**Curso de Pós-Graduação em Cloud Computing e Mobile**
**Disciplina DM 117 – Introdução a Desenvolvimento de Jogos com Unity**
**Professor: Phyllipe Lima**

Aula 2

Ao final desse relatório você terá:

- Usado comando híbridos para detectar mouse/touch
- Feito movimento exclusivo via touch
- Implementado um gesture (swipe)
- Usado o acelerômetro
- Identificar GameObjects que foram tocados via touch

## 1 – Detectar clique via mouse

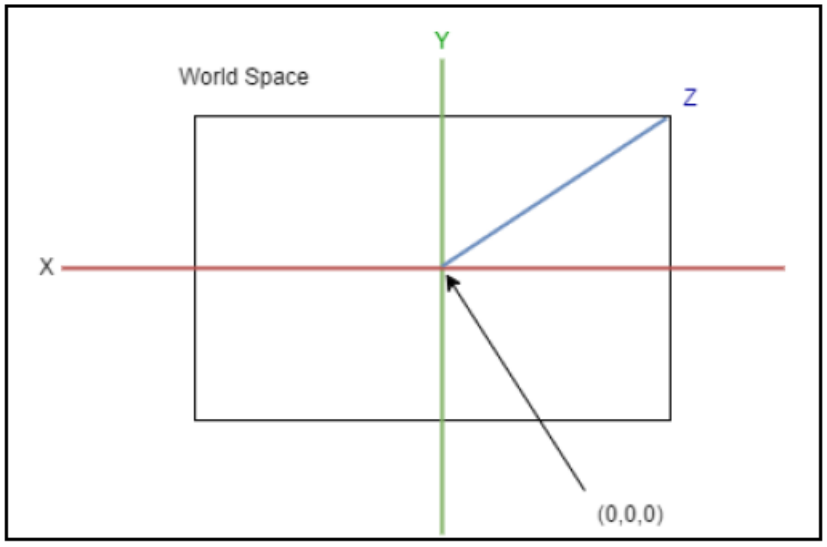
Antes de pensarmos em soluções únicas para dispositivos móveis, vamos usar uma técnica que funciona tanto para cliques quanto para toques na tela. Neste caso se perde um pouco das funcionalidades presentes na interface exclusiva para toque, mas nos permite fazer alguns testes no jogo sem a necessidade de instalar o jogo no dispositivo móvel a cada nova modificação.

Abra o script JogadorComportamento e dentro do método Update() escreva:

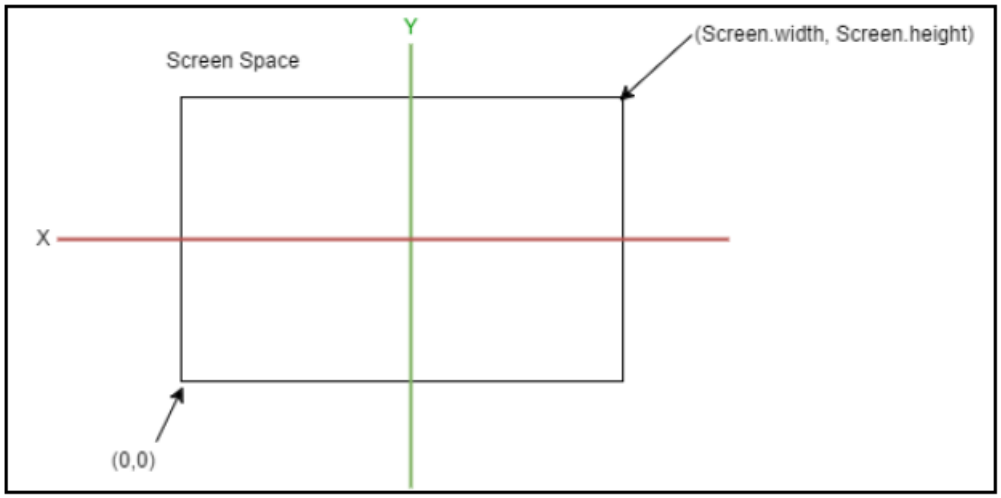
```
27      void Update () {
28          //Verificar para qual lado o jogador deseja esquivar
29          var velocidadeHorizontal
30              = Input.GetAxis("Horizontal") * velocidadeEsquiva;
31
32          //Detectando se houve clique com o botao direito (opcao 0) ou toque na tela
33          if (Input.GetMouseButton(0)) {
34              float direcaoX = 0;
35              var pos = Camera.main.ScreenToViewportPoint(Input.mousePosition);
36
37              if (pos.x < 0.5)
38                  direcaoX = -1;
39              else
40                  direcaoX = 1;
41
42              velocidadeHorizontal
43              = direcaoX * velocidadeEsquiva;
44          }
```

A classe Input oferece uma série de funcionalidades para detectarmos interação do jogador, via mouse, teclado, gamepad (controle), toque na tela, acelerômetro e etc. Vamos começar com o mouse. O método GetMouseButton() detecta se houve clique com o botão direito (opção 0, 1 para botão esquerdo e 2 para o botão do meio) ou se houve touch.

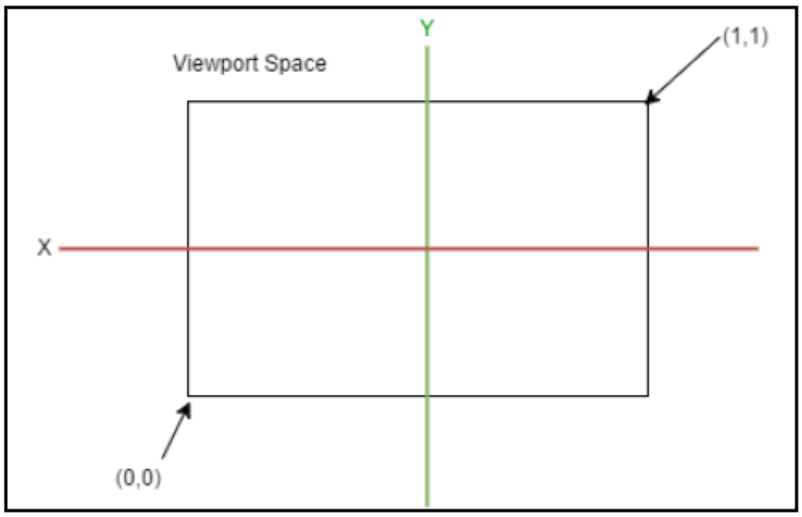
Na linha 35 usamos um método da classe Camera para nos retornar um valor em modo View Port. Para isso precisamos entender como o Unity mostra os GOs na tela. O primeiro modo e o World Space.




Neste modo temos uma coordenada X,Y,Z nos mostrando onde no mundo do jogo o nosso GO se encontra. Na aba Scene o que vemos é o World Space. Porém, quando usamos a classe Input, os valores são retornados no Screen Space.




Este é o espaço da câmera, dado em duas dimensões (X,Y). Onde o canto direito superior nos devolve, em pixels, a largura (x) e altura (y) da tela. Nós poderíamos usar este valor, mas fica bem mais simples convertermos isso para o Viewport Space. Esse espaço nada mais é do que a Screen Space normalizado entre 0 e 1.




Por isso verificamos se o valor de x é menor ou maior que 0,5. Sendo menor sabemos que o mouse/touch está sendo feito no lado esquerdo da tela, e assim fazemos a bola se movimentar para a esquerda. E o valor de x sendo maior fazemos a bola ir para a direita. Salve tudo e faça e aperte Play. Agora faça o deploy em algum dispositivo móvel e teste o touch.

## 2 – Movendo via touch

Agora vamos usar uma solução exclusiva para mobile, através da struct Touch. Volte no script JogadorComp e insira o seguinte código:

```
46          //Detectando exclusivamente via touch
47          if(Input.touchCount > 0) {
48              //Obtendo o primeiro touch na tela dentro do frame
49              Touch toque = Input.touches[0];
50              float direcaoX = 0;
51
52              //Convertendo para Viewport space (entre 0 e 1)
53              var pos = Camera.main.ScreenToViewportPoint(toque.position);
54
55              if (pos.x < 0.5)
56                  direcaoX = -1;
57              else
58                  direcaoX = 1;
59
60              velocidadeHorizontal
61              = direcaoX * velocidadeEsquiva;
62
63          }
```

Na linha 47 detectamos se houve um touch. Depois na linha 49 pegamos o primeiro touch ocorrido naquele frame e em seguida trabalhamos de forma semelhante como fizemos para ler o clique do mouse. Vamos criar um método ao invés de duplicar código.

```
48      /// <summary>
49      /// Metodo para calcular para onde o jogador se deslocara na horizontal
50      /// </summary>
51      /// <param name="screenSpaceCoord">As coordenadas no Screen Space</param>
52      /// <returns></returns>
53      private float CalculaMovimento(Vector2 screenSpaceCoord)
54      {
55          var pos = Camera.main.ScreenToViewportPoint(screenSpaceCoord);
56
57          float direcaoX = 0;
58
59          if (pos.x < 0.5)
60              direcaoX = -1;
61          else
62              direcaoX = 1;
63          return direcaoX * velocidadeEsquiva;
64      }
65  }
66
```

Crie um método chamado CalculaMovimento(). Ele recebe a posição em Screen Space e retorna um *float* com a magnitude e direção que a bola deve se deslocar.

Agora volte no método Update e atualize a forma como iremos detectar clique o mouse ou o touch.

```
27      void Update () {
28          //Verificar para qual lado o jogador deseja esquivar
29          var velocidadeHorizontal
30              = Input.GetAxis("Horizontal") * velocidadeEsquiva;
31
32          //Detectando se houve clique com o botao direito (opcao 0) ou toque na tela
33          if (Input.GetMouseButton(0)) {
34              velocidadeHorizontal = CalculaMovimento(Input.mousePosition);
35          }
36
37          //Detectando exclusivamente via touch
38          if (Input.touchCount > 0) {
39              //Obtendo o primeiro touch na tela dentro do frame
40              Touch toque = Input.touches[0];
41              velocidadeHorizontal = CalculaMovimento(toque.position);
42          }
43
44          //Aplicar uma força para que a bola se desloque
45          rb.AddForce(velocidadeHorizontal, 0, velocidadeRolamento);
46      }
```

Ainda podemos aprimorar este código e diferenciar se estamos em um dispositivo móvel ou no PC/Standalone adicionando algumas diretivas do Unity. Observe o código abaixo.

```
32  #if UNITY_STANDALONE || UNITY_EDITOR || UNITY_WEBPLAYER
33          //Detectando se houve clique com o botao direito (opcao 0) ou toque na tela
34          if (Input.GetMouseButton(0)) {
35              velocidadeHorizontal = CalculaMovimento(Input.mousePosition);
36          }
37  //Verifica se estamos no iOS ou Android. Se não estivermos o codigo estara cinza
38  #elif UNITY_IOS || UNITY_ANDROID
39
40          //Detectando exclusivamente via touch
41          if (Input.touchCount > 0) {
42              //Obtendo o primeiro touch na tela dentro do frame
43              Touch toque = Input.touches[0];
44              velocidadeHorizontal = CalculaMovimento(toque.position);
45          }
46  #endif
```

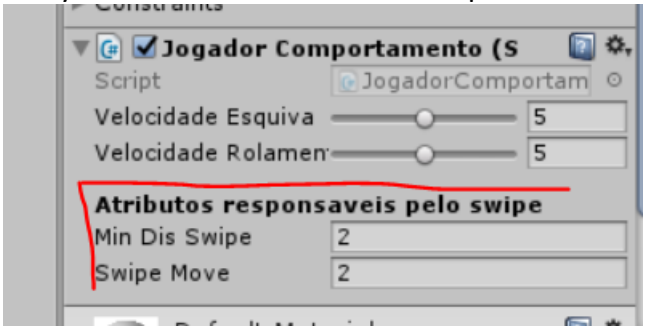
As diretivas usadas auxiliam na criação de códigos multiplataformas. Mude a Build e você verá que agora outra parte do código que passará a compilar.

## 3 – Usando gestures

Vamos usar o *swipe* para fazermos a bola se deslocar abruptamente, quase um tele transporte, na horizontal. Para isso vamos definir primeiro alguns atributos da classe JogadorComportamento.

```
20      [Header("Atributos responsaveis pelo swipe")]
21      [Tooltip("Determina qual a distancia que o dedo do jogador " +
22          "deve deslocar pela tela para ser " +
23          "considerado um swipe")]
24      public float minDisSwipe = 2.0f;
25
26      [Tooltip("Distancia que a bola ira percorrer atraves do swipe")]
27      public float swipeMove = 2.0f;
28
29      /// <summary>
30      /// Ponto inicial onde o swipe ocorreu
31      /// </summary>
32      private Vector2 touchInicio;
```

Na linha 20 usamos o *attribute* [Header] que irá nos ajudar a organizar o Editor do Unity. Ficará fácil identificarmos lá quais as variáveis que lidam com swipe.



Seguindo nas linhas 24 e 27 temos as variáveis que controlam respectivamente qual deverá ser o deslocamento mínimo do dedo pela tela para ser considerado swipe e qual a distância que a bola irá se deslocar pelo swipe. Por fim na linha 32 nós salvamos qual foi a posição inicial do touch.

Agora vá até o método Update() e faças as modificações abaixo

```
76      private void SwipeTeleport(Touch toque)
77      {
78          //Verifica se esse eh o ponto onde o swipe comecou
79          if (toque.phase == TouchPhase.Began)
80              toqueInicio = toque.position;
81
82          //Verifica se o swipe acabou
83          else if(toque.phase == TouchPhase.Ended) {
84
85              Vector2 toqueFim = toque.position;
86              Vector3 direcaoMov;
87
88              //Faz a diferenca entre o ponto final e inicial do swipe
89              float dif = toqueFim.x - toqueInicio.x;
91              //Verifica se o swipe percorreu uma distancia suficiente para
92              //ser reconhecido como swipe
93              if (dif >= Math.Abs(minDisSwipe))
94              {
95                  //Determina a direcao do swipe
96                  if (dif < 0)
97                      direcaoMov = Vector3.left;
98                  else
99                      direcaoMov = Vector3.right;
100             }
101             else
102                 return;
103             //Raycast eh outra forma de detectar colisao
104             RaycastHit hit;
105
106             //Vamos verificar se o swipe nao vai causar colisao
107             if (!rb.SweepTest(direcaoMov, out hit,swipeMove))
108                 rb.MovePosition(rb.position + (direcaoMov * swipeMove));
109         }
110     }
```

Vamos analisar este método.
Primeiro verificamos se o swipe começou (79) e ou se terminou (83). Em ambos os casos nós salvamos as coordenadas e na linha 85 verificamos a diferença em x. Se a diferença for maior que nosso valor mínimo definido continuamos o processo e verificamos se o swipe for para a direita ou esquerda e salvamos a direção na variável direcaoMov.

OBS: Na linha 93 o if está invertido. Deveria ser if(Mathf.Abs(dif) >= minDisSwipe)

Porém, devemos verificar se este swipe irá causar alguma colisão da bola com o obstáculo. E assim usamos uma técnica conhecida como *Raycast* para detectar colisão. Está técnica consiste em projetar um vetor invisível na direção desejada e verificar se

este vetor colide com algum objeto na Scene. Está técnica é largamente utilizada em jogos de tiro em primeira pessoa.
Na linha 107 usamos o método SweepTest() para verificar se ocorreria uma colisão entre a bola, representado pelo seu Rigidbody (variável rb) e um qualquer outro GO com um Collider genérico. Esse método recebe três parâmetros: A direção do movimento, um objeto para salvar informações do potencial objeto em rota de colisão, e qual o tamanho do vetor invisível quer fará o *raycast* (ou seja, a distância que seria percorrida pela bola).

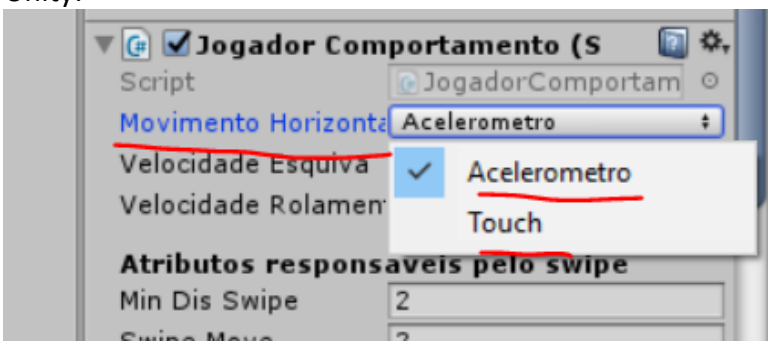
Caso não houver nenhum objeto em rota de colisão, o método retorna falso e movimentamos a bola.

## 4 – Usando o acelerômetro

Outra forma de fazer movimentação no mobile é pelo acelerômetro. Primeiro vamos criar um enum para decidirmos se iremos movimentar pelo touch ou acelerômetro. Dentro da classe JogadorComportamento, antes dos atributos crie o enum abaixo:

```
7   public class JogadorComportamento : MonoBehaviour {
8
9       public enum TipoMovimentoHorizontal
10      {
11          Acelerometro,
12          Touch
13      }
14
15      public TipoMovimentoHorizontal movimentoHorizontal =
16          TipoMovimentoHorizontal.Acelerometro;
17
```

Na linha 15 crie uma variável para segurar esse valor. Observe como fica o Editor do Unity:




Com essa técnica facilmente algum designer poderá fazer testes no jogo.

Agora volte para o método Update() para tratarmos a opção de movimentação via acelerômetro.

```
63  //#elif UNITY_IOS || UNITY_ANDROID
64
65          if (movimentoHorizontal == TipoMovimentoHorizontal.Acelerometro)
66              // Move a bola baseada na direcao do acelerometro
67              velocidadeHorizontal = Input.acceleration.x * velocidadeRolamento;
68          else {//Movimento por touch
69              //Detectando exclusivamente via touch
70              if (Input.touchCount > 0)
71              {
72                  //Obtendo o primeiro touch na tela dentro do frame
73                  Touch toque = Input.touches[0];
74
75                  //Usando por touch
76                  velocidadeHorizontal = CalculaMovimento(toque.position);
77                  //Usando movimento por swipe
78                  //Lembrando que podemos ter o jogo pelo swipe e touch ao mesmo tempo
79                  SwipeTeleport(toque);
80              }
81          }
```

Na linha 65 verificamos se o movimento será pelo acelerômetro. Se for aplicar na velocidadeHorizontal a posição x do acelerômetro. Caso contrário damos sequência ao movimento via touch.

Se você segurar o celular na vertical no modo retrato (com o botão home embaixo) o eixo x é positivo para a sua direita. O eixo y é positivo para cima, e o eixo z é positivo em direção a você.

## 5 – Detectando toque em GO

Para finalizar vamos verificar se algum objeto foi tocado pelo touch. Na classe JogadorComportamento crie o método abaixo:

```
146     /// <summary>
147     /// Metodo para identificar se objetos foram tocados
148     /// </summary>
149     /// <param name="toque">O toque ocorrido nesse frame</param>
150     private static void TocarObjetos(Touch toque) {
151
152         //Convertemos a posicao do toque (Screen Space) para um Ray
153         Ray toqueRay = Camera.main.ScreenPointToRay(toque.position);
154
155         //Objeto que ira salvar informacoes de um possivel objeto que foi tocado
156         RaycastHit hit;
157
158         if (Physics.Raycast(toqueRay, out hit))
159             hit.transform.SendMessage("ObjetoTocado",
160                 SendMessageOptions.DontRequireReceiver);
161     }
```

Na linha 153 convertermos a posição do toque para um *Ray* e na linha 158 verificamos se esse *Ray* colide com algum objeto através do método Raycast da classe Physics. O RaycastHit é o objeto que estava na rota de colisão.

Se algo tiver sido tocado, ou seja, detectado pelo Raycast, nós chamaremos o método ObjetoTocado(), se existir, no objeto que foi tocado. Fazemos isso através do SendMessage() que é uma forma de GOs enviarem mensagens entre si. O

SendMessage() recebe um parâmetro que é justamente o nome do método que queremos invocar no GO. Voltemos para o método Update().

```
77                  //Usando movimento por swipe
78                  //Lembrando que podemos ter o jogo pelo swipe e touch ao mesmo tempo
79                  SwipeTeleport(toque);
80
81                  //Verifica se esse toque atingiu algum objeto
82                  TocarObjetos(toque);
83              }
84          }
85
```

Toda vez que tocarmos na tela, vamos verificar se houve colisão com algum obstáculo. O objetivo é destruirmos esses obstáculos com o toque. Para isso vamos agora no script ObstaculoComp e criarmos o método ObjetoTocado().

```
31      /// <summary>
32      /// Metodo invocado atraves do SendMessage(), para detectar que este objeto foi tocado
33      /// </summary>
34      public void ObjetoTocado() {
35          if (explosao != null) {
36              //Cria o efeito da explosao
37              var particulas = Instantiate(explosao, transform.position,
38                      Quaternion.identity);
39              //Destroi as particulas
40              Destroy(particulas, 1.0f);
41          }
42          //Destroi este obstaculo
43          Destroy(this.gameObject);
44      }
45
```

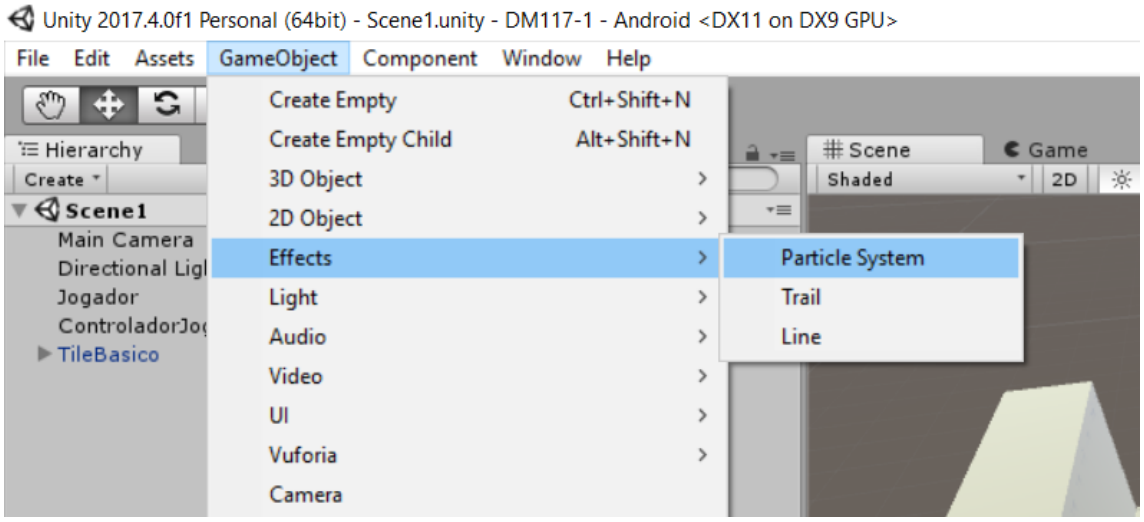
Neste método simplesmente criamos o efeito da explosão e depois destruímos o efeito e o próprio obstáculo. Não se esqueça de declarar no começo da classe uma referência para a explosão.

```
6   public class ObstaculoComp : MonoBehaviour {
7
8       [Tooltip("Particle System da explosao")]
9       public GameObject explosao;
10
11      [Tooltip("Quanto tempo antes de reiniciar o jogo")]
12      public float tempoEspera = 2.0f;
13
```

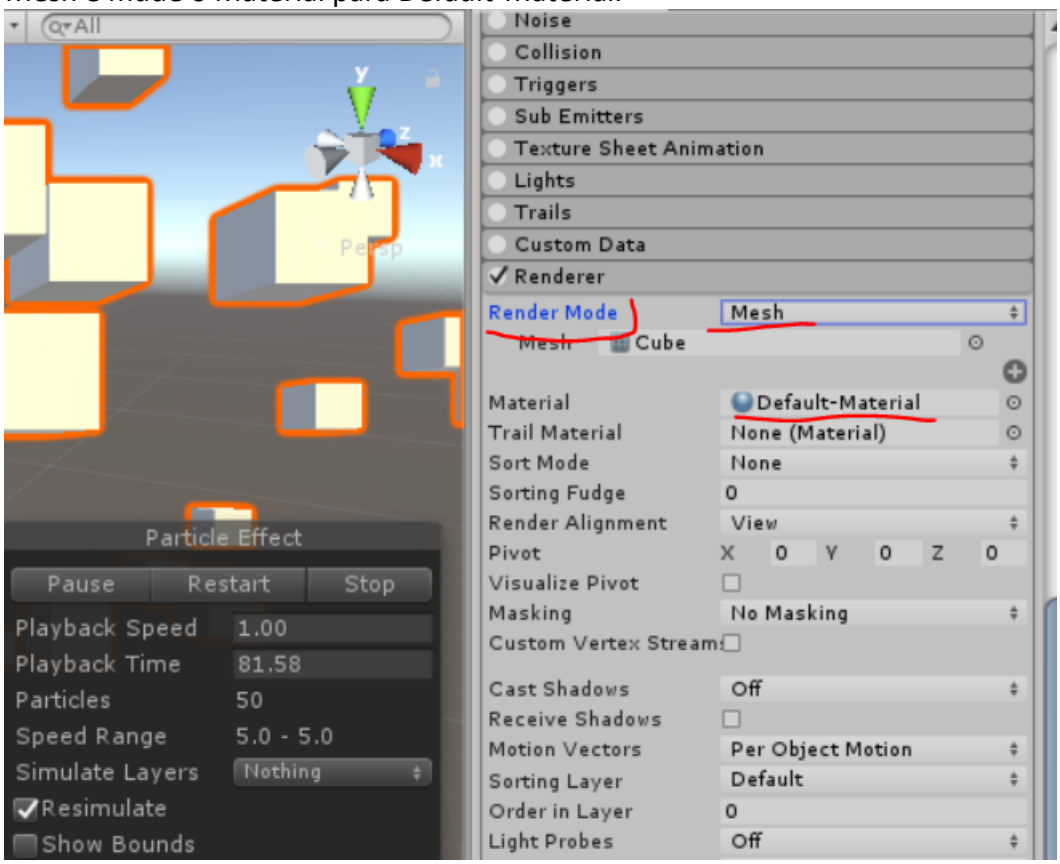
## Criar a explosão

Vamos agora ter um contato inicial com o *Particle System*(PS) do Unity que nos permite criar efeitos como fumaça, explosões e fogo. Mais para frente em outra aula iremos trabalhar novamente com isso. Faremos aqui um uso mais simples.

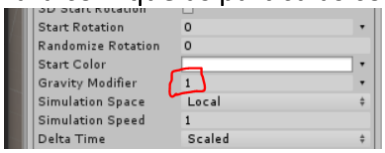
Para começar vá em **GameObject -> Effects -> Particle System**



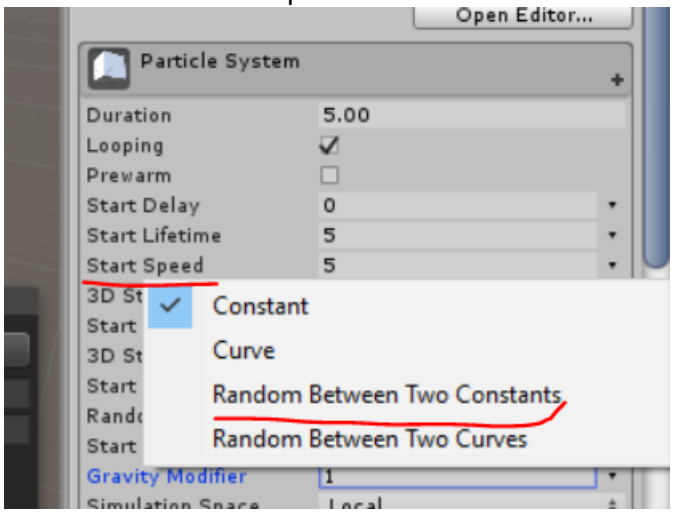

Observe na aba Hierarchy que o PS também é um GO e possui o componente básico Transform. Renomeio o GO para Explosao e na aba Inspector, dentro do componente Particle System procure a opção Renderer bem no final. Mude o Render Mode para Mesh e mude o Material para Default-Material.



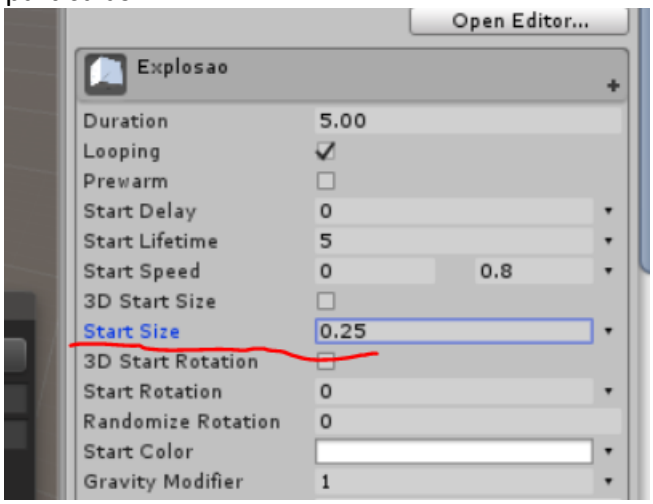

Veja que o PS fica constantemente gerando partículas, criando a animação que você vê na Scene. Vá na parte inicial do componente e altere o Gravity Modifier para 1. Isto fará com que as partículas comecem a cair no sentido negativo do eixo Y.



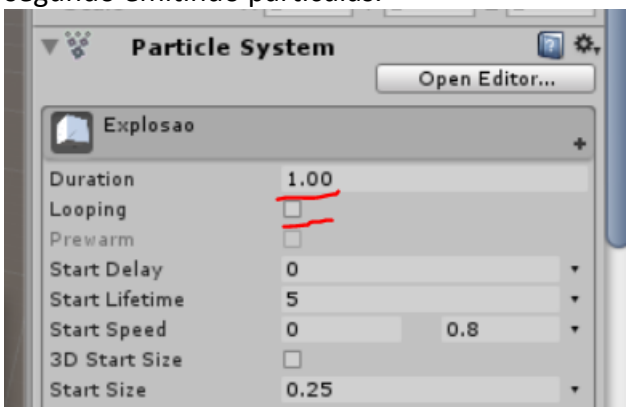
No campo Start Speed (um pouco acima do Gravity Modifier) mude o modo para Random Between Two Constants, e coloque os valores 0 e 0.8. Isso fará com que a velocidade inicial das partículas varie entre estes valores.



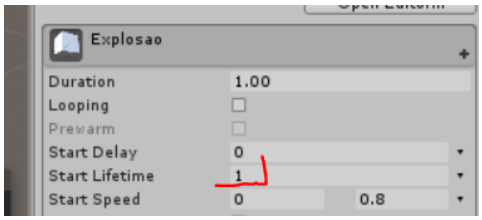

Mude o Start Size para algo próximo de 0.25. Isso altera o tamanho inicial das partículas.



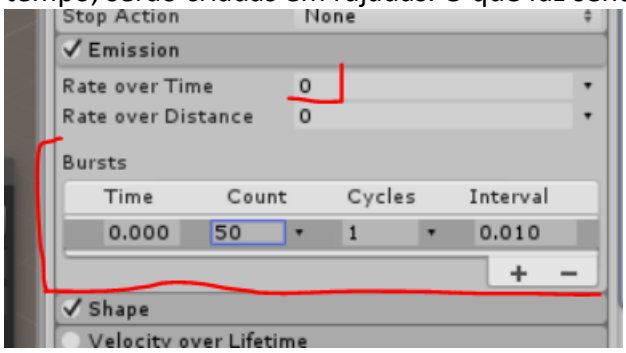

Mude o Duration para 1 e desmarque a opção Looping. Isso faz com que o PS fique 1 segundo emitindo partículas.



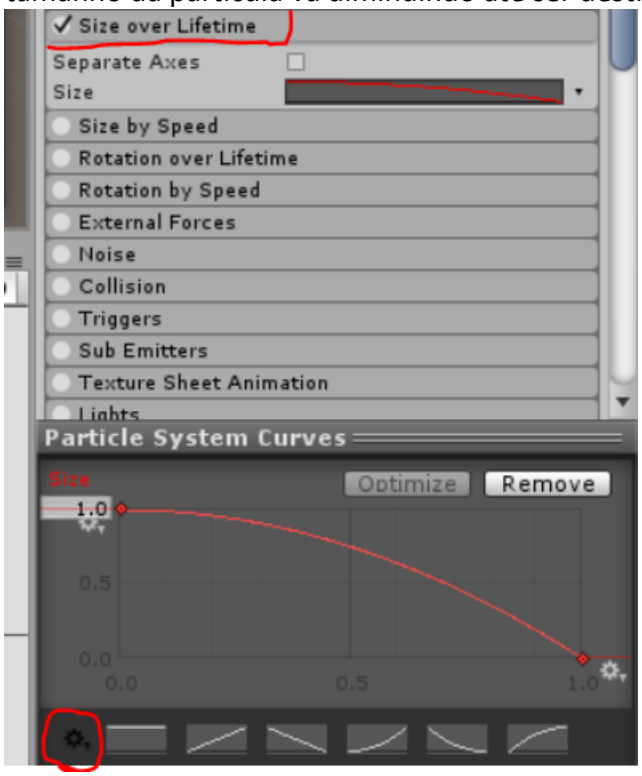

Mude o Start Lifetime para 1. Isso fará com que as partículas sejam destruídas 1 segundo após terem sido criadas.



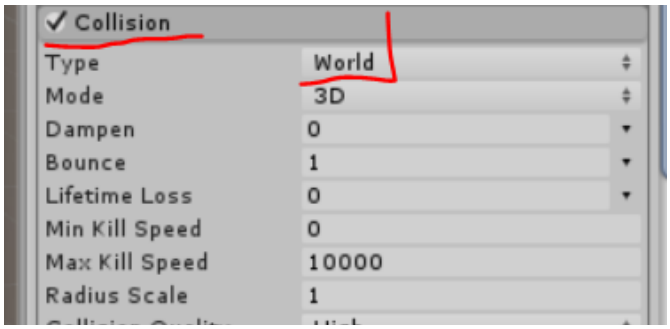

Na aba Emission altere o Rate Over Time para 0 e adicione um Burst com count valendo 50. Ao invés de partículas serem criadas seguindo uma taxa em relação ao tempo, serão criadas em rajadas. O que faz sentido, pois queremos criar uma explosão.



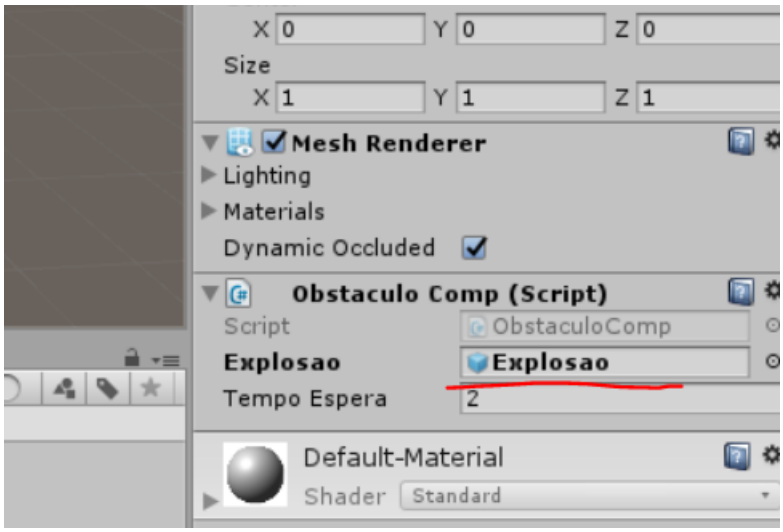

Selecione o Size Over Lifetime clicando no checkbox. E defina uma curva parecida com a da foto. Existe um botão com alguns presets. Essa propriedade faz com que o tamanho da partícula vá diminuindo até ser destruída.




E para terminar a configuração desse efeito de explosão ative a Collision, e mude o Type para World. Isso fará com que as partículas colidam com o chão.




Faça essa Explosão se tornar um prefab (arrastando da Hierarchy para dentro da pasta Prefabs). Remova agora a Explosão que está na Scene. No Script ObstaculoComp adicione essa Explosão (já como prefab) para o campo Explosão (para isso coloque um prefab Obstáculo na Scene, depois o delete.




Divirta-se!