

IA en Videojuegos

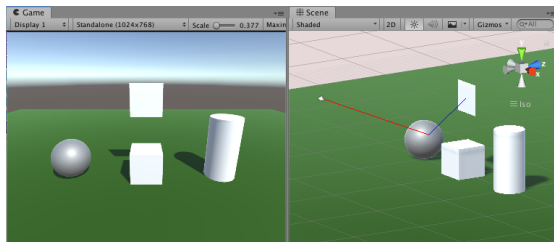
Cuarto Proyecto

L. Daniel Hernández¹

Departamento de Ingeniería de la Información y las
Comunicaciones

Universidad de Murcia

25 de enero de 2022



¹Erratas y sugerencias a ldaniel@um.es. Algunas imágenes pueden ser de internet. Si tuvieran derechos de autor/distribución conocidos, por favor, avisen.

Índice de Contenidos

- 1 La Escena
- 2 GetComponent<>()
- 3 Me tocan
- 4 Te toco
- 5 Crear instancias de objetos
- 6 Indicar objetos seleccionados con el ratón

La Escena

La escena

La escena de esta sesión constará de los siguientes elementos (ver la portada), para los que no deberías tener ningún problema.

- Un suelo, llamado Plane, formado por un plano con material de color verde (u otro color que contraste con el azul y el rojo).
- Un cubo, llamado Cube: Position (0, 0, 0); Scale (4, 5, 5).
- Una esfera, llamada Sphere: Position (-10, 4, 0); Scale (5, 5, 5).
- Un cilindro, llamado Cylinder: Position (10, 4, 0); Scale (4, 4, 4).
- Un quad, llamado Quad: Position (0, 10, 0); Scale (4, 4, 1).
- Añadir a Sphere, Cylinder y Quad una componente Rigidbody con Use Gravity marcado.
- La cámara principal tendrá Position (0, 13, -22) y Rotation (19, 0, 0).
Tendrás una perspectiva similar a la imagen izquierda de la portada de estas transparencias.

En la ventana escena, visualiza las 4 figuras sobre el suelo y la cámara. Procura que tenga una perspectiva similar a la imagen derecha de la portada.

GetComponent<>()

GameObject.GetComponent

La clase `GameObject` es la base para todas las entidades de una escena. Todas tienen componentes (como mínimo `Transform`). Se puede extraer información de todas las componentes de un objeto (u objeto padre y objeto hijo).

El método `GetComponent` retorna la componente del tipo indicado si el objeto tiene tal componente y retorna `null` si el objeto no tiene tal componente.

Manual:



`GameObject`



`Using Components`

Scripts:



`GameObject`



`GameObject.GetComponent`

GetComponent: un ejemplo

Objetivo: obtener las referencias de las componentes Rigidbody y Mesh Renderer de un objeto, mostrando sus valores en las dos variables públicas.

Añadir el siguiente script al objeto Cylinder (u otro) y verás que en tiempo de ejecución se modifican los valores Rigidbody.mass y MeshRenderer.Materials[0].

```
1 public class GetComponent : MonoBehaviour
2 {
3     public float mass = 0;      // public information
4     public Material material = null; // public information
5
6     Rigidbody rb = null; // Reference of the Rigidbody
7     MeshRenderer mr = null; // Reference of the MeshRenderer
8
9     // Start is called before the first frame update
10    void Start()
11    {
12        rb = GetComponent<Rigidbody>(); // Get Rigidbody from this.GameObject
13        mr = GetComponent<MeshRenderer>(); // Get MeshRenderer from this.GameObject
14    }
15
16    // Update is called once per frame
17    void Update()
18    {
19        if (rb != null) mass = rb.mass; // Modify component Rigidbody and observe
20        if (mr != null) material = mr.material; // Modify component MeshRenderer and observe
21    }
22 }
```

Me tocan

OnMouseXXX

Una forma de interactuar durante el juego con sus objetos es utilizando las funciones `OnMouseXXX()`. Todo objeto que tenga un Collider y estas funciones en una componente script podrá cambiar su comportamiento si el puntero del ratón pasa “sobre él” en la ventana de juego. El objeto se “siente” tocado. Añade el script al objeto Quad.

```
1 public class OnMouse : MonoBehaviour
2 {
3     public Renderer rend;
4
5     void Start()
6     {
7         rend = GetComponent<Renderer>();
8     }
9
10    void OnMouseEnter() // The mesh goes red when the mouse is over it...
11    {
12        rend.material.color = new Color(1, 0, 0); //Color.red;
13    }
14
15    void OnMouseOver() // ...the red fades out to blue as the mouse is held over...
16    {
17        rend.material.color += new Color(-.5f, 0, .5f) * Time.deltaTime;
18    }
19
20    void OnMouseExit() // ...and the mesh finally turns white when the mouse moves away.
21    {
22        rend.material.color = Color.white;
23    }
24 }
```



Te toco

La idea

En vez de que un objeto se “sienta” tocado, se puede tener la perspectiva del que lo toca. P.e. cuando el jugador dispara a un objeto. Esto se consigue mediante el lanzamiento de rayos. La idea es:

- Colocar el ratón en un lugar de la ventana de juego. Se corresponderá con una coordenada de la pantalla.
- Determinar cuál es el punto que le corresponde en el mundo de la escena.
- Si está tocando algún objeto obtener información de ese objeto como posición, lugar de impacto, etc ...

Unity dispone de métodos que permiten realizar todas estas tareas.

Vamos a crear un Script que permite mostrar en la ventana de la escena un rayo que va desde el punto de la ventana donde se encuentre “tocando” el ratón a un objeto hasta el punto de incidencia sobre ese objeto, para a continuación mostrar un rayo que es normal a la superficie en el punto de incidencia. También cambiaremos el material del objeto que está siendo tocado por el ratón.

Rayos y golpes

Para lanzar un rayo desde la ventana al escenario y obtener información del impacto, implementa esta función en el nuevo script `RayTraceScreen.cs`:

```
1 void Update()
2 {
3     // Returns a ray going from camera through a screen point.
4     Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
5     RaycastHit hit;
6
7     // Other: .Linecast() .BoxCast() .SphereCast() .CapsuleCast()
8     if (Physics.Raycast(ray, out hit))
9         draw(ray, hit); // Dibujar los rayos.
10 }
```



Camera



Physics.Raycast



Ray



RaycastHit

Añade el script `RayTraceScreen.cs` a la cámara principal.

Dibujar los rayos

Con la información del rayo y el impacto se puede hacer lo que se quiera. En nuestro caso, dibujar el rayo desde la pantalla, el rayo normal al punto de impacto y cambiar de color el objeto golpeado por el rayo. Añade este código al script `RayTraceScreen.cs`:

```
1 void draw(Ray ray, RaycastHit hit)
2 {
3     // The hit object is not the plane
4     string str = hit.transform.gameObject.name;
5     if ( !(str.Equals("Plane") || str.Equals("Quad")))
6     {
7         Debug.DrawLine(ray.origin, hit.point, Color.red);
8         Debug.DrawLine(hit.point, hit.point + 20*hit.normal, Color.blue);
9     }
10
11     // we will change the color, if possible
12     changeColor(hit);
13 }
```



Debug



Color

Cambiar la textura

Añade este código al script RayTraceScreen.cs para cambiar el color del objeto mientras el ratón esté “encima” de él. Ejecuta el juego.

```
1 private GameObject firstThing = null;
2 private GameObject secondThing = null;
3
4 MeshRenderer m_Renderer = null;
5 Color m_OriginalColor = Color.green;
6
7 void changeColor(RaycastHit hit)
8 {
9     string str = hit.transform.gameObject.name;
10    if (firstTime && ! (str.Equals("Plane") || str.Equals("Quad")))
11    {
12        firstThing = hit.transform.gameObject;
13        m_Renderer = firstThing.GetComponent<MeshRenderer>();
14        m_OriginalColor = m_Renderer.material.color;
15        m_Renderer.material.color = Color.gray;
16        firstTime = false;
17        return;
18    }
19
20    if (firstThing == null) return;
21
22    secondThing = hit.transform.gameObject;
23    if (firstThing == secondThing) return;
24
25    m_Renderer.material.color = m_OriginalColor;
26
27    firstTime = true;
28 }
```

Crear instancias de objetos

La idea


En tiempo de ejecución se puede crear o destruir nuevos objetos.

Se quiere que al hacer click sobre un objeto aparezca un cubo girando, pero si el cubo ya estaba entonces destruirlo. Esta idea se puede usar para indicar qué objetos han sido seleccionados con el ratón.

Los pasos son:

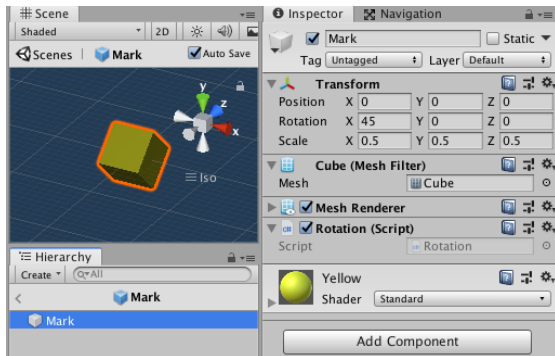
- Crear un script que haga girar a un objeto.
- Crear un prefab con el cubo y el script anterior.
- Crear un script que cree un nuevo objeto hijo sobre el game object sobre el que se ha hecho click o destruirlo si existiera.

Crear el prefab del objeto marcador

El algoritmo de rotación que se usará es muy simple:  Rotate

```
1 public class Rotation : MonoBehaviour
2 {
3     void Update()
4     {
5         // Rotate around the World's Y axis
6         transform.Rotate(Vector3.up * Time.deltaTime * 1000, Space.World);
7     }
8 }
```

El prefab que se debe construir no tiene collider. Copia los valores de la captura. Llámalo Mark y guárdalo en la carpeta Prefabs.



Añadir o destruir el objeto marcador

Crea un script con las siguientes funciones y añádelo a la cámara principal.

```
1 void Update()
2 { // the user pressed down the virtual button identified by "Fire1".
3   if (Input.GetButtonDown("Fire1")) {
4     Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
5     RaycastHit hit;
6     if (Physics.Raycast(ray, out hit)) mark(hit.transform.gameObject);
7   }
8 }

10 private void mark(GameObject thing) {
11   GameObject marker = null;

13   // If there is a child in the object called Mark
14   if (thing.transform.Find("Mark") != null) // Get your reference
15     marker = thing.transform.Find("Mark").gameObject;

17   // If there is no reference then
18   if (marker == null) {
19     // create an instance of the prefab «« CREATE INSTANTIATE
20     marker = Instantiate(markSpecial, thing.transform);

22     marker.transform.localPosition = Vector3.up * 1; // Change your position relative
23     marker.name = "Mark"; // Change your name

25   } else // If there is no reference then
26     Destroy(marker); // Destroy the object «« DESTROY
27 }
```



Input



Instantiate



Destroy

Indicar objetos seleccionados con el ratón

La idea

Se quiere que al hacer click sobre un objeto aparezca “algo” que indique que ha sido seleccionado.

Las 2 transparencias anteriores es una solución a este problema. Otra solución es considerar que el cubo es un objeto hijo del GameObject que se activa o desactiva según se seleccione o no.

```
1 void Update()
2 { // the user pressed down the virtual button identified by "Fire1".
3     .....
4     if (Physics.Raycast(ray, out hit))
5         enable(hit.transform.gameObject);
6 }
7
8 private void enable(GameObject thing)
9 {
10     bool show = false;
11     Transform marker = thing.transform.Find("Mark"); // Find Mark.
12     show = marker.GetComponent<Renderer>().enabled;
13
14     // Change check enabled.
15     marker.GetComponent<Renderer>().enabled = !show;
16     marker.GetComponent<Rotation>().enabled = !show;
17 }
```

Otras consideraciones

- Considerar varios Prefab como indicadores. Cada objeto que pueda ser seleccionado tienen un script con una variable pública que almacena la referencia a un prefab. Cuando se selecciona el objeto se muestra el objeto al que hace referencia la variable pública.
- Crear una script/clase con una variable estática que almacene una lista pública de las referencias de los objetos que son seleccionados. Según se haga click sobre un objeto, se añade o se suprime de dicha lista.
- Puedes sobrescribir el método `MonoBehaviour.OnGUI()` para poner etiquetas sobre el objeto que han sido “tocados” utilizando el método `GUI.Box (new Rect (a, b, c, d), texto)`.



`MonoBehaviour.OnGUI()`



`Immediate Mode GUI (ImGui)`

- Aparte de rayos, se puede lanzar o una línea, o un cubo, o una esfera o una cápsula; lo que permite abarcar una mayor cantidad de posibles puntos de impacto.

Los métodos son: `.Linecast()` `.BoxCast()` `.SphereCast()` `.CapsuleCast()`

Selección de varias unidades

- Considera el script incompleto que se llama `PropuestaOrdenarIrAUnLugar.cs`
- Observa lo que hace el código.
- En tu escenario añade el tag "Terrain.^al suelo y el tag "NPC.^a todos los objetos que sean personajes.
- Modifica el código para que cada vez que se seleccione un objeto se añada a una lista.
- En el caso de que el personaje ya se encuentre en la lista, haz que el objeto salga de la lista.
- Estos principios básico serán necesario para hacer formaciones más adelante.