Desenvolvimento de Videojogos

Licenciatura em Engenharia Informática – 2020/2021

Guia de Laboratório nº.3

Unity* – Motor de física: utilização de *colliders* e *triggers

---

Introdução e Objetivos

No laboratório anterior introduziu-se o conceito de componente, tendo sido explorado o componente *script*. Este permite programar o comportamento de um determinado *Game Object*. Os exemplos e exercício consistiram na manipulação da posição de um objeto (componente *Transform*).

Neste laboratório será programado um comportamento similar, mas recorrer-se-á ao motor de física incluído no motor *Unity*.

Preparação teórico-prática

Componente Collider

Colliders são componentes que permitem ao *Game Object*, ao qual estão associados, reagir com outros objetos através de colisões.
Os *Colliders* possuem variadas formas e feitios, e são representados na vista da *Scene* através de um contorno verde. Podem ter as seguintes formas primitivas: esfera, cápsula ou caixa.
Por exemplo, quando é criado um cubo, este apresenta por defeito os componentes apresentados na Figura 1. O componente *Box Collider* que é adicionado automaticamente encontra-se associado ao motor de física do *Unity*.

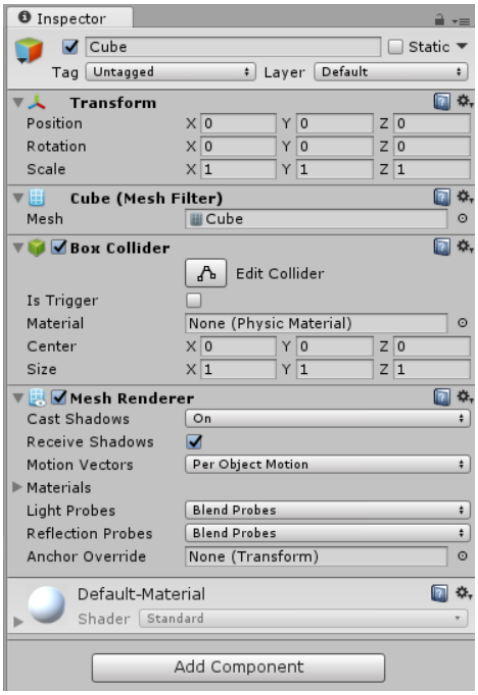

Figura 1 – Componentes base que fazem parte de um cubo

Para termos formas mais complexas, temos duas opções:
1. Combinar várias formas primitivas, aplicando *Colliders* aos diferentes objetos da hierarquia.
2. Usar um *Mesh* (em português, malha) *Collider*. Um *Mesh Collider* tem a forma exata da malha da forma. Esta última opção é a que tem mais precisão, mas possui um inconveniente: o *Mesh Collider* adquire exatamente a mesma forma da malha do modelo. Se o modelo for muito detalhado então o *Mesh Collider* terá também muito detalhe e isso poderá afetar a performance.

Quando uma colisão ocorre, vários eventos poderão ser lançados:

- *OnCollisionEnter*: quando a colisão de inicia.
- *OnCollisionStay*: enquanto os objetos estão em colisão.
- *OnCollisionExit*: quando a colisão termina.

Componente *Rigid Body*

Para ter um "comportamento físico" e a colisão ocorrer, é também necessário acrescentar o componente *Rigid Body*. Este tem como principal finalidade dar existência enquanto corpo físico ao *Game Object*. Para que uma colisão ocorra um dos *Game Objects* tem de possuir um componente *Rigid Body* associado.

Selecionando um cubo criado previamente, podemos acrescentar o referido componente da forma apresentada na Figura 2.

