

Pós-Graduação em Desenvolvimento de Aplicações para Dispositivos Móveis e Cloud Computing

Disciplina DM 117 – Desenvolvimento de Jogos com Unity
Professor: Phyllipe Lima

Roteiro 5 – Jogo 2D no estilo do clássico *Arkanoid*

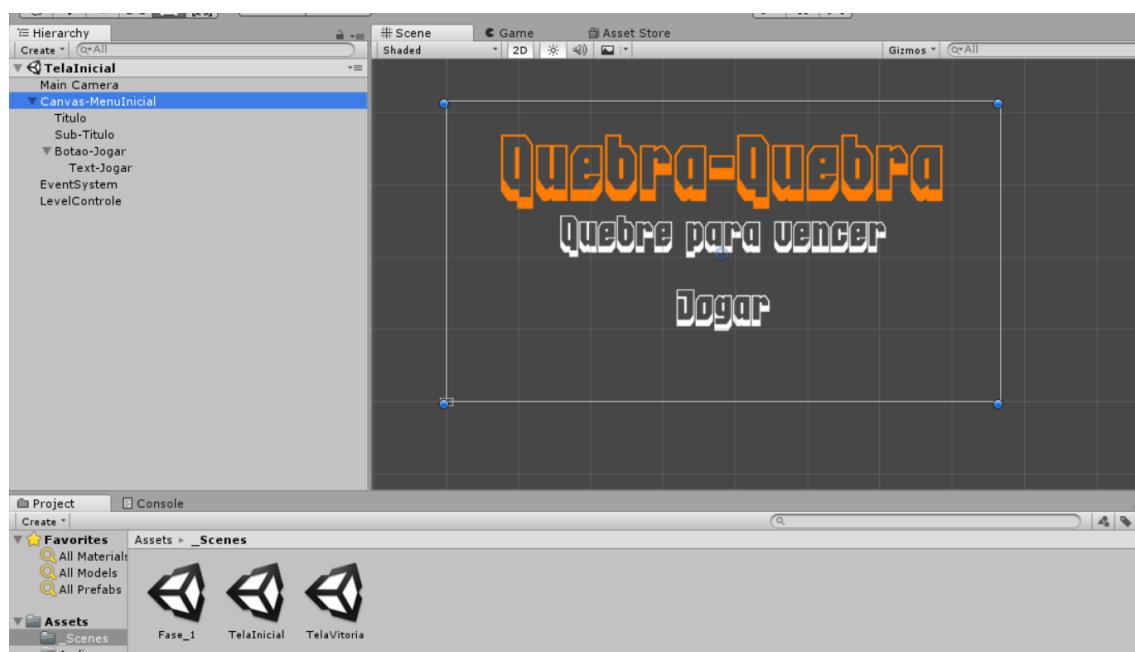
1 – Importando os *sprites/áudio/fonts*

- Faça o download no *link* abaixo:
https://1drv.ms/u/s!Av0id4Knsr2vk7B66_0ex5iFAviZqA
- Arraste o conteúdo para a pasta adequada

2 – Criando a Tela Inicial/Jogo/Vitoria

Criando a Telainicial

- Crie um botão, um Título e um Subtítulo
- Se quiser usar a fonte fornecida. Simplesmente arraste da pasta **Fonts** para o local adequado do componente **Text**. Esse procedimento deve ser repetido em qualquer lugar novo que você queira alterar a fonte.





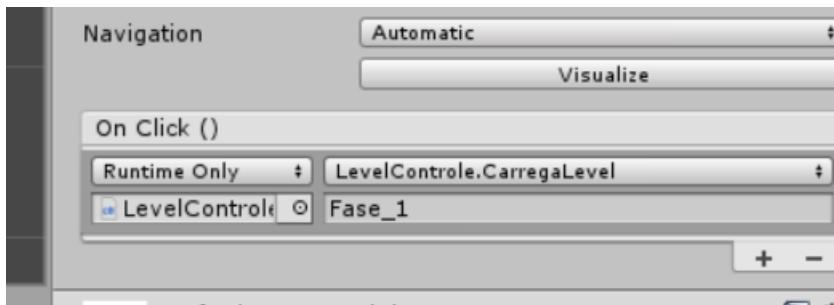
- Salve essa **scene** como Telalnicial
- Crie um GO vazio chamado LevelControle
- Crie um script chamado LevelControle e associe ao GO anterior
- Adicione um método que carregue **Scenes** do Unity

```

6   public class LevelControle : MonoBehaviour {
7
8     public void CarregaLevel(string levelNome) {
9       SceneManager.LoadScene(levelNome);
10    }
11
12  }
13

```

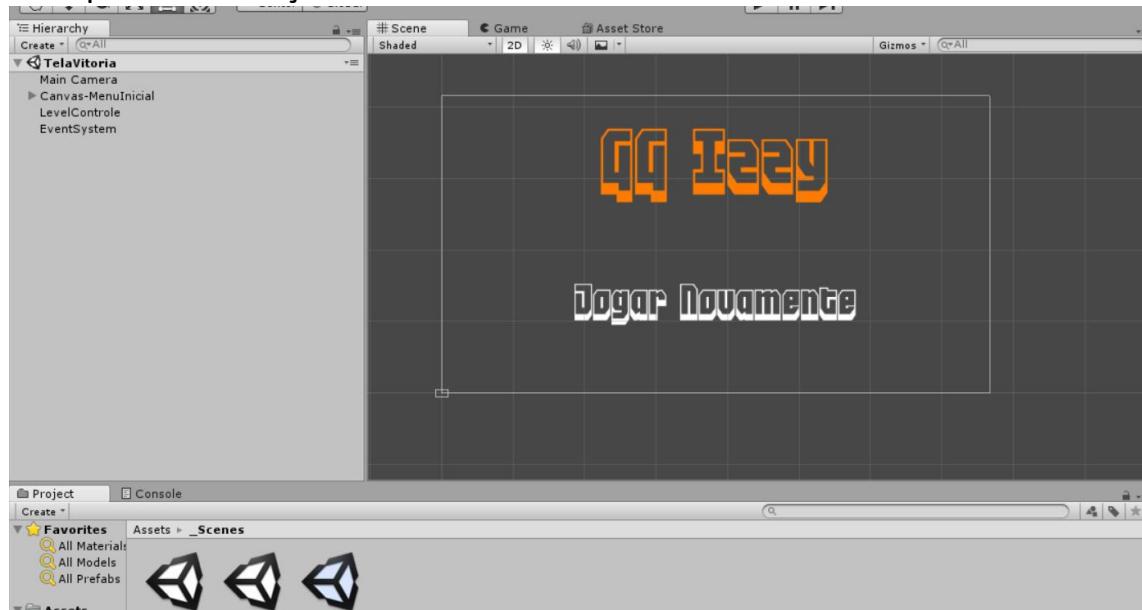
- Faça o botão “Jogar” carregar uma **scene** chamada Fase_1
- Transforme esse GO LevelControle e um prefab



Criando a TelaVitoria

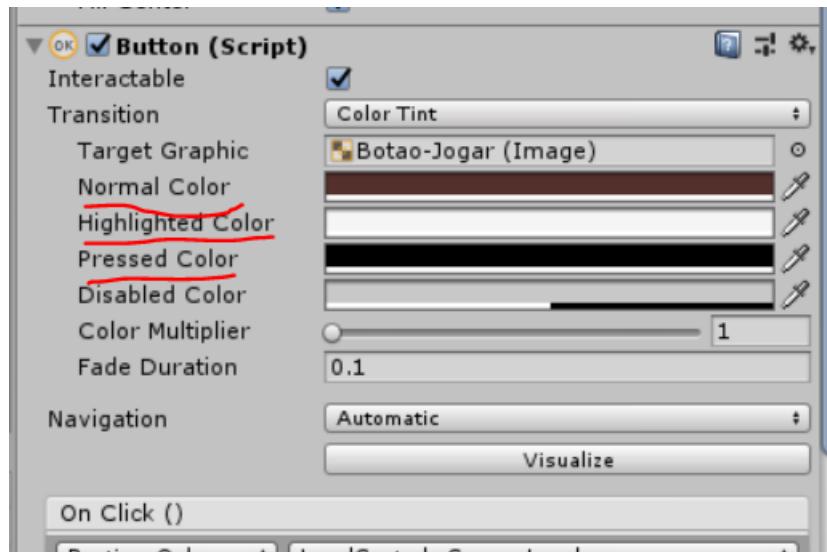
- Seguindo o procedimento anterior crie a tela mostrada abaixo
- Faça o botão “Jogar Novamente” retornar a Telalnicial

- Lembre-se que essa **scene** também precisa do GO LevelControle para manipular a transição das **scenes**.



Você tem a opção de mudar a cor do botão quando o mouse “passa” por cima (**hover**) e quando efetivamente se clica no botão.

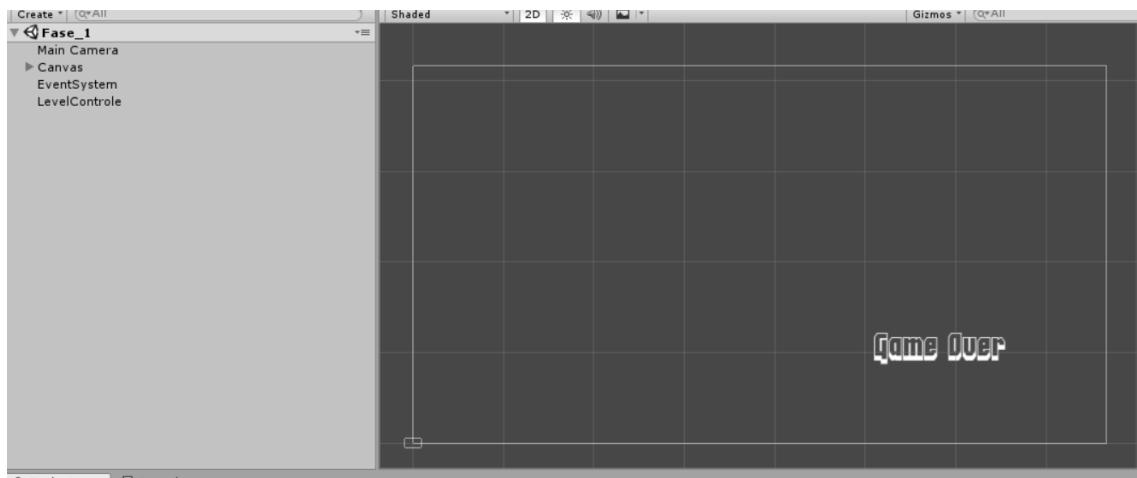
- Para o **hover** altere a cor **Highlighted Color**
- Para o “pressionado” altere a cor do **Pressed Color**
- É necessário entrar no modo **play** para ver o resultado.



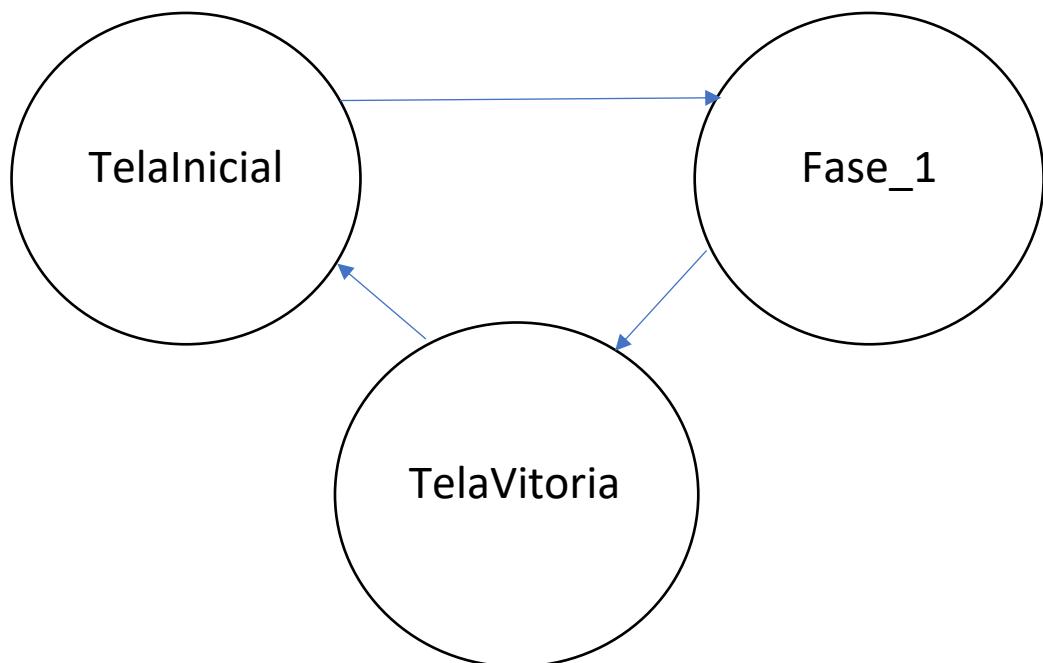
Criando a Fase_1

A Fase_1 é apenas um **placeholder** por enquanto. Queremos testar a navegação das telas.

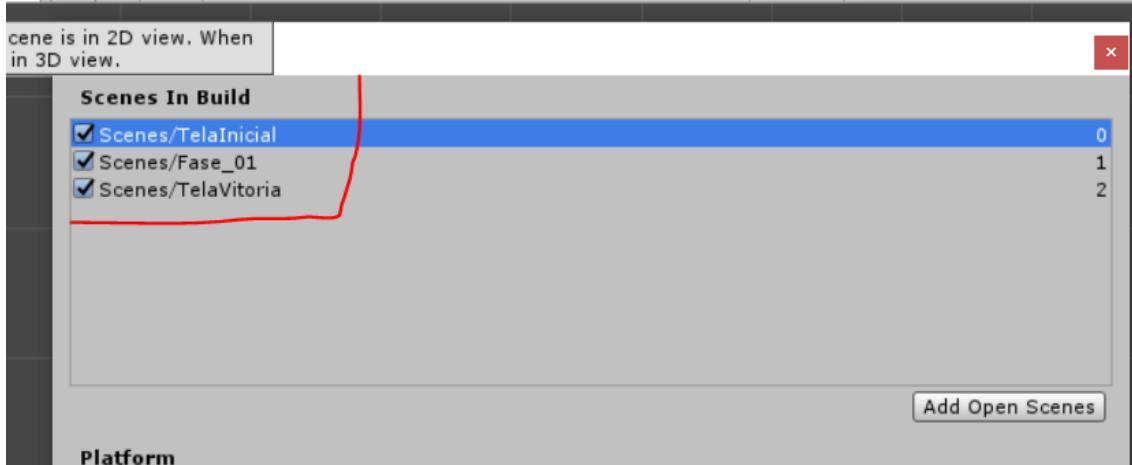
- Crie uma **scene** chamada Fase_1
- Adicione um LevelControle e um botão com o texto **Game Over**
- Faça esse botão carregar a **scene** TelaVitoria ao ser clicado.



- Garanta que o fluxo de troca de telas está funcionando.



- Se houver problema para carregar alguma **scene** verifique se elas estão no *Build Settings*.

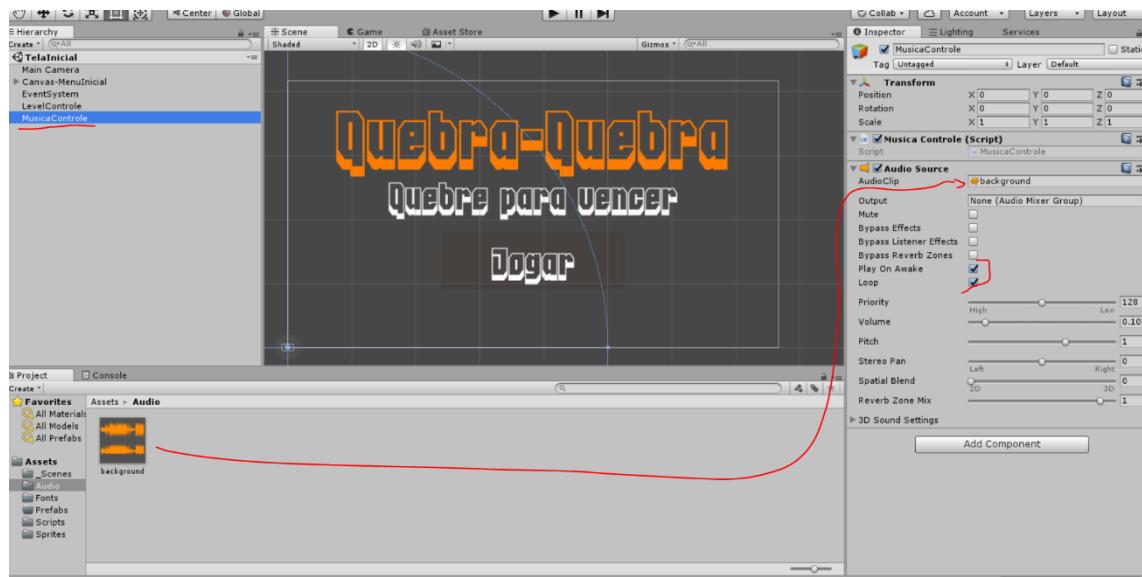


Adicionando música de background

O Unity tem um componente chamado **Audio Listener**. Por padrão, ele vem associado ao **Main Camera** do jogo. Este componente capta todo o áudio sendo emitido dentro da **Scene**. E como a câmera, normalmente, acompanha o jogo, faz sentido que este componente fique nela. Outro possível candidato para ter esse componente é o **Player**, ou o jogador. Só é possível ter um **Audio Listener** por **Scene**. Então devemos experimentar qual se adequa melhor ao jogo em questão. Se o componente, por algum motivo, não estiver na **Main Camera**, basta adicioná-lo como qualquer outro componente em **Add Component**.

O componente responsável por gerar o áudio no Unity é o **Audio Source**. Mas antes disso vamos escolher alguns sons para o nosso jogo. O link <https://freesound.org> possui uma boa variedade de áudio que pode ser utilizado dentro do seu jogo. No arquivo fornecido, já existe uma pasta Audio com os sons que irei usar neste Roteiro. Fique à vontade para escolher outros que lhe agrade mais. Mas tome cuidado com o tamanho, pois se estiver interessado em jogo mobile isso se torna um problema. E opte por arquivos com extensão .ogg ou .mp3

- Volte na **scene** Telainicial
- Crie um GO vazio chamado MusicaControle.
- Adicione um **Audio Source** nesse GO
- Como **AudioClip** forneça a música background
- Crie um script chamado MusicaControle e adicione a este GO
- Marque as opções **Play on Awake** e **Loop**



- Abra o script MusicaControle

```

5  public class MusicaControle : MonoBehaviour {
6
7      // Use this for initialization
8      void Start () {
9          GameObject.DontDestroyOnLoad(gameObject);
10     }
11 }
12

```

O método **DontDestroyOnLoad()** faz com que o GO em questão não seja destruído quando o jogo carregar um nova **scene**. Assim, a música de background irá persistir. Porém há um problema. Caso vá da TelaVitoria para a Telainicial, outra música de fundo será criada e sobreposta a essa, misturando os sons. Vamos resolver isso com o padrão **Singleton**.

```

5  public class MusicaControle : MonoBehaviour {
6
7      public static MusicaControle musicaControle = null;
8
9      private void Awake() {
10         if (musicaControle != null) {
11             Destroy(gameObject);
12         } else {
13             musicaControle = this;
14             GameObject.DontDestroyOnLoad(gameObject);
15         }
16     }
17

```

Ao ser executado pela primeira vez, a variável ***musicaControle*** vale ***null*** e, portanto, a primeira instância desse GO a existir no jogo será atribuído a ela (padrão ***Singleton***, nenhum segredo). Toda vez que algum outro GO com o ***script*** *MusicaControle* é executado verificamos se já existe alguma instância, se existir o GO será destruído.

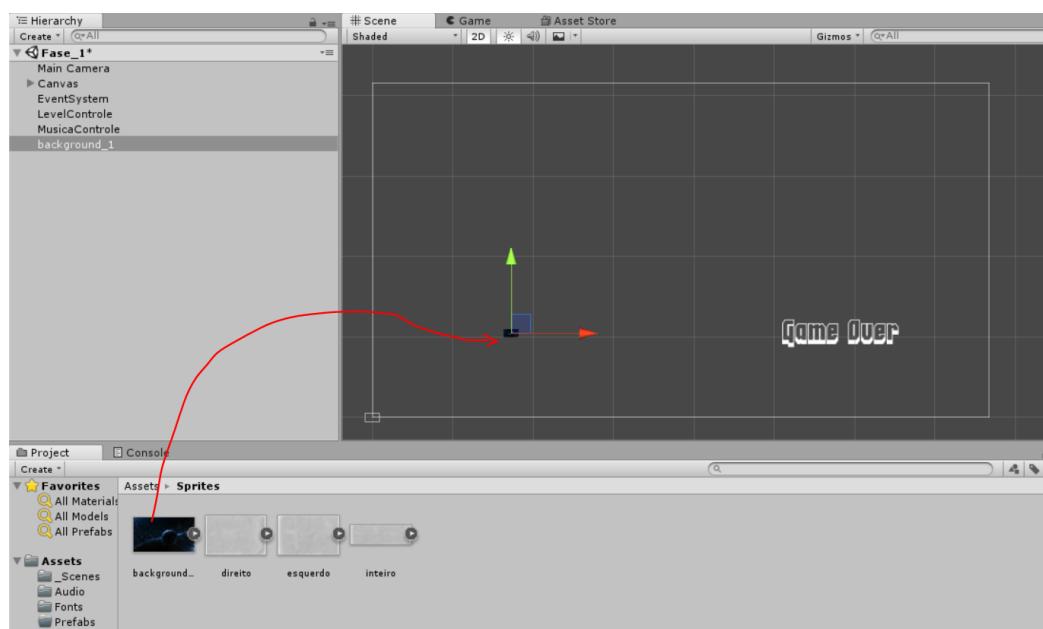
Observe que estamos usando o método ***Awake()*** ao invés do ***Start()***, pois o ***Awake*** é invocado uma única vez também, e ele executa antes do ***Start()***, antes mesmo da ***scene*** começar a executar. Assim garantimos que qualquer cópia do *MusicaControle* será destruída antes que tenha chance de ser executada.

3 – Configurando o World Unit

Precisamos entender que o Unity desloca os GO no mundo do jogo usando as unidades do mundo, ou seja, ***world units***. Mas as imagens que usamos para importar no Unity estão em ***pixels***. Assim, precisamos fazer uma conversão.

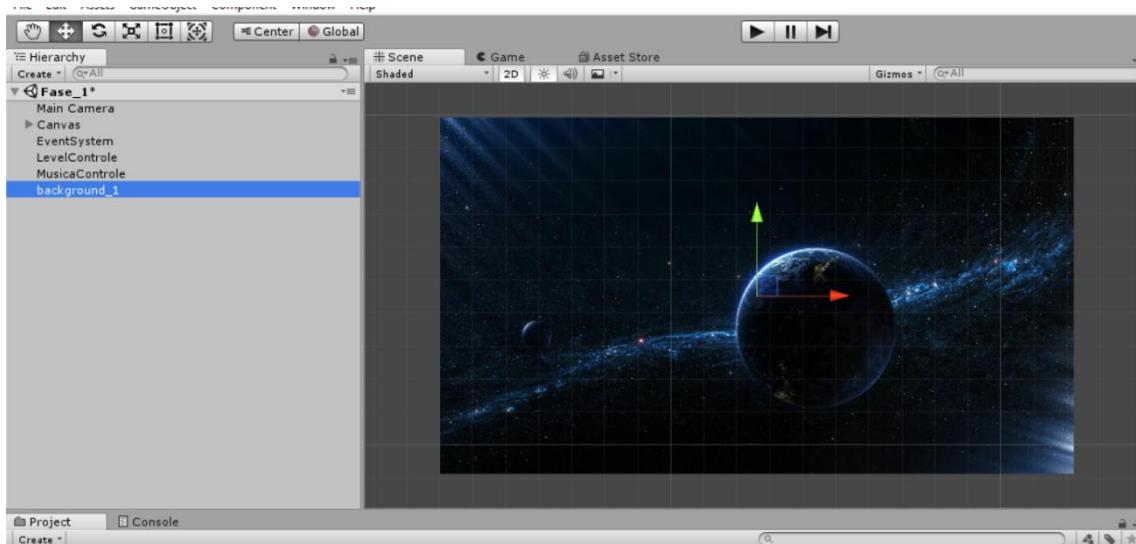
Para esse jogo, estilo ***Arkanoid***, eu irei usar uma resolução de 1920x1080 o que nos dá uma relação de aspecto de 16:9. O jogo está sendo planejado para ter 16 blocos na horizontal. Ou seja, cada bloco deverá ocupar 1 ***world unit***. Assim, o ***background*** deverá ocupar 16 ***world units***.

- Vá até a ***scene*** Fase_01
- Coloque na ***scene*** o arquivo background_1



Talvez você esteja um pouco confuso, pois o background está bem pequeno. O que acontece é que temos duas coisas nessa tela. A primeira será o jogo, onde está o **background** que acabamos de adicionar. A outra é a UI que está em um **Canvas** no modo **Screen Space Overlay**. Isso significa que esta tela é sobreposta (overlay) ao jogo. Fazemos a sua criação sem se preocupar com o jogo em si, e apenas com a tela.

- Dê um duplo clique no **background**.

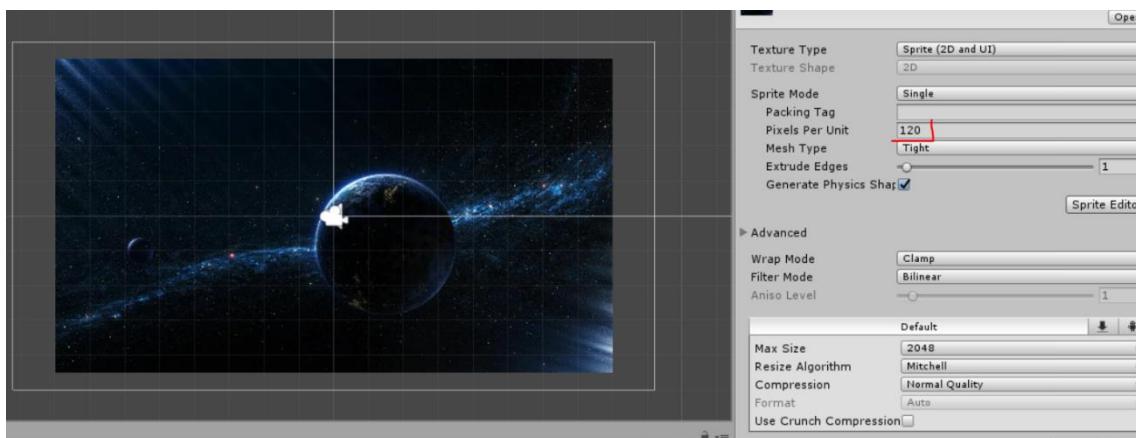


Iremos fazer alguns ajustes.

- Resete a posição do **background**.

- Na aba **Projects** selecione o background e vá na aba **Inspector**.

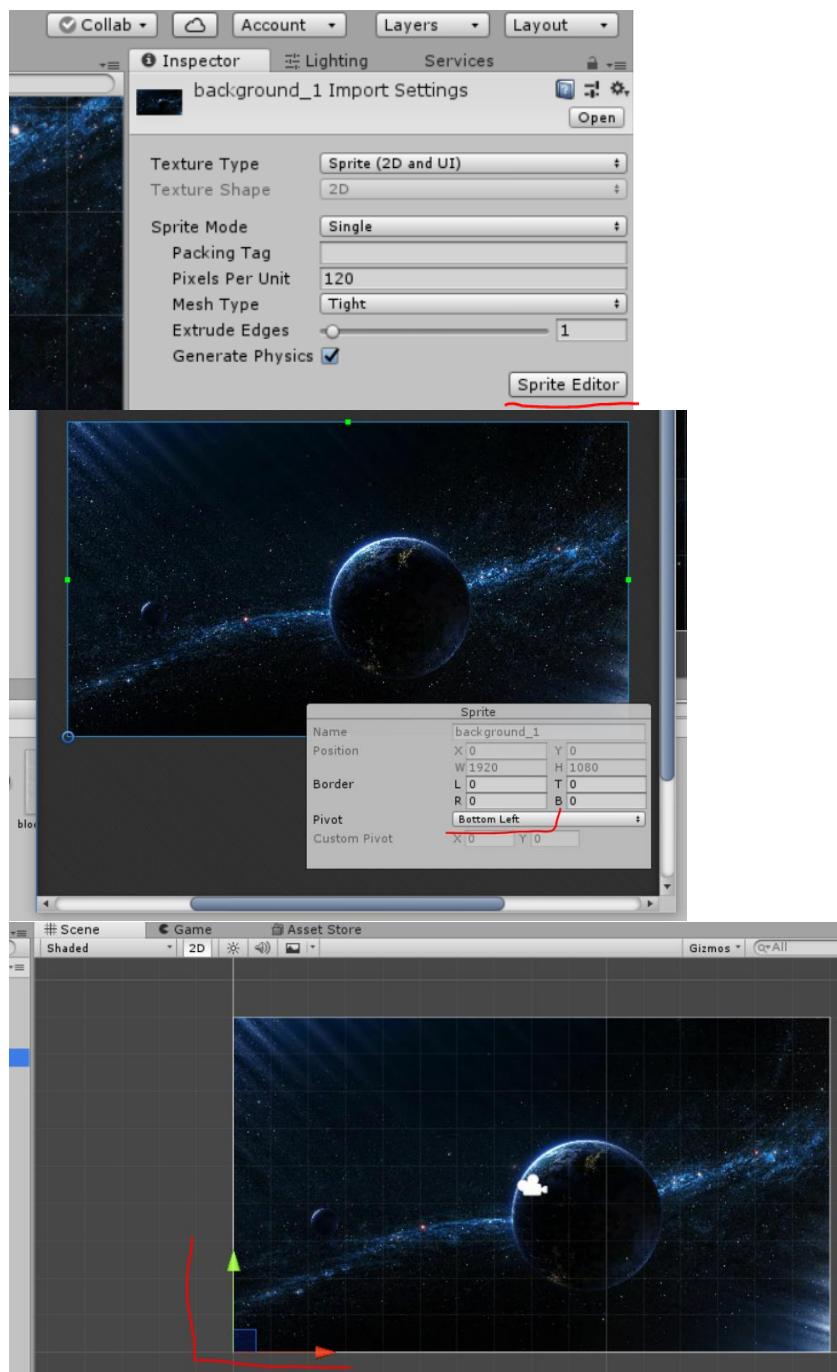
Queremos que o **background** ocupe 16 **world units**. Se ele possui largura de 1920 pixels, então fazemos $1920/16 = 120$. Ou seja, cada **world unit** deverá ter 120 pixels. Esse parâmetro é o **pixels per unit**. E cada imagem deverá ser configurado apropriadamente



Observe que agora a imagem ocupa exatamente 16 ***world units***. Você pode verificar isso pelo ***grid*** disponível na aba ***Scene***.

Outra operação que irá nos ajudar é colocar o ***pivot point*** do ***background*** no canto esquerdo inferior, ao invés de usar o centro.

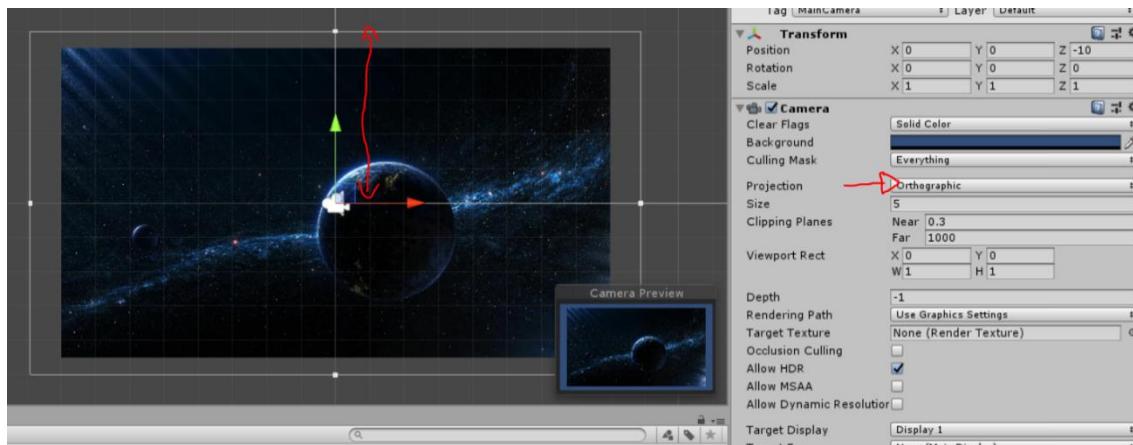
- Na aba ***Inspector*** vá em ***Sprite Editor***
 - Coloque o ***pivot*** no ***Bottom Left*** (canto esquerdo inferior).
- Observe que o ***gizmo*** agora fica no canto inferior esquerdo do ***background***, e não no meio.



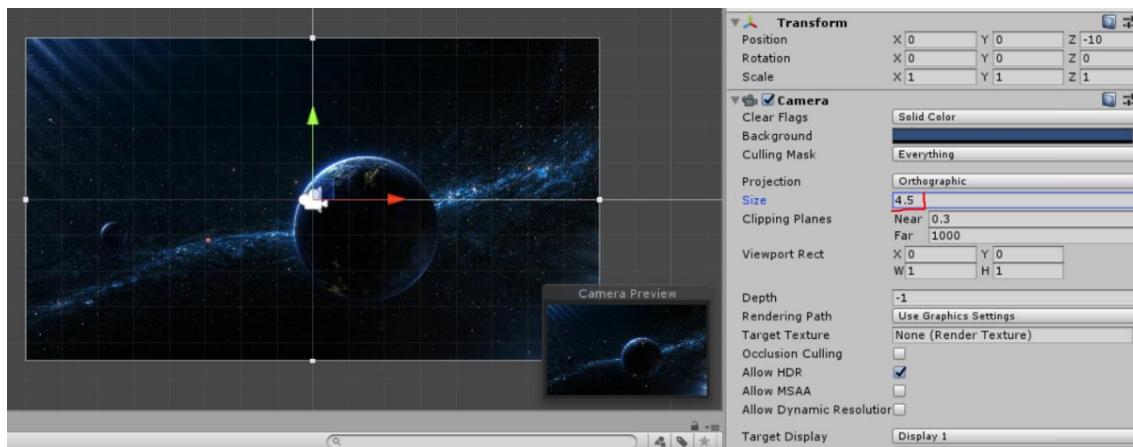
Configurando a Câmera

- Na aba **Games** coloque a resolução de 16:9
- No GO **Main Camera**, certifique que o **Projection** está ortográfico

Para calcular o **Size** da Câmera precisamos fazer uma análise rápida. Este parâmetro indica qual metade da altura da Câmera.

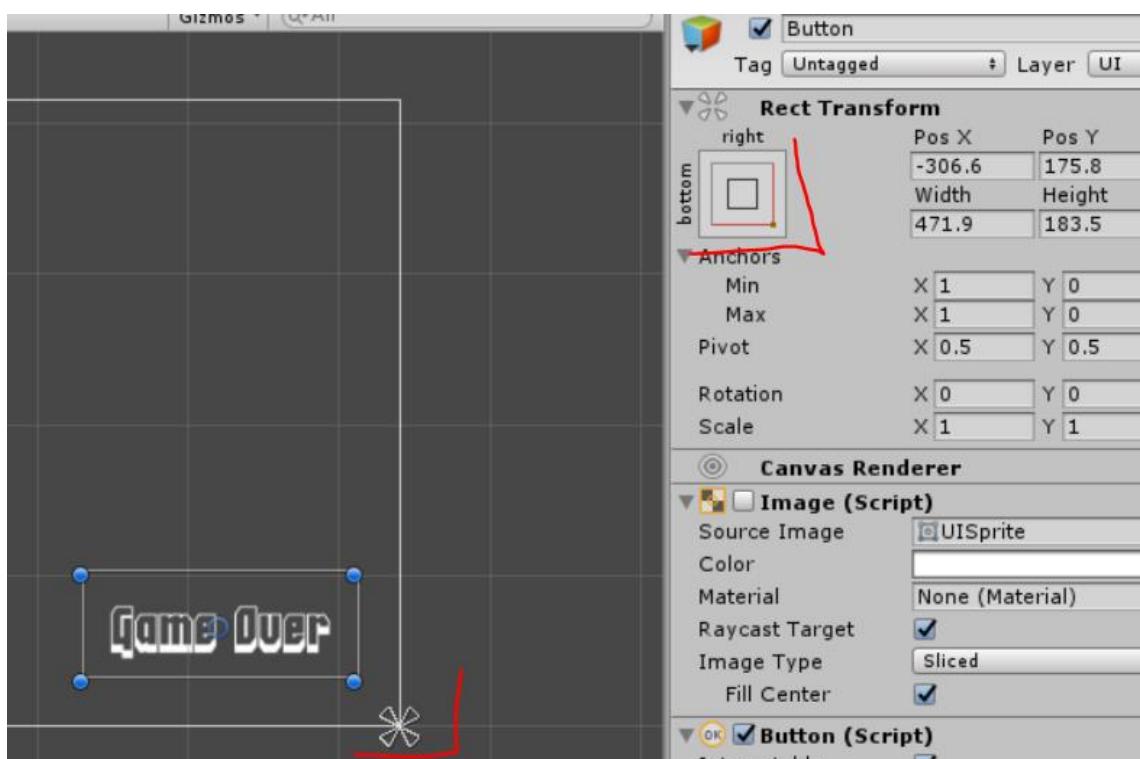
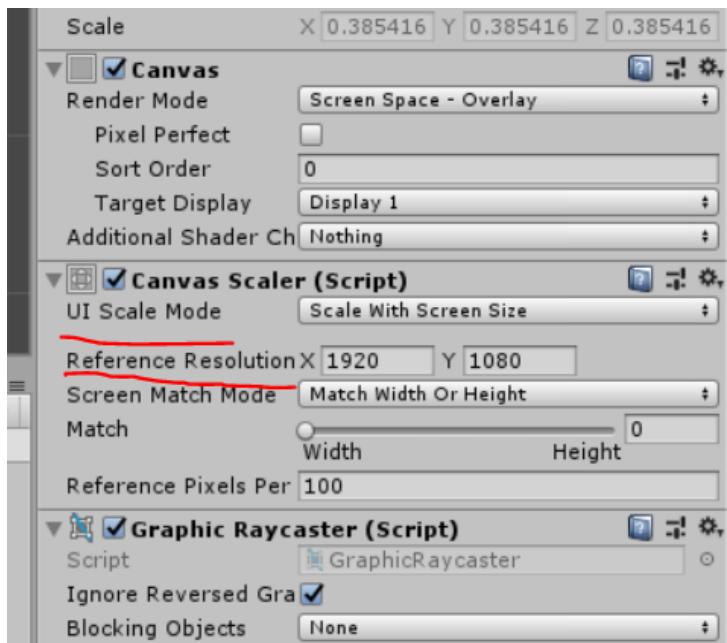


Se temos uma resolução de 16:9, e estamos ocupando 16 **world units** de largura, então a altura deverá ser $16/9 * 16 = 9$ (Uau). Assim o **Size** deverá ser $9/2 = 4.5$



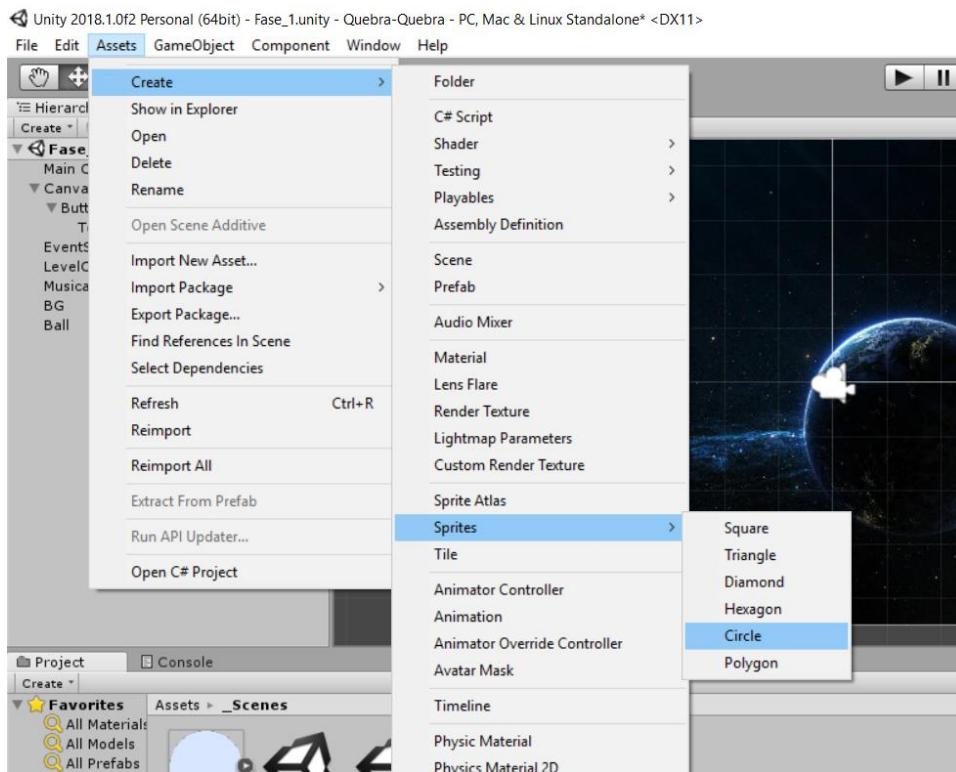
Vamos arrumar alguns detalhes no botão **Game Over**. Apesar de ser temporário, é uma boa oportunidade de praticarmos alguns conceitos de UI.

- Mude a Ancora do botão para o canto direito
- Mude o **Canvas Scale** para **Scale With Screen Size**

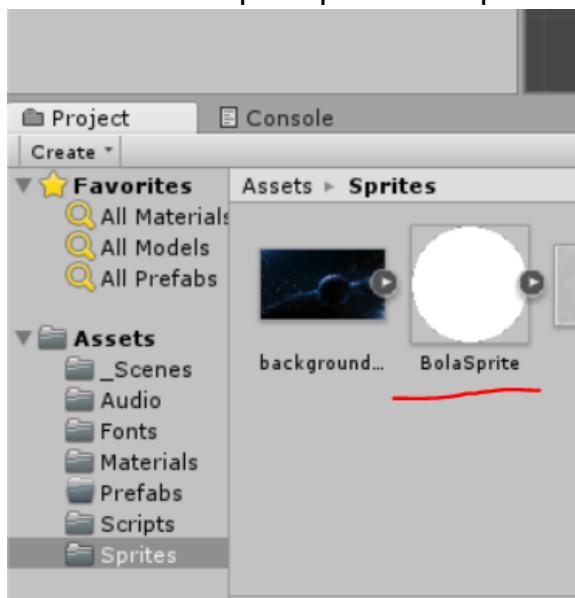


4 – Adicionando o elemento Bola

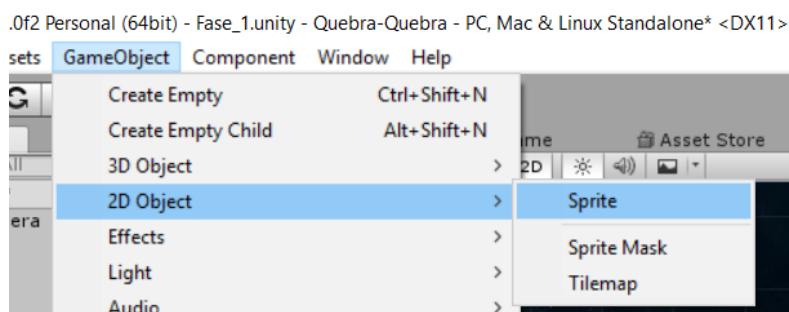
- Vá até a pasta **Sprite**
- Adicione um **Sprite** do tipo **Circle**. Observe a figura abaixo



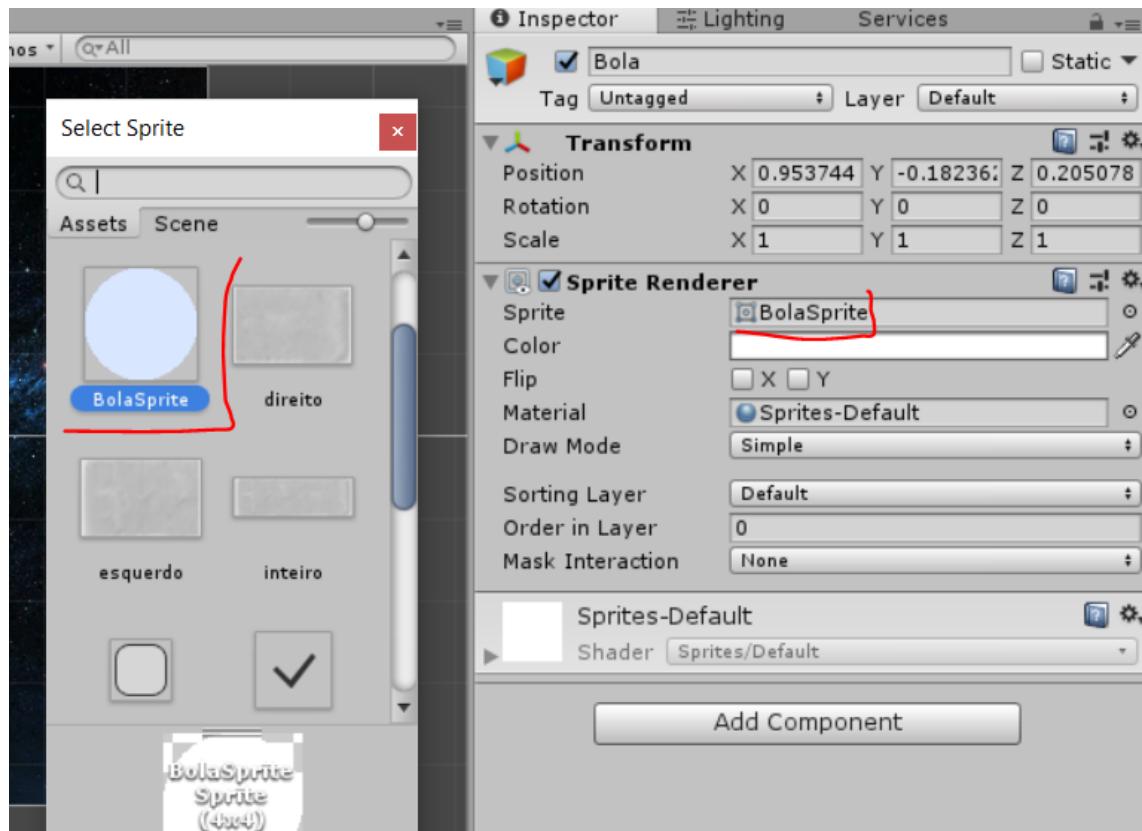
- Renomeie o Sprite para BolaSprite



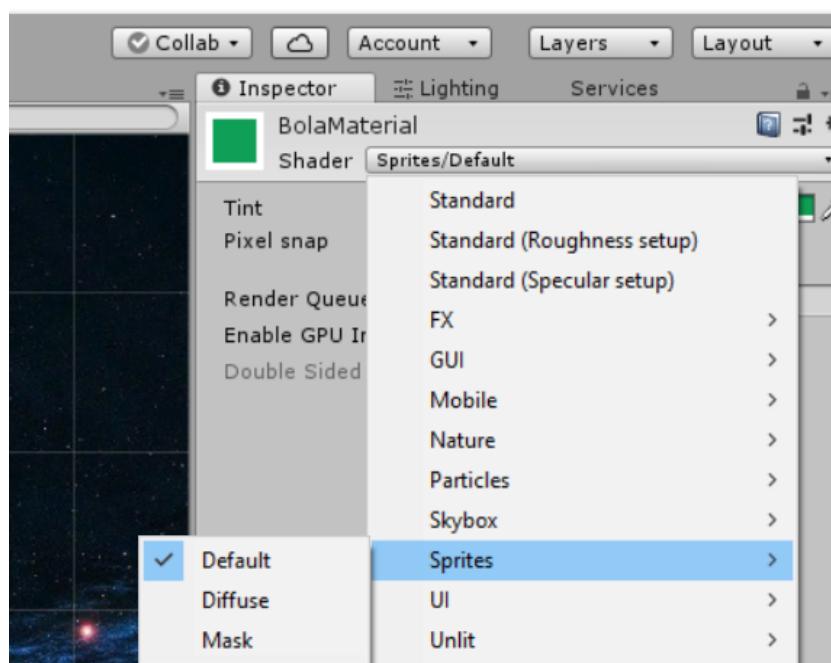
- Crie um GO **Sprite**. Chame de Bola



- No componente **Sprite Renderer** busque o **sprite** BolaSprite

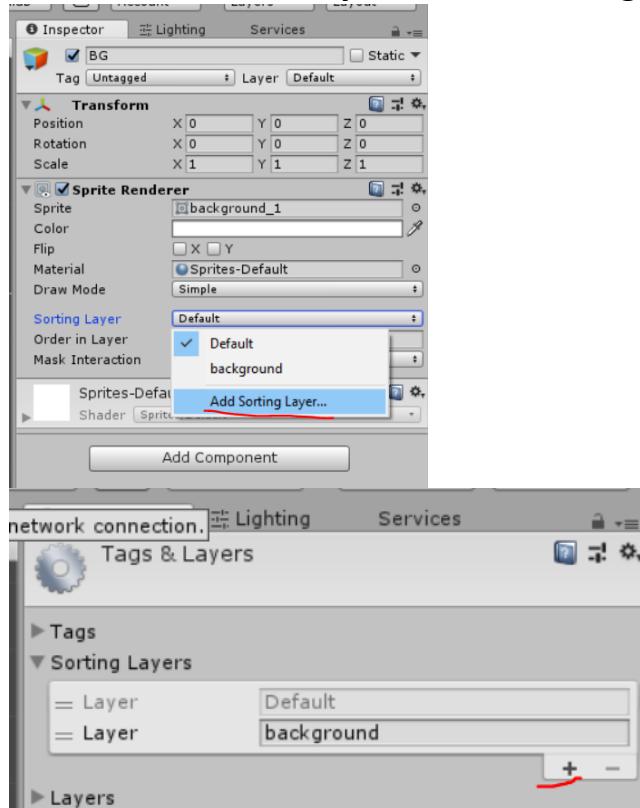


- Crie uma pasta **Materials** e crie um **material** chamada BolaMaterial
- Aplique o **material** no GO Bola
- Mude o **shader** desse **material** para **Sprite->Default**
- Coloque uma cor de sua preferência no **material**.



Se na **Scene** o GO Bola não estiver visível, uma opção é arrastá-lo para fora do **background** e ajustar o **material**. Porém, precisamos de uma solução definitiva. Uma opção é ajustar a coordenada Z da bola, para que esta seja renderizada antes do **background**. Mas essa não é a melhor solução. Vamos criar **layers** (camadas). Os **layers** informam a câmera a ordem que os GOs devem ser renderizados. Camadas mais alta são renderizadas antes.

- No GO BG crie um **layer** chamado **background**.

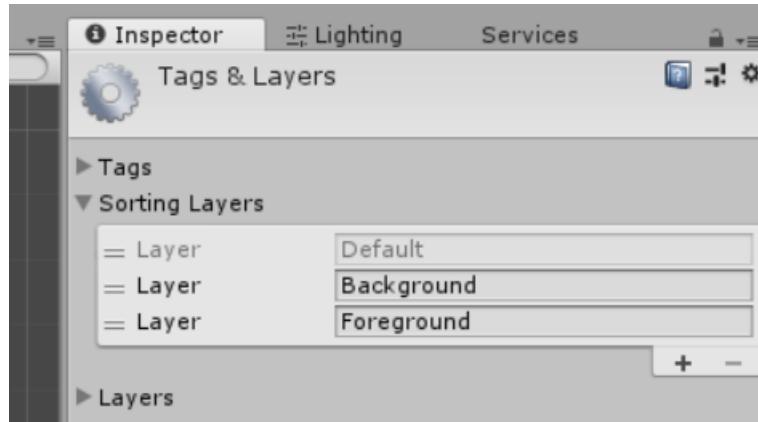


- No GO Bola crie e adicione o **layer** “foreground”
- Redimensione a bola pela metade.



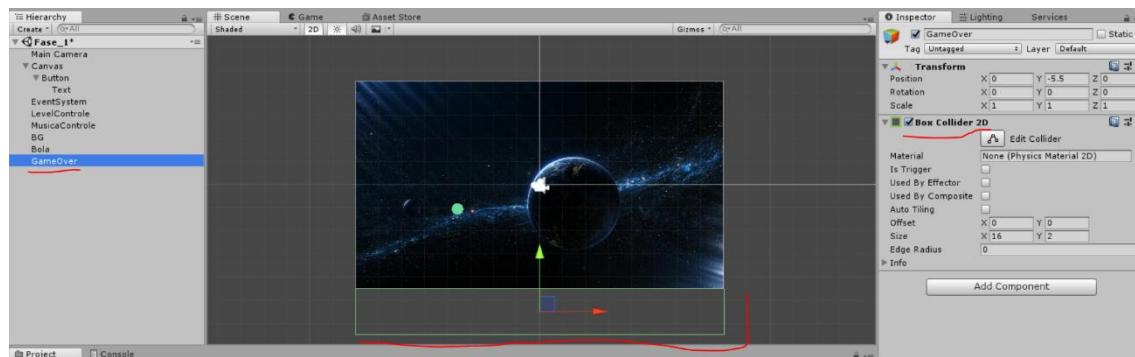
Na tela de criação de *layers*, repare que o “jogador” está logo embaixo do “background”. Por isso o “foreground” é renderizado antes

OBS: Estamos lidando na aba *Sorting Layer*, não confundir com a aba *Layer* que trata de colisões. Confuso não é mesmo?



Adicionando Física a Bola

- Adicione um *RigidBody2D* e um *CircleCollider2D* ao GO Bola
- Queremos que a Bola acerte os blocos, portanto deixe desmarcado *IsTrigger*. E deixe o *GravityScale* padrão.
- Crie um GO vazio chamado *GameOver* que será usado para identificar o fim do jogo. Adicione um *BoxCollider2D* no *GameOver* e marque como *IsTrigger*



- Faça um script que identifique colisão entre a Bola e o GO *GameOver*. Após detectar a colisão, carregue a *scene* TelaVitoria

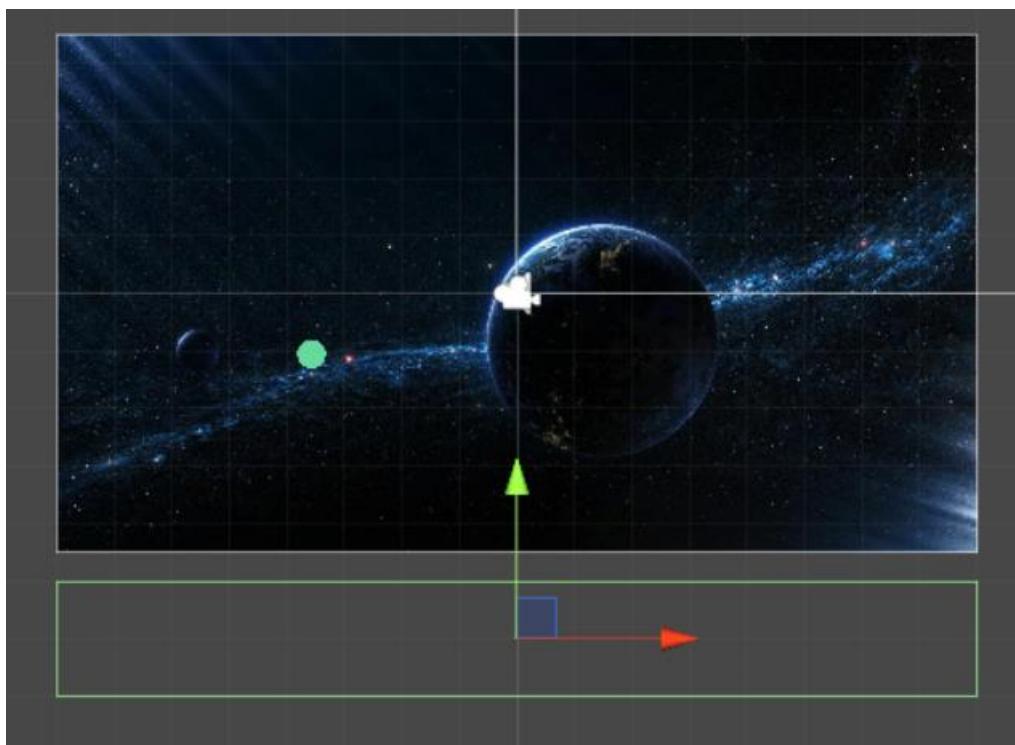
```

5     public class GameOverControle : MonoBehaviour {
6
7         [SerializeField]
8         private LevelControle levelControle;
9
10        private void OnTriggerEnter2D(Collider2D collision) {
11
12             levelControle.CarregaLevel("TelaVitoria");
13
14        }
15

```

- Lembre-se de associar o **LevelControle**.
- Adicione o **script GameOverControle** ao GO **GameOver**

Se der **Play** irá observar que assim que a bola bate no **GameOver** já ocorre a transição. Para ficar mais agradável esteticamente, desloque para baixo o **GameOver**.

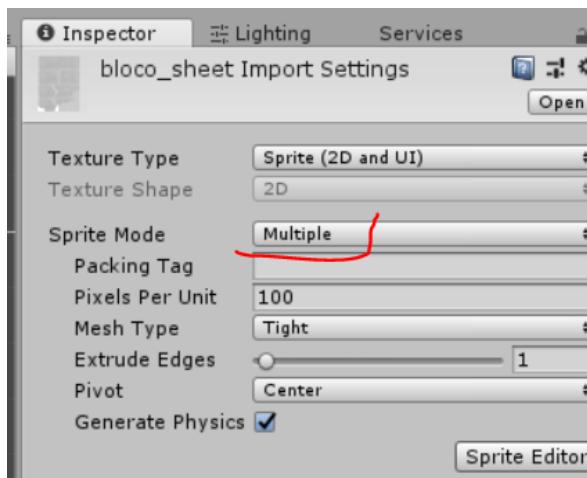


- Apague o **Canvas** e o **EventSystem** da **Fase_1**. Não precisaremos mais de UI por enquanto nessa **scene**.

5 - Adicionando a plataforma

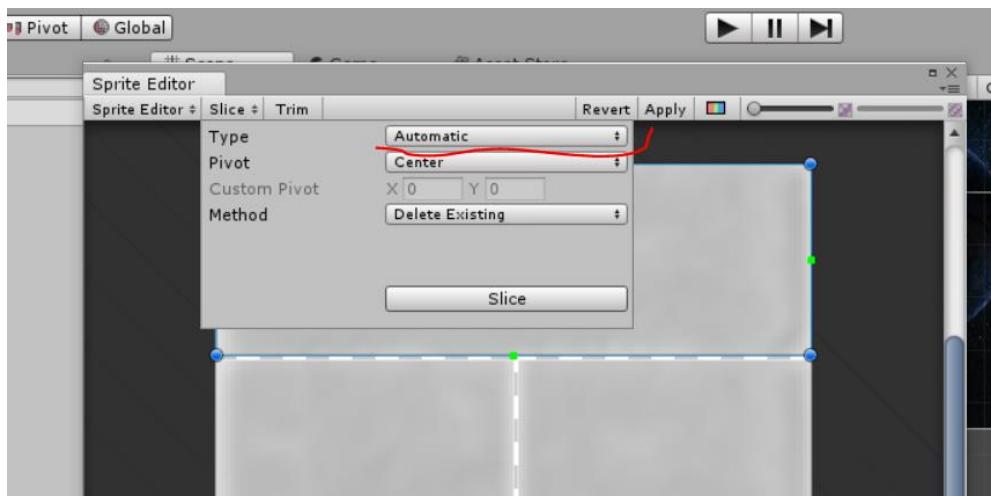
Devemos fazer pequenos ajustes nos **sprites**. Por padrão, o Unity importa **sprites** como se fossem **single**. Mas o arquivo **bloco_sheet** deveria ser do tipo **Multiple**, onde uma mesma figura contém múltiplos **sprites**, conhecido como **sprite_sheet**.

- Na pasta **Sprites** selecione o arquivo **bloco_sheet** e vá na aba **Inspector**. Altere o **Sprite Mode** para **Multiple**. Depois pressione **Apply**.

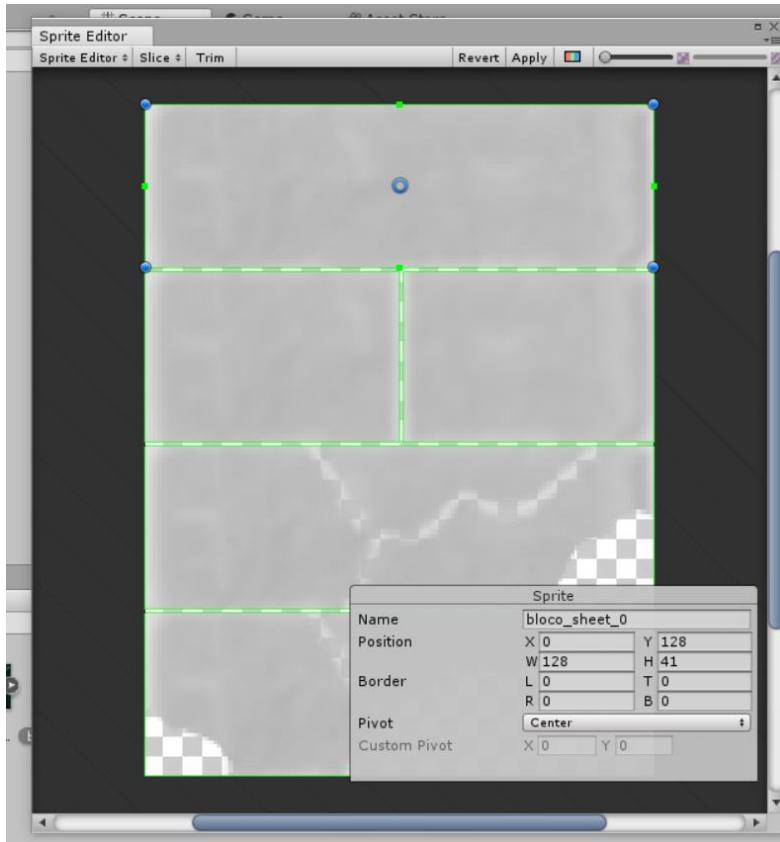


Voltando na pasta **Sprite** ainda sim os **sprites** não mostram as imagens de forma isolada, o que se tem é ainda uma imagem única. Vamos fazer os “cortes” (**slice**) nesse **Sprite**.

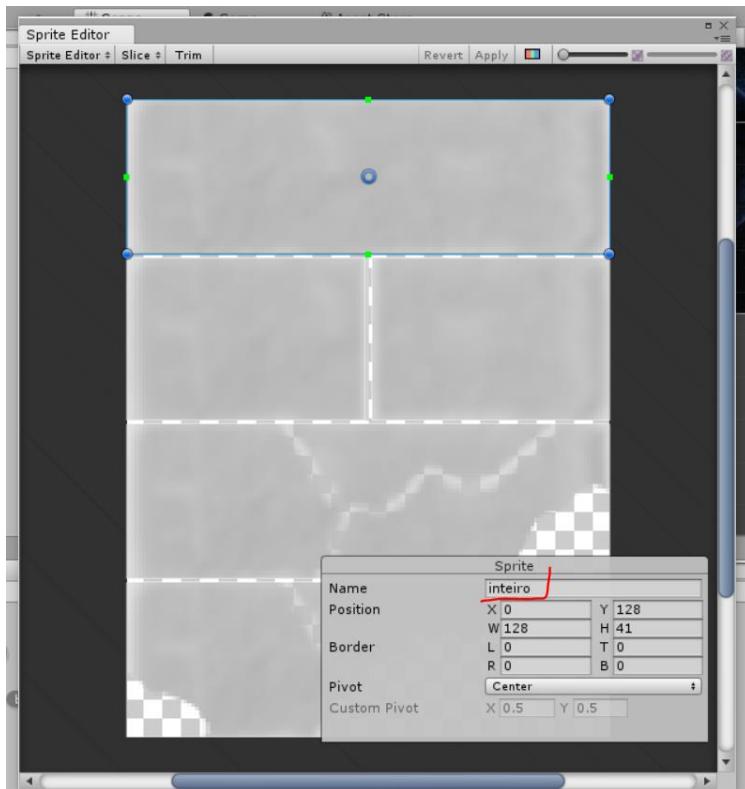
- Selecione o arquivo **sprite_sheet**
- Na aba **Inspector**, clique no **Sprite Editor**
- Faça o corte(**slice**) do tipo **Automatic**

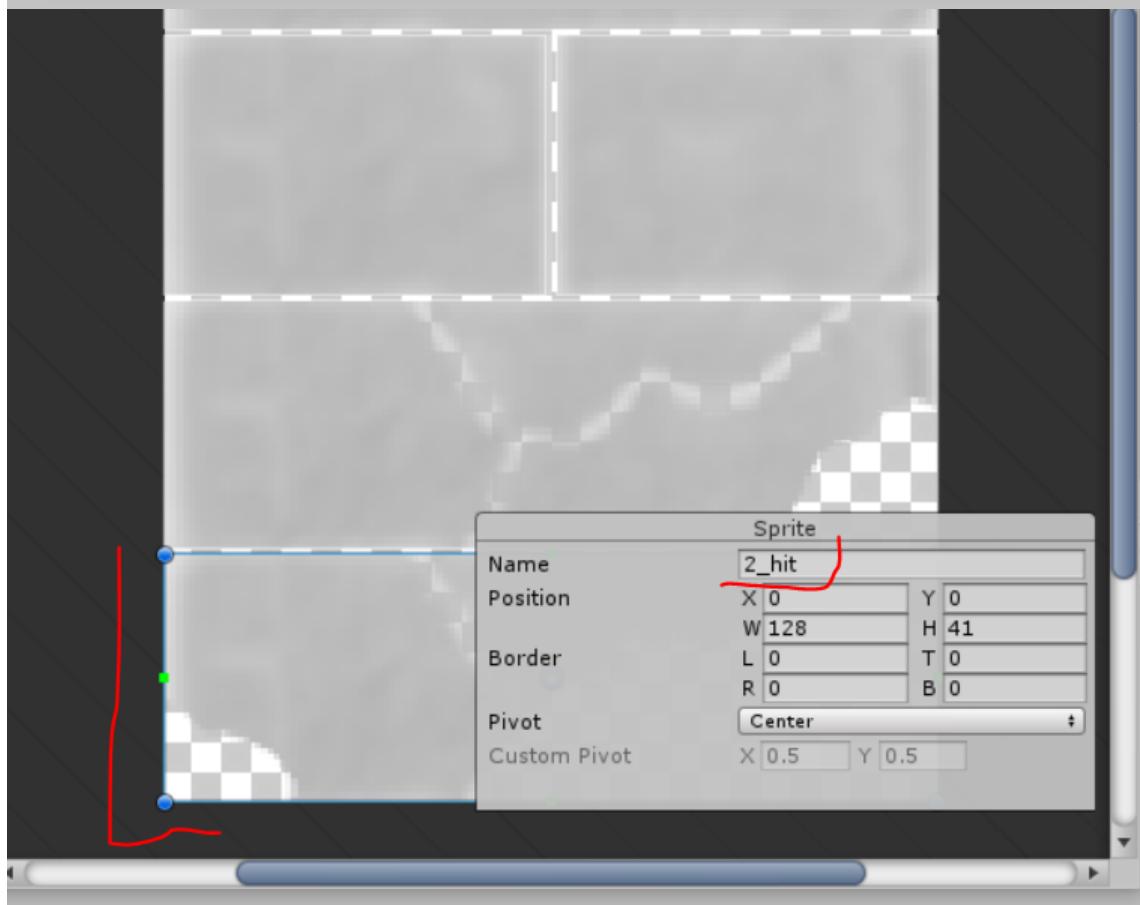


- Pressione o Ctrl (Command no Mac) e verá uma borda verde em volta dos *sprites* que foram identificados.

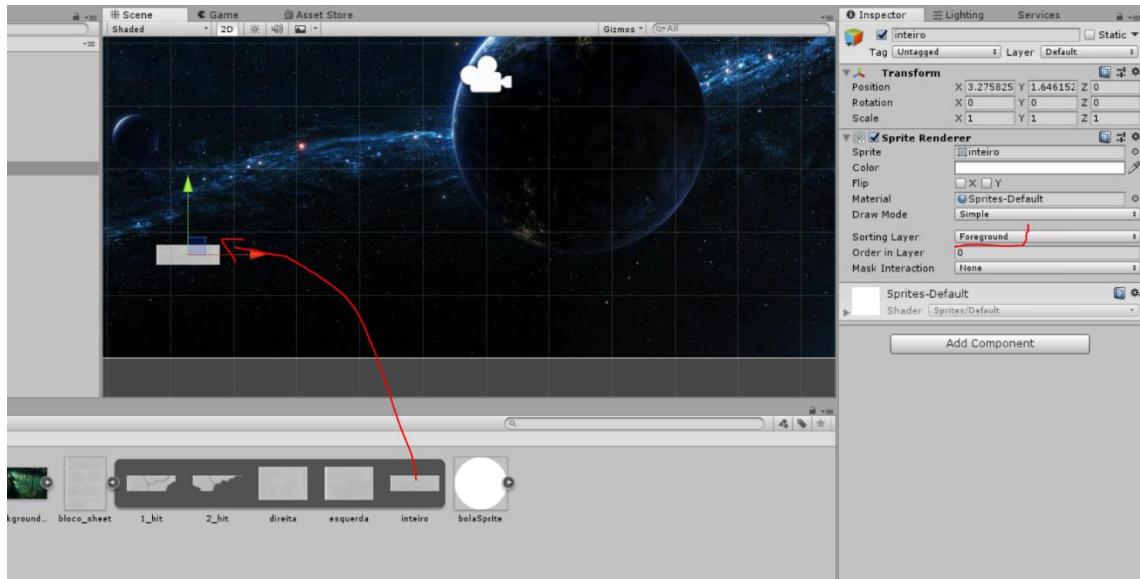


- Nomeie adequadamente os *sprites* conforme mostrado nas Figuras abaixo

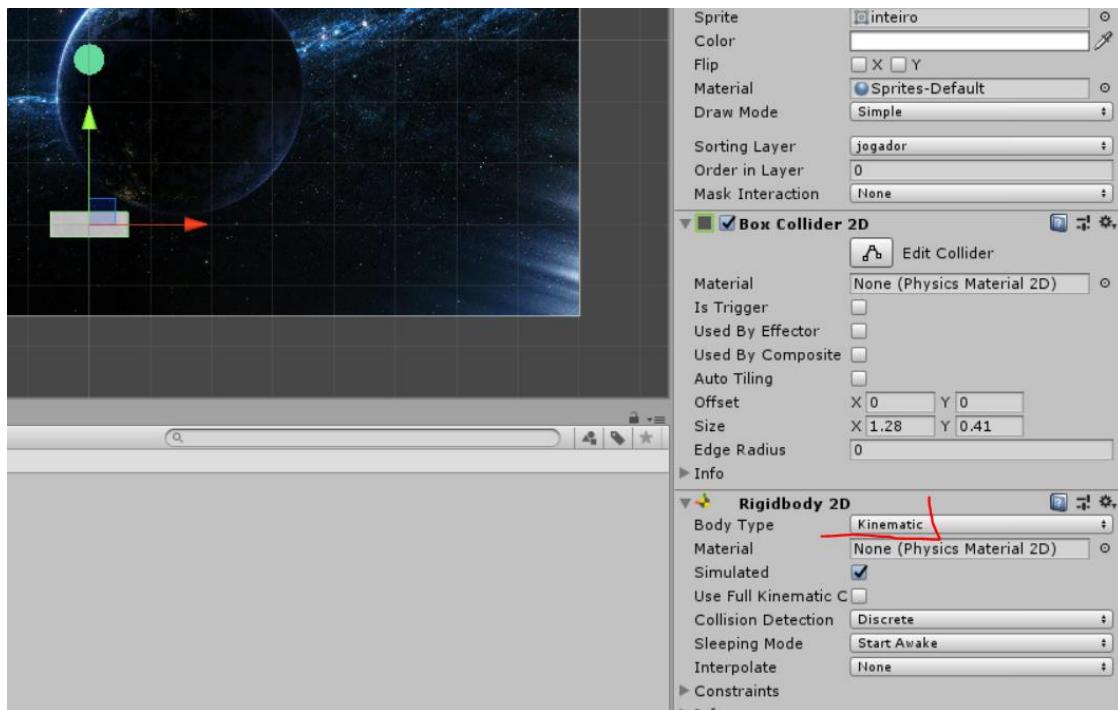




- Na pasta **Sprite**, adicione o **sprite** “inteiro” na **scene**. Para ter acesso você deve expandir o arquivo **bloco_sheet**. Coloque no **layer** “foreground”.



- Nomeie esse GO como **Plataforma**.
- Adicione um **BoxCollider2D** e um **RigidBody2D**. Marque como **Kinematic**. Isso quer dizer que a física não entrará em ação. Este objeto só será movido pela sua **Transform**.

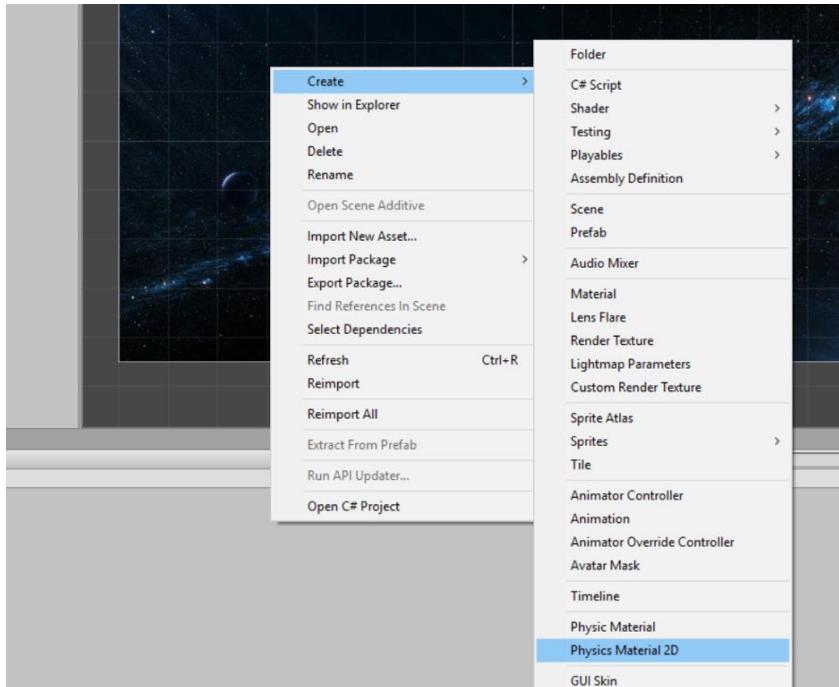


Adicionando Physics Material

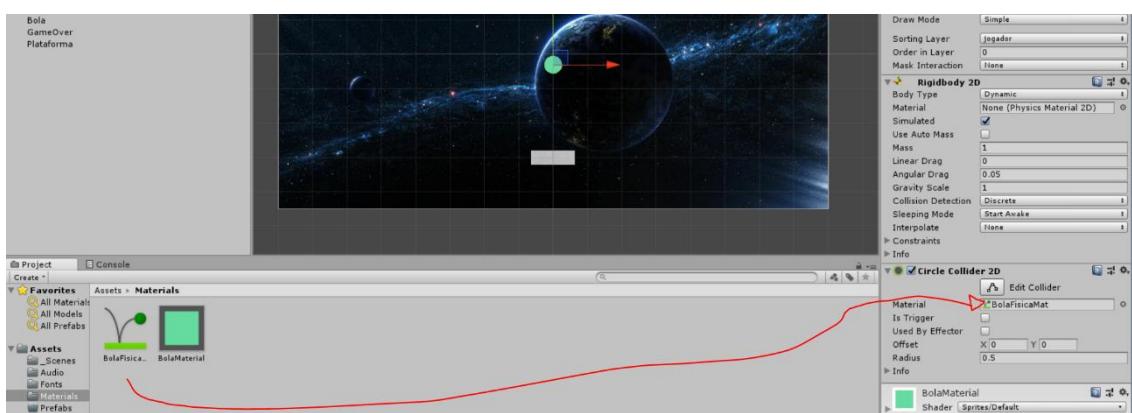
Por enquanto o jogo está bem sