los siguientes comportamientos:

1. Patrullar por posiciones aleatorias dentro de un área cuadrada dada si se trata del líder, ya que hay un EnemigoAire más grande que el resto. Funcionan a modo de colmena por lo que los EnemigoAire más pequeños en lugar de patrullar por posiciones aleatorias seguirán al líder.
2. En el caso de que el raycast que sale del enemigo hacia abajo (Vector3.down) encuentre al jugador disparará balas con un ratio de disparo dado hacia abajo.
3. Si el raycast deja de ver al jugador el EnemigoAire volverá a patrullar.

Estos comportamientos se resumen en los siguientes estados:

1. Estado Patrulla
2. Estado Disparo

3.Metodología

Para realizar la práctica se han creado dos scripts de MáquinaDeEstados distintos, uno para cada tipo de enemigo. Estos scripts llaman a scripts separados con una función dependiendo del estado. También se creó un script especial heredado de ScriptableObject, para crear objetos en la carpeta “Assets” de Unity donde guardaremos variables exclusivas de cada instancia de enemigo (su rangoPersecucón, rangoDetección, etcétera). Cabe destacar que el código está explicado en los comentarios de este.

véase en el EnemigoSuelo:

Código máquina de estados

```
using UnityEngine;
using UnityEngine.AI;

public class StateMachine : MonoBehaviour
{
    public enum estados {estadoPatrulla, estadoPersecuccion, estadoAlerta
} //creamos un enum con los diferentes estados


    //Todas las variables y referencias utilizadas
    #region REFERENCES
    public static estados estado = estados.estadoPatrulla;
    public estados estadoActual;
    public AlertScript alert;

    public GameObject[] waypoints;
    public NavMeshAgent agent;
    public Transform player;

    public EnemyStats stats;
    public static EnemyStats enemyStats;

    public static float rotationSpeed;
    public float rotationspeed;

    public static RaycastHit hit;
    public GameObject raycaster;

    public bool playerVision;
    public static bool playerVisto;


    #endregion

    public void Start()
    {
        rotationSpeed = rotationspeed;
        agent.angularSpeed = 300f;
        playerVisto = false;
        playerVision = false;



    }
      public void Update()
    {
```

```
        //switch para cambiar de estados llamando a la función necesaria
de los scripts de cada estado
        switch(estadoActual)
        {
            case (estados.estadoPatrulla):
            PatrolScript.Patrullar(waypoints,agent);
            break;
            case (estados.estadoPersecuccion):
            PursuitScript.Pursuit(agent,player);

            break;
            case (estados.estadoAlerta):
            AlertScript.BusquedaTarget(agent, player, raycaster,estadoAct
ual);
            playerVision = playerVisto; //igualamos una variable estática
 cogida de la clase AlertScript a una local
            break;
        }

        if(playerVisto)
        {
            estadoActual = estados.estadoPersecuccion; //si la variable l
ocal es true pasamos a estado persecucción

        }
        //si el jugador está entre el rango de detección y el rango de pe
rsecucción del enemigo y el enemigo no lo está viendo
        //pasar a estado alerta
        if(Vector3.Distance(this.transform.position,player.transform.posi
tion) <= stats.detectionRange
        && Vector3.Distance(this.transform.position,player.transform.posi
tion) > stats.persecutteRange && !playerVisto
        && estadoActual!=estados.estadoPersecuccion)
        {
            estadoActual = estados.estadoAlerta;
        }
        //Si el jugador está dentro del rango de persecucción se pasa a e
stado persecucción
        else if(Vector3.Distance(this.transform.position,player.transform
.position) <= stats.persecutteRange)
        {
            estadoActual = estados.estadoPersecuccion;
        }
        //Si el jugador está más lejos que el rango de detección pasamos
a estado patrulla, reseteando además
        //la booleana que nos dice si ha visto el enemigo al jugador
        else if(Vector3.Distance(this.transform.position,player.transform
.position) > stats.detectionRange)
```

```
        {
            playerVisto = false;
            playerVision = playerVisto;
            estadoActual = estados.estadoPatrulla;

        }

    }









    void OnDrawGizmos()
    {
        //Dibuja en el editor los rangos de detección, de persecucción ad
emás de el raycast del enemigo

        Gizmos.color = Color.white;
        Gizmos.DrawWireSphere(this.transform.position,stats.detectionRang
e);

        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(this.transform.position,stats.persecutteRan
ge);

        Gizmos.color = Color.green;
        Gizmos.DrawLine(raycaster.transform.position, raycaster.transform
.position + transform.forward * (stats.detectionRange-1));
        Gizmos.color = Color.blue;
        Gizmos.DrawWireSphere(raycaster.transform.position + transform.fo
rward * (stats.detectionRange-1),0.5f);

    }









}
```

Código estado patrulla

```
using UnityEngine;
using UnityEngine.AI;


public class PatrolScript
{

    static int destination = 0;
    public static void Patrullar(GameObject[] waypoints, NavMeshAgent age
nte)
    {
        agente.isStopped = false;
        if(destination >= waypoints.Length) //asegurarse que el nuevo way
point destino esté dentro del array
            destination = 0;
        if(!agente.pathPending && agente.remainingDistance <= 0.1f) //si
no queda camino pendiente hasta el destino y la distancia
        //que queda por recorrer es menor de 0.1f asignar nuevo destino
        {
            agente.autoBraking = false;
            agente.destination = waypoints[destination].transform.positio
n;
            destination = (destination + 1) % waypoints.Length;
        }

    }
}
```

Código estado alerta

```
using UnityEngine;
using UnityEngine.AI;

public class AlertScript
{

    public static float sphereRaduis = 0.5f;

    public static void BusquedaTarget(NavMeshAgent agente, Transform play
er,GameObject Raycaster, StateMachine.estados estado)
    {
        RaycastHit hit;
        agente.isStopped = true;
        agente.updateRotation = true;
        agente.transform.Rotate(Vector3.up); //hacer que el enemigo rote
sobre el eje y

        //agente.transform.RotateAroundLocal(Vector3.up,StateMachine.rota
tionSpeed * Time.deltaTime); -
> el rotateAroundLocal esta obsoleto pero sigue funcionando, hace lo mism
o que Rotate

        if(Physics.Raycast(Raycaster.transform.position,Raycaster.transfo
rm.forward,out hit, 10f))
        {
            //casteo un raycast desde el enemigo si encuentra al jugador
con tag player el enemigo se queda mirandolo
            //y activamos la variable playerVisto a true, condicionante p
ara cambiar de estado
            if(hit.collider.gameObject.tag == "Player")
            {
                agente.transform.LookAt(hit.transform.position);
                StateMachine.playerVisto = true;
            }
        }
    }
}
```

Código estado persecución

```
using UnityEngine;
using UnityEngine.AI;

public class PursuitScript
{
    public static void Pursuit(NavMeshAgent agente, Transform player)
    {
        agente.isStopped = false;
        agente.destination = player.position; //asignar el nuevo destino
en la posición del jugador

    }
}
```

Además de los scripts de la máquina de estados, he añadido un script para que el enemigo ataque al jugador al colisionar con él:

Código ataque

```
using UnityEngine;

public class EnemyAttackScript : MonoBehaviour
{

    //si en enemigo colisiona con el jugador con tag player el jugador es
 destruido
    void OnCollisionEnter(Collision collision)
    {
            if(collision.gameObject.name == ("player"))
            Destroy(collision.transform.gameObject);

    }
}
```

Como se ha podido comprobar, el único script de la máquina de estados que hereda de monobehaviour es “StateMachine”, el resto de scripts son referenciados a este para no tener que añadirlos al objeto EnemigoSuelo; por ello y para poder enviar leer, dar valor y usar funciones entre scripts todos aquellos que no heredan de monobehaviour son de tipo static.