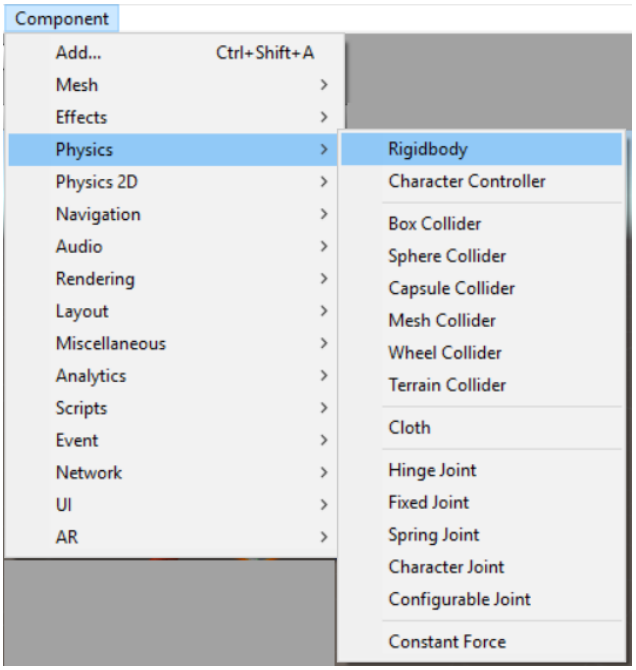

Figura 2 – Criação de um componente Rigid Body

O componente *Rigid Body* fornece massa e gravidade, que podem ser configurados alterando as propriedades observadas na Figura 3.

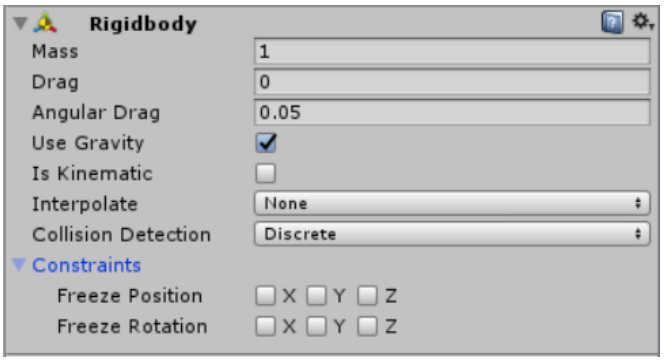

Figura 3 – Componente Rigid Body

Resumo de cada propriedade:

- *Mass* – Massa. Valor em quilogramas que irá influenciar principalmente os ressaltos entre objetos aquando de colisões entre estes.

- *Drag* – Atrito.

- *Angular Drag* – Atrito "angular".

- *Use gravity* – Utilização de gravidade. Permite definir se pretendemos aplicar automaticamente uma força correspondente à gravidade ao objeto.

- *Is Kinematic* – Cinemático. Um objeto que seja definido como cinemático fica fixo na cena e não está sujeito a forças externas. Irá ter influência noutros objetos, mas o seu posicionamento ficará intacto. É útil para partes do cenário que são inalteráveis.

- *Interpolate* – Interpolação.

- *Collision Detection* – Deteção de colisões.

- *Constraints* – Restrições. Permite restringir o comportamento físico (translações e rotações independentemente) num determinado eixo. Por exemplo, num jogo de plataformas, podemos querer limitar o movimento da personagem no plano do ecrã.

Triggers

Um componente *Collider* pode ser um *Trigger*. Quando um componente é um *Trigger*, o Game Object associado não vai participar em colisões, em vez disso, os objetos irão passar através dele e os seguintes eventos poderão ser lançados:

- *OnTriggerEnter*: o objeto entrou no *Trigger*.
- *OnTriggerStay*: o objeto encontra-se dentro do *Trigger*.
- *OnTriggersExit*: o objeto saiu no *Trigger*.

Os *Triggers* são úteis quando pretendemos detetar a interação entre dois objetos sem que exista uma colisão visível. Por exemplo, se pretendermos detetar se um *player* passou por uma porta, colocamos um *Trigger* no local de passagem da porta. Quando o *player* passa pelo *Trigger* (não vai haver uma colisão pois passa através dele), o evento pode ser tratado.
Da mesma forma que nas colisões, um dos objetos tem de ter um componente *Rigid Body*. Por norma, os *Triggers* são objetos em que não são aplicadas forças físicas e que se mantêm estáticos, sendo o objeto que colide com o *Trigger* quem possui o componente *Rigid Body*.

Adicionar forças físicas

Para que um *Game Object*, que possua um *Rigid Body*, se mova segundo uma dada força, podemos usar o método *addForce* aplicado ao *Rigid Body*. Este método recebe como parâmetro a força a aplicar.

No *script* seguinte, são utilizadas as setas do cursor (usando `Input.GetAxis(...))` no teclado para guiar o objeto num plano:

```
private Rigidbody rb;

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

 • • •

void FixedUpdate () {
       float moveHorizontal = Input.GetAxis ("Horizontal");
       float moveVertical = Input.GetAxis ("Vertical");

       Vector3 movement = new Vector3 (moveHorizontal, 0, moveVertical);
       rb.AddForce (movement * speed);
}
```

Exemplo – Colisões e Triggers

Na *Scene* da Figura 4 existe um cubo com o componente *Box Collider* criado por defeito e uma esfera por cima do cubo com um componente *Sphere Collider* que também é criado por defeito. Reproduza esta *Scene* e faça *Play*. O que acontece?

