Desenvolvimento de Videojogos

Licenciatura em Engenharia Informática – 2020/2021

Guia de Laboratório nº.2

Unity* – Introdução à estrutura de componentes e *scripting

Introdução e Objetivos

No laboratório anterior, fez-se uma introdução ao motor *Unity*, onde se aprendeu a criar uma cena, compondo-a com diferentes *Game Objects*. Neste trabalho de laboratório pretende-se conhecer o conceito de componente, o qual é fundamental em *Unity*, bem como introduzir o mecanismo de *scripting*.

Preparação teórico-prática

Conceito de componentes

Vimos anteriormente que uma cena é composta por *Game Objects*. Por exemplo, podemos criar um cubo através do menu *Game Object* → *3D Object* → *Cube*. Cada *Game Object* é composto por diversos componentes. Estes são apresentados no inspetor quando o objeto é selecionado na cena ou na hierarquia. Na Figura 1 é apresentado o inspetor ao selecionar-se um cubo após a sua criação.

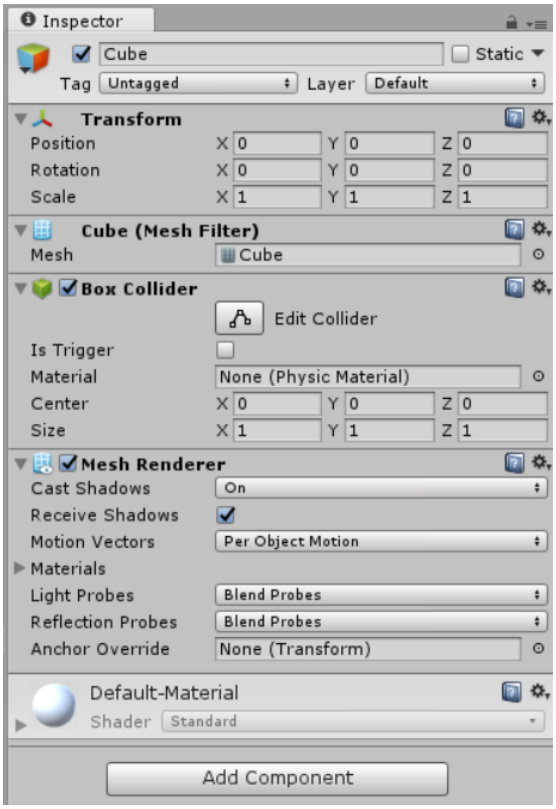

Figura 1 – Componentes base que fazem parte de um cubo

O cubo criado possui 4 componentes:

- *Transform*, que é o componente responsável por armazenar a informação espacial do objeto (a sua posição, escala e orientação). Este componente está presente em todos os *Game Objects*.

- *Mesh Filter*, o componente que processa um modelo em formato de malha poligonal. Neste caso, este componente foi definido automaticamente com um modelo predefinido para um cubo.

- *Box Collider*, uma primitiva em forma de cubo que é utilizada para deteções de colisões.

- *Mesh Renderer*, um componente que recebe o modelo do *mesh filter* e desenha-o no ecrã.

É possível apagar, copiar, cortar e colar componentes dentro de um objeto através do menu apresentado na Figura 2, o qual é acessível clicando com o botão direito do rato no nome do componente ou clicando na roda dentada no canto superior direito do respetivo componente.

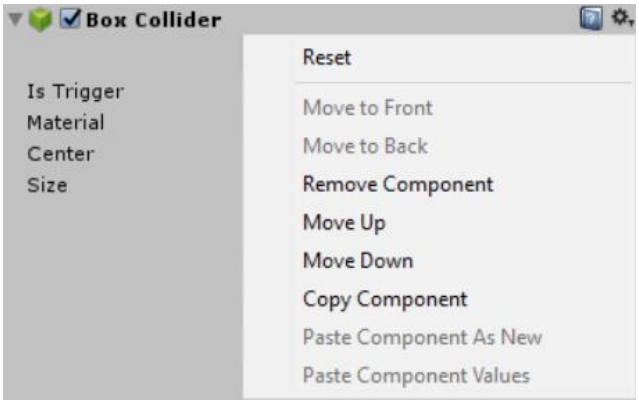

Figura 2 – Opções sobre componentes

Script

Script é um componente particular que podemos associar a um objeto para programar o seu comportamento. Este pode ser programados em *C#*, *JavaScript* ou *Boo Script*.

