Computació Gràfica i Mutimèdia

Introduction to 2D Graphics in Unity

Master's Degree in Informatics Engineering

2024/2025 year

This session is a hands-on introduction to 2D graphics in Unity.

These tasks have been designed as a self-learning activity. You do not have to present the results to the professor.

Create a new project and a basic 2D scene

1. Launch Unity and create a new 2D project.
2. You can add 2D objects to the scene. "GameObject" → "2D Object" → "Sprites" → "Square" or "Circle". Add a square.
3. Play the game and see the result.
4. Now, select the Camera in the "Hierarchy" window, and in the "Inspector" window modify its size. This modifies the camera zoom. Test it.
5. Select some object and, in the inspector, change its color. Test it.

Scripts

Unity is a component-based environment. You can provide additional features to your objects by adding components to it. Scripts are an example.

1. Create a new script "Assets" → "Create" → "C# Script".
2. Before any other action, assign a name to the new script. For instance "BasicMovement".
3. Double-click on it in the "Assets" window.
4. Add the following code to its Update() method. This function is called once per frame.

```
if (Input.GetKey(KeyCode.A))
 transform.position = transform.position + new Vector3(-1.0f * Time.deltaTime, 0.0f, 0.0f);
```

   Note: "Time.deltaTime" returns the time elapsed since the last frame.

5. Select one of your objects and drag the script onto its inspector area. Now, play the game and hold the "A" key.
6. Add code to your script so that the object can also move right, up and down. In 2D-graphics, the z-coordinate (the third one) is of little importance. Test it.
7. Add the following code to your script and test the result.

```
if (Input.GetKey(KeyCode.Q))
 transform.eulerAngles = transform.eulerAngles + new Vector3(0.0f,0.0f,Time.deltaTime * 90.0f);
else if (Input.GetKey(KeyCode.E))
```

```
transform.eulerAngles = transform.eulerAngles + new Vector3(0.0f,0.0f,-Time.deltaTime * 90.0f);
```

8. Add an additional square. Play the game and make the moving square collide against it.

Physics-enabled movement

1. Remove the “BasicMovement” script from your polygon.
2. In the inspector, add a “Box collider 2D” component to your polygon. Colliders are used for detecting collisions between objects. They do not react to them.
3. In the inspector, add a “Rigidbody 2D” component to your polygon. Rigidbodies react to collision events detected by colliders.
4. Now, play the game. You can disable the effect of gravity from the inspector. Set “Gravity scale” to 0.
5. Now, create a new C# script named “PhysicsMovement”.
6. Add the following code to it. This code creates a reference to a rigidbody component contained in the game object the script is attached to.

```
Rigidbody2D rb;

void Start()
{
 rb = GetComponent<Rigidbody2D>();
}
```

7. Now, you can move your object as follows. Put this code inside the “update()” method:

```
if (Input.GetKey(KeyCode.W))
 rb.velocity = new Vector3(0.0f, 5.0f, 0.0f);
```

8. Add code so that your object can move into all the directions and it stops when no key is pressed.
9. Add an additional polygon to the scene. Add a 2D collider to it, and make your object collide against it. Test it.
10. Now, add a Rigibody2D to the additional polygon. Disable its response to gravity and, in the inspector, set “Linear Drag” = 1 and “Angular Drag” = 1. See what happens when you collide against it.

Triggers

Colliders can be configured to act as trigger. A trigger is a collider that generates an event when another collider overlaps it. Once a collider is set as a trigger, it no longer generates collisions.

1. Select the secondary polygon in your scene and, from the inspector, enable its collider to act as a trigger.
2. Now, create a new C# script named “TriggerResponse”.

3. Add the following code to it:

```
SpriteRenderer rd;

void Start()
{
 rd = GetComponent<SpriteRenderer>();
}

void OnTriggerEnter2D(Collider2D other)
{
 rd.color = new Color(1f, 0f, 1f, 1f);
}
```

4. Make this script be a component of the secondary polygon. Then, play the game and make your moving object intersect it.
5. When an object stops intersecting a trigger, an event is generated which can be processed by a procedure with the following definition:

```
void OnTriggerExit2D(Collider2D other)
```

6. Implement the "TriggerResponse" script so that the secondary polygon takes back its original color when the moving one stops intersecting it.

Object spawning

Unity allows the possibility of creating objects in a dynamic fashion.

1. Create a new polygon, name it "Child" and color it blue.
2. Select it at the hierarchy window and drag it into the assets area. Now, you have created a "prefab". You can drag the asset into the scene in order to create more instances of it. Test it.
3. Create a new C# script named "SpawnControl". Add the following code to it:

```
public class SpawnControl : MonoBehaviour
{
    [SerializeField] public GameObject pref;

(···)

void Update()
{
 GameObject obj;

 if(Input.GetKeyDown(KeyCode.Space))
 {
  obj = Instantiate(pref) as GameObject;
  obj.transform.position = transform.position;
 }
}
```

4. Make this script be a component of your moving polygon. In the inspector, drag the previous asset to the "Pref" slot.
5. Play the game and see what happens when you press the space key.

Destroying objects
Objects can be destroyed dynamically.

1. Create a C# script and name it “TTLScript”.
2. Add the following code to it:

```
public class TTLScript : MonoBehaviour
{
    float countDown = 3.0f;
(...)
void Update()
{
 countDown -= Time.deltaTime;
 if (countDown <= 0.0f)
  Destroy(this.gameObject);
}
```

3. Make this script be a component of the “Child” prefab. Test the result.

Mouse-based interface
Unity can capture and react to events arising from mouse activity.

1. Take your moving square and remove its “PhysicsMovement” script from it.
2. Create a new script and name it “MouseMovement”. Add the following code to it.

```
public class MouseMovement : MonoBehaviour
{
    Rigidbody2D rb;
    Vector3 destinyPoint;

bool moving = false;
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            destinyPoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);
            destinyPoint.z = 0;
            moving = true;
        }

        if (moving)
        {
            if (Vector3.Distance(transform.position, destinyPoint) >= 0.1f)
                rb.velocity = 10.0f * (destinyPoint - transform.position);
            else
            {
                rb.velocity = new Vector3(0.0f, 0.0f, 0.0f);
                moving = false;
            }
        }
    }
}
```

3. Make this script be a component of your moving polygon. Test the result.