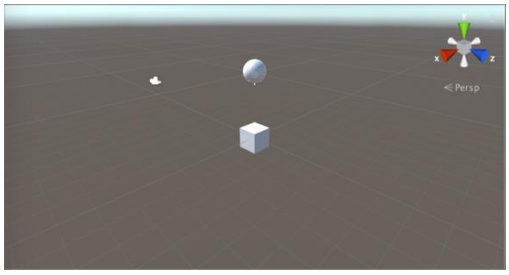

Figura 4 – Scene com um cubo e uma esfera alinhados na vertical.

E se adicionar um componente *Rigid Body* à esfera, como na Figura 5?

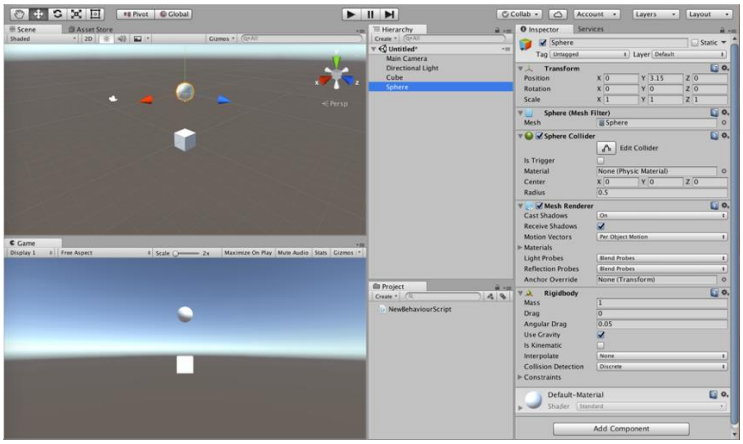

Figura 5 – Scene com um cubo e uma esfera (com componente *Rigid Body*) alinhados na vertical.

Experimente mover a esfera um pouco para a direita de forma a que, quando caia em cima do cubo, lhe acerte na aresta e não fique pousada em cima (ou seja, de forma à colisão ter um fim), como na Figura 6.

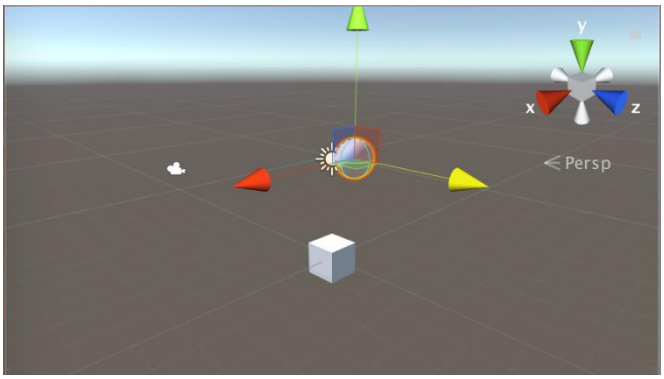

Figura 6 – Scene com um cubo e uma esfera alinhados na vertical, em que a esfera possui um componente Rigid Body.

Adicione o seguinte *script* à esfera:

```
public class SphereController : MonoBehaviour {

	void OnCollisionEnter()
	{
		Debug.Log ("Início da colisão");
	}

	void OnCollisionStay()
	{
		Debug.Log ("Em colisão");
	}

	void OnCollisionExit()
	{
		Debug.Log ("Fim da colisão");
	}
}
```

e observe as mensagens da consola, tal como na Figura 7.

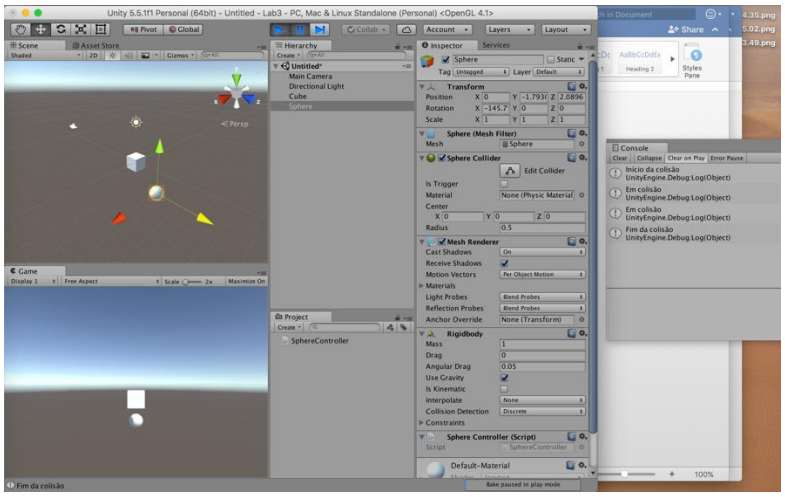

Figura 7 – Mensagens da Consola (Collider).

Agora, defina o cubo como um *Trigger*, tal como na Figura 8.

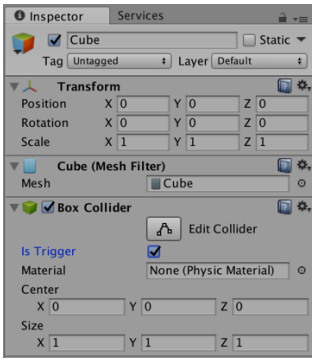

Figura 8 – Cubo que é um Trigger.

Adicione o seguinte *script* ao cubo:

```
public class CubeController : MonoBehaviour {

	void OnTriggerEnter()
	{
		Debug.Log ("Objeto entrou no Trigger");
	}

	void OnTriggerStay()
	{
		Debug.Log ("Objeto dentro do Trigger ");
	}

	void OnTriggerExit()
	{
		Debug.Log ("Objeto saiu do Trigger ");
	}
}
```

e observe as mensagens da consola, tal como na Figura 9.

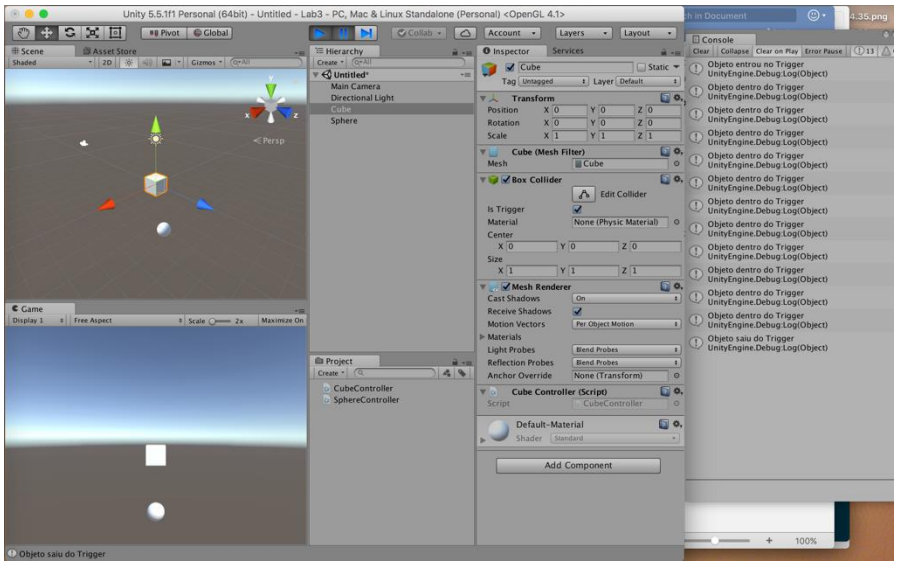

Figura 9 – Mensagens da Consola (Trigger).

Trabalho Laboratorial – Exercício 1 – Baliza

Reproduza a *Scene* da Figura 10.

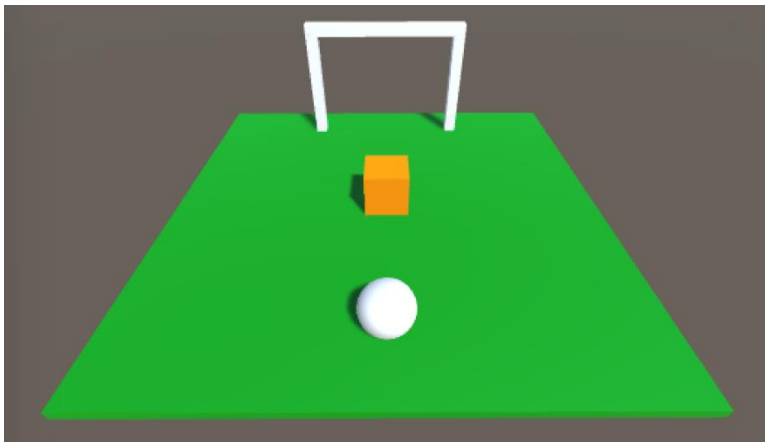

Figura 10 – Scene do Exercício.