A criação de um novo *script* pode ser feita da forma apresentada na Figura 3. Neste caso está a ser criado um novo *script* em *C#*. O script criado fica disponível como *Asset* do nosso projeto, tal como se apresenta na Figura 4.

Existem diversas formas de associar o script ao cubo anteriormente criado, nomeadamente:

- Selecionar o cubo e arrastar o *Asset* do script para a zona de componentes do inspetor

- Selecionar o cubo e clicar na opção *Add Component* que aparece na parte de baixo do inspetor, selecionando depois *Scripts* e o nome do *script* criado.

Podemos editar um *script* fazendo duplo clique no respetivo *Asset*. O ficheiro é aberto no IDE que estiver definido.

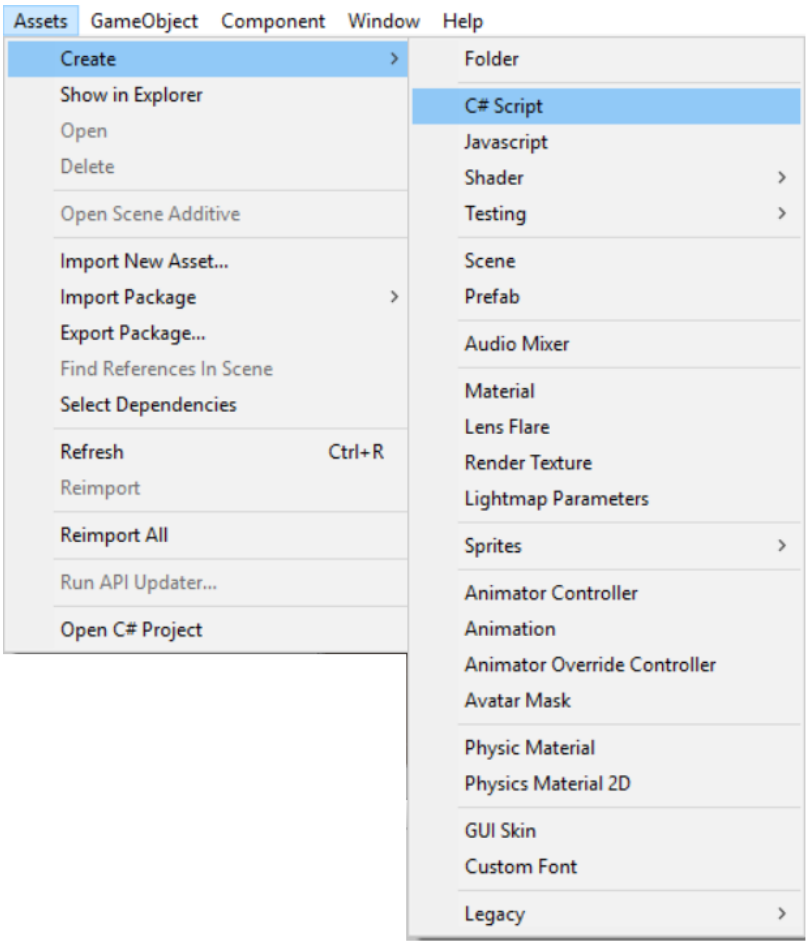

Figura 3 – Criação de um script C# a partir do menu principal

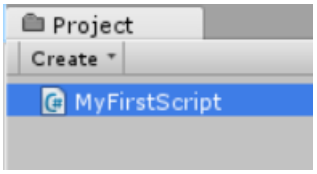

Figura 4 – Script na janela de Assets do projeto

Um script é construído por defeito com o seguinte código:

```
using UnityEngine;

public class MyFirstScript : MonoBehaviour {

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update () {

	}
}
```

Tendo em conta o código criado, e que ainda não faz nenhuma ação, há já a ter em conta os seguintes aspetos:

- Observando a primeira linha, podemos ver que é utilizada a biblioteca (mais corretamente, o *namespace*) *UnityEngine*, onde se encontram as classes fundamentais do motor *Unity*.

- Tecnicamente, um *script* é uma classe. Na execução da aplicação, quando um *Game Object* é criado, são também criados objetos das classes dos seus componentes.

- Os *scripts* associados a *Game Objects* são classes que derivam de *MonoBehaviour*, uma classe que possui um conjunto de métodos e atributos adequados para serem utilizados como componentes representativos do comportamento de um *Game Object*.

- Não sendo obrigatório, é frequente que um *MonoBehaviour* tenha métodos *Start* e *Update*. O método *Start* é executado automaticamente quando o *GameObject* é criado e o método *Update* é executado automaticamente sempre que é desenhada uma nova *frame* no ecrã.

Exemplo – Colocação de um cubo a rodar

O *Unity* cria a sua cena por defeito com uma câmara e uma fonte de iluminação. Antes de prosseguirmos para a edição do script, é importante configurar a posição da câmara e do cubo criado para que este último esteja visível de um ângulo adequado, como se pode observar na Figura 6. Neste exemplo, o cubo mantém-se colocado na origem e a câmara possui a transformação apresentada na Figura 5.

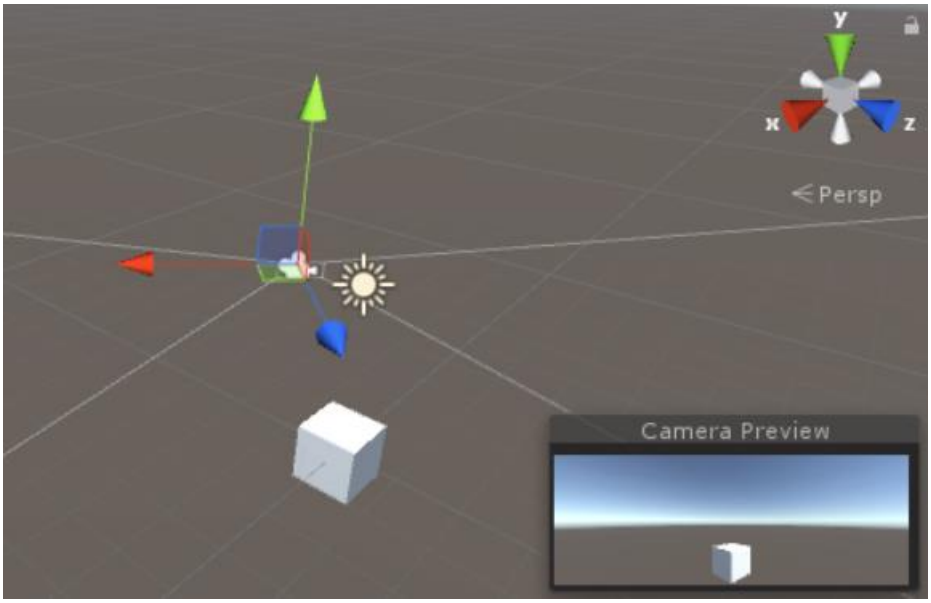

Figura 5 – Posicionamento da câmara para observação de um objeto

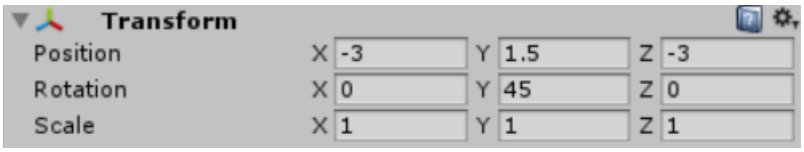

Figura 6 – Configuração do componente Transform da câmara para ficar orientado para o cubo criado no exemplo

Voltando à edição do script, podemos alterar o método *update* da seguinte forma:

```
void Update ()
{
    transform.Rotate(0, 1, 0);
}
```

Em relação ao script, é importante referir o seguinte:

- Por herança, o nosso script tem acesso a atributo *transform*, no qual temos a informação do respetivo componente.
- O atributo *transform* tem um conjunto de métodos para controlar a transformação do objeto, nomeadamente reposicionar, rodar, mudar de posição, entre outras funcionalidades. Neste exemplo, estamos a aceder ao método *Rotate*.
- Uma das assinaturas do método *Rotate* tem 3 parâmetros, relativos a cada um dos eixos cartesianos. Neste caso, a chamada do método roda o objeto 1 grau sobre o eixo dos Y.
- Como o método *update* é chamado uma vez por *frame*, o cubo roda continuadamente. Podemos verificar este comportamento executando o programa (botão *play* na barra de ferramentas).

Este primeiro exemplo permite-nos ver o funcionamento de um script para manipular um objeto. No entanto, tem algumas limitações e alguns aspetos que devem ser melhorados, tal como vamos ver em seguida.

Em primeiro lugar, a velocidade de rotação está dependente da capacidade do computador. Num computador que consiga desenhar a cena a uma velocidade de 60 *frames* por segundo (fps), o cubo rodará 60 graus por segundo, mas num computador que desenhe a cena a 120 *frames* por segundo a velocidade de rotação duplicará. Adicionalmente, em aplicações gráficas é comum ter-se um número de *frames* por segundo variável, consoante a complexidade dos constituintes da cena.

Para resolver o problema anterior, este tipo de alterações deve ser feita em proporção ao tempo que passou desde que a última *frame* foi desenhada. O *Unity* tem uma classe estática *Time* que faculta uma propriedade *deltaTime* que representa precisamente esse intervalo de tempo.

Podemos assim alterar a linha de código definida anteriormente por esta nova versão:

```
    transform.Rotate(0, 90 * Time.deltaTime, 0);
```

Assim, temos por *frame* uma rotação de 90 graus a multiplicar pelo tempo que passou desde que a cena foi desenhada. Na prática, significa que o cubo irá rodar a uma velocidade de 90 graus por segundo.

Execute novamente a aplicação para verificar que o cubo continua a rodar. Dependendo do computador, poderá notar uma rotação com um comportamento mais uniforme.

É boa prática de programação não ter o valor de constantes colocadas diretamente no código (chamadas normalmente por "*magic numbers*"). Relembrando que um script é uma classe, faz sentido que o valor 90, que representa a velocidade de rotação do cubo, seja armazenado como um atributo. O script pode então passar a ser o seguinte:

```
public class MyFirstScript : MonoBehaviour
{
    private float RotationSpeed = 90.0f;

    void Update ()
    {
        transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
    }
}
```

Expor atributos para utilização no editor

Vamos agora ver como podemos interligar os atributos de uma classe com a interface do *Unity*. Considere-se que pretendemos ter vários cubos na cena, a rodar a velocidades diferentes. Naturalmente, não faz sentido fazer um script diferente para cada cubo quando a diferença está no valor de um atributo. É possível expor um atributo de uma classe para o editor *Unity*, bastando para isso colocar o atributo como público. Como esta abordagem é utilizada fundamentalmente em campos que podem ser vistos como parâmetros de inicialização, não existe por norma o problema de comprometer o encapsulamento da classe.

Tornando o atributo publico, este aparece automaticamente junto ao respetivo componente no inspetor, como se pode observar na Figura 7.

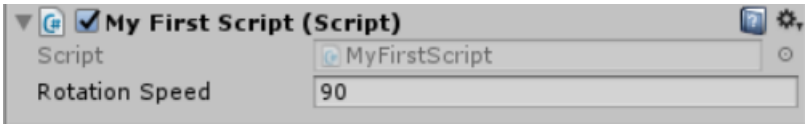

Figura 7 – Exemplo de um componente Script com uma propriedade exposta

Nessa mesma janela podemos agora definir um valor para a propriedade *Rotation Speed*. O valor definido é fornecido ao objeto da classe aquando da execução do programa.