Ahora veamos la máquina de estados del EnemigoAire:

Código Máquina de estados voladora

```
using UnityEngine;
using UnityEngine.AI;

public class StateMachineFlying : MonoBehaviour
{
    public enum Estados  {estadoPatrulla, estadoDisparo} //enum con los d
iferentes estados

    //todas las variables necesarias
    #region REFERENCES

    public GameObject lider;
    public NavMeshAgent agente;
    public Transform player;
    public Estados estadoActual = Estados.estadoPatrulla;
    public GameObject laser;
    public float fireRate = 3f;
    private float timepassed;
    #endregion



    public void Update()
    {

        //switch con los diferentes estados
        switch(estadoActual)
        {
            //en el caso de patrulla, diferenciamos según el tag si se tr
ata del enemigo líder o el secuaz,
            //dependiendo de cual sea llamaremos a una u otra función de
la clase RandomPatrol
            case Estados.estadoPatrulla:
            if(this.gameObject.tag !="Leader")
                RandomPatrol.FollowLeader(agente, lider);
            RandomPatrol.RandomPatrolFunction(agente);
            break;

            //en estado disparo llamaremos a la función EstadoDisparo()
            case Estados.estadoDisparo:
            EstadoDisparo();
            break;

        }
```

```
        timepassed += Time.deltaTime;

        //si la distancia entre el jugador y el enemigo es menor o igual
a 6 unidades pasamos a estado disparo
        if(Vector3.Distance(this.transform.position,player.position) <= 6
f)
            estadoActual = Estados.estadoDisparo;

        //si es superior a 6 unidades pasamos a estado patrulla
        else
            estadoActual = Estados.estadoPatrulla;
    }

    //Dado que para instanciar necesitamos una clase que herede de monobe
haivour en lugar de crar una clase separada
    //para estado disparo haremos una función en StateMachineFlying : Mon
obehaivour.
    public void EstadoDisparo()
    {
        //Hacemos que el enemigo deje de moverse y un contador de tiempo
real comparandolo con la variable fireRate
        //para que cada 3sg en este caso (en función del valor de la vari
able FireRate) se instanciará una bala.
        agente.isStopped = true;

        if(timepassed >= fireRate)
            {
                timepassed = 0;
                GameObject instancia = Instantiate(laser, new Vector3(thi
s.transform.position.x,this.transform.position.y-
0.4f,this.transform.position.z) ,Quaternion.identity);
                instancia.GetComponent<Rigidbody>().AddForce(Vector3.down
);
            }
    }
    void OnDrawGizmos()
    {
        //dibuja en el editor una línea desde el enemigo hasta el suelo.
        Gizmos.color = Color.magenta;
        Gizmos.DrawLine(this.transform.position,this.transform.position +
 Vector3.down * 6f);


    }
}
```