Nesta *Scene* existe um *player* que é uma esfera branca, um cubo laranja e uma baliza (*Trigger*). O objetivo consiste em termos uma esfera que empurra o cubo até este passar pela baliza. O cubo ao passar pela baliza desaparece. Para que tal aconteça, é necessário que a baliza possua uma etiqueta (por exemplo "Goal"). Quando o *Trigger* é acionado, verifica-se se a etiqueta é a pretendida, e caso seja, o objeto é tornado inativo. Ver o script seguinte:

```
if (other.CompareTag("Goal")) {
            rb.gameObject.SetActive (false);
}
```

O jogador guia a esfera, através do plano verde, usando as teclas do cursor do teclado, sendo aplicada uma força no seu *Rigid Body*.

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

Trabalho Laboratorial – Exercício 2 – Micro Jogo de Futebol

Reproduza a *Scene* da Figura 11.

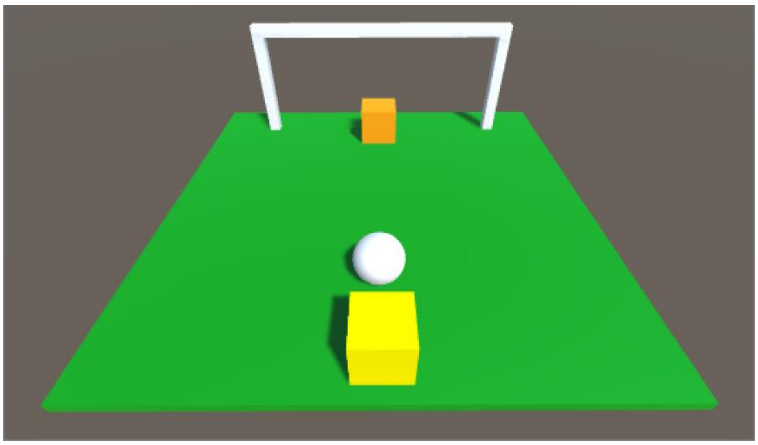

Figura 11 – Scene do Exercício.

Neste exercício, **os papeis dos sólidos geométricos vão inverter-se**, isto é, nesta *Scene* existe um *player* que é um cubo amarelo, uma esfera branca que representa a bola, um cubo laranja que é o guarda-redes e uma baliza (*Trigger*).
O objetivo do exercício consiste em termos um cubo que empurra a esfera até esta passar pela baliza. A esfera, ao passar pela baliza, retorna à sua posição inicial e fica parada.
O guarda-redes encontra-se em permanente movimento, indo de lado a lado da baliza, de forma a que marcar um golo seja mais difícil.
O jogador guia a esfera, através do plano verde, usando as teclas do cursor do teclado.

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

Trabalho Laboratorial – Exercício 3

Altere o Exercício 2 de forma a que tanto o *player* como o guarda-redes possuam a forma geométrica de cápsula.

Material complementar para consolidação

Tutoriais de Apoio – Complementares para consolidação

Colliders
https://unity3d.com/pt/learn/tutorials/topics/physics/colliders?playlist=17120

Colliders as Triggers
https://unity3d.com/pt/learn/tutorials/topics/physics/colliders-triggers?playlist=17120

Rigid Bodies
https://unity3d.com/pt/learn/tutorials/topics/physics/rigidbodies?playlist=17120

Adding Physics Forces
https://unity3d.com/pt/learn/tutorials/topics/physics/adding-physics-forces?playlist=17120

Detecting Collisions (On Collision Enter)
https://unity3d.com/pt/learn/tutorials/topics/physics/detecting-collisions-oncollisionenter?playlist=17120

Documentação Oficial

Fixed Update
https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html
https://unity3d.com/pt/learn/tutorials/topics/scripting/update-and-fixedupdate

Tags
https://docs.unity3d.com/Manual/Tags.html

Collider
https://docs.unity3d.com/ScriptReference/Collider.html

Rigid Body
https://docs.unity3d.com/ScriptReference/Rigidbody.html