Para consolidar os conhecimentos, faça as seguintes tarefas:

- Teste a aplicação colocando vários valores na velocidade.

- Crie um segundo cubo (ou outra forma primitiva) e adicione-lhe também o *script* criado. Configure valores de velocidade de rotação diferentes nos dois objetos e confirme que ambos rodam a velocidades diferentes.

Classes de Input

Numa aplicação interativa, o mecanismo de *Input* é fundamental. O *Unity* possui a classe *Input* que nos dá acesso a diversas opções relacionadas com este aspeto, nomeadamente controlo de quais as teclas pressionadas, o estado do posicionamento do rato, acesso ao giroscópio de dispositivos móveis, entre muitos outros.

Nesta primeira introdução iremos aceder somente o controlo pelo teclado, em que é importante ficar a conhecer os seguintes aspetos:

- O enumerado `KeyCode` guarda os nomes das diversas teclas (exemplos: `KeyCode.A`, `KeyCode.LeftArrow`, etc.).
- O método `public static bool GetKey(KeyCode key)` permite-nos verificar se uma determinada tecla está a ser premida ou não na *frame* atual.
- Os métodos `public static bool GetKeyUp(KeyCode key)` e `public static bool GetKeyUp(KeyCode key)` permitem saber se na *frame* atual uma determinada tecla foi premida ou largada, respetivamente.

Exemplo – Rotação de um cubo com controlo por teclado

No script anterior a rotação era automática. Podemos alterar o script para que a rotação seja feita só quando uma determinada tecla está premida. Com o seguinte script, o cubo roda para a esquerda ou direita com a utilização dos cursores (*Left Arrow* e *Right Arrow*):

```
public float RotationSpeed = 90.0f;

void Update ()
{

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        transform.Rotate(0, RotationSpeed * Time.deltaTime, 0);
    }

    if (Input.GetKey(KeyCode.RightArrow))
    {
        transform.Rotate(0, -RotationSpeed * Time.deltaTime, 0);
    }

}
```

Trabalho Laboratorial – Exercício

A partir da *Asset Store*, importe um modelo de um veiculo.

Associe um script para controlar com teclado o veiculo com as seguintes características:

- A tecla ESC reposiciona o veiculo na posição inicial do nível.

- As teclas "*Left Arrow*" e "*Right Arrow*" fazem rodar o veículo.

- A tecla "*Up Arrow*" faz o veiculo andar para a frente. Utilize o método *transform.Translate* para o efeito. Como a translação deve ser feita na direção do veículo (ou seja, a referência é o próprio objeto e não o mundo), o método deve ser utilizado de forma semelhante ao apresentado em seguida:
```
transform.Translate(0,0, MovementSpeed * Time.deltaTime, Space.Self);
```

- A tecla "*Down Arrow*" faz o veiculo andar para trás, de forma semelhante à funcionalidade anterior.

Na página da disciplina está disponibilizado um vídeo com o comportamento expectável da aplicação.

Material complementar para consolidação

Bibliografia

Sue Blackman, *Beginning 3D game development with unity 4 (2nd edition)*, Apress, 2013. Capítulo 3 - Páginas 63 a 84

Tutoriais de Apoio – Complementares para consolidação

Game Objects* e *Components
https://unity3d.com/pt/learn/tutorials/topics/interface-essentials/game-objects-and-components?playlist=17090

Delta Time
https://unity3d.com/pt/learn/tutorials/topics/scripting/delta-time?playlist=17117

Transformações
https://unity3d.com/pt/learn/tutorials/topics/scripting/translate-and-rotate?playlist=17117

Documentação Oficial

Utilização de componentes
https://docs.unity3d.com/Manual/UsingComponents.html

Classe Input
https://docs.unity3d.com/ScriptReference/Input.html