Código patrulla aleatoria

```
using UnityEngine;
using UnityEngine.AI;

public class RandomPatrol
{

//Función adscrita al resto de enemigos secuaces, éstos siguen al enemigo
 líder si están más lejos de 8 unidades,
//si se acerca más de 8 unidades, el nuevo destino es el secuaz mismo, po
r lo que se queda quieto, si además de
//acercarse a más de 8 unidades no tiene camino por recorrer, lo alejamos
 dándole un destino aleatorio nuevo.
public static void FollowLeader(NavMeshAgent agente, GameObject lider)
{
    if(Vector3.Distance(agente.transform.position,lider.transform.positio
n) >8f)
    agente.destination = lider.transform.position;
    if(Vector3.Distance(agente.transform.position,lider.transform.positio
n) <=8f)
    agente.destination = agente.transform.position;
    if(Vector3.Distance(agente.transform.position,lider.transform.positio
n) <=8f && !agente.pathPending)
        agente.destination = RandomPointInBox(new Vector3(-
276.5f,10.53614f,48.00449f),new Vector3(37.70092f,37.70092f,37.70092f));
}

//Función adscrita solamente al enemigo líder, le da un destino en funció
n de RandomPointInBox() con los parámetros
//del tamaño de la escena
public static void RandomPatrolFunction(NavMeshAgent agente)
    {
        agente.isStopped = false;
        if(!agente.pathPending && agente.remainingDistance <= 0.1f)
        {
            agente.autoBraking = false;
            agente.destination = RandomPointInBox(new Vector3(-
276.5f,10.53614f,48.00449f),new Vector3(37.70092f,37.70092f,37.70092f));

        }

    }

    //Función que devuelve una posición aleatoria (en vector3) de área da
da
    private static Vector3 RandomPointInBox(Vector3 center, Vector3 size)

    {
```

```
            return center + new Vector3(
                (Random.value - 0.5f) * size.x,
                (Random.value - 0.5f) * size.y,
                (Random.value - 0.5f) * size.z
            );
        }
}
```

Como se ha mencionado anteriormente, este enemigo dispara y para optimizar el programa se ha creado un script que elimina el objeto bala instanciado al colisionar:

Código bala colisiones

```
using UnityEngine;

public class BulletSelfDestruction : MonoBehaviour
{

    //este script se incorpora a las balas que instancia este tipo de ene
migo en su estadoDisparo
    void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.name == ("player"))
        {
            Destroy(collision.gameObject);
        }
        Destroy(this.gameObject);
    }
}
```

Como se ha explicado antes, se ha creado un script heredado de ScriptableObject donde almacenamos variables específicas de cada enemigo (dado que se han instanciado en escena más de un enemigo con la misma máquina de estados):

Código estadísticas de enemigo

```
using UnityEngine;


[CreateAssetMenu(menuName = "EnemyStats")] //sirve para habilitar dentro
de unity (con el click derecho)
//la creación de un objeto scriptable EnemyStats
public class EnemyStats : ScriptableObject
{
    //clase Scriptable para poder crear objetos en la carpeta Assets, don
de guardo variables
    //específicas de cada enemigo

    public float speed;
    public float detectionRange;
   public float persecutteRange;


}
```

Ahora veremos un par de códigos extra para una mejor experiencia. En el caso de GizmoWaypoint es un script exclusivo para ver en el editor de unity (scene view) exactamente por dónde patrulla cada instancia de EnemigoSuelo:

Código gizmo puntos de patrulla

```
using UnityEngine;

public class GizmoWaypoint : MonoBehaviour
{

    //función exclusiva para dibujar en el editor los waypoints de cada e
nemigo

    public GameObject padre;
    void OnDrawGizmos()
    {
        if(padre.name == "Enemy1")
            Gizmos.color = Color.green;
        if(padre.name == "Enemy2")
            Gizmos.color = Color.blue;
        if(padre.name == "Enemy3")
            Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(this.transform.position,0.1f);
    }
}
```

Por último, un script que administra si el jugador ha sido destruido o no para reiniciar la escena:

Código manejador de escena

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneManagerScript : MonoBehaviour
{
    public GameObject player;
    //Script que busca si el jugador ha sido destruido cada frame, si es
así se reinicia la escena
    void Awake()
    {
        player = GameObject.Find("player");
    }
    void Update()
    {
        if(player == null)
            SceneManager.LoadScene("MainScene");
    }
}
```

4.Visualización gráfica de los scripts

He subido un vídeo del proyecto tanto desde el punto de vista del editor de escena de unity como desde la vista cenital ingame : https://www.youtube.com/watch?v=htEb8XlsAWg&feature=youtu.be

Por supuesto, tanto la build como el proyecto en sí están en gitlab disponibles para su descarga y visualización:

https://gitlab.com/msomele/statemachine_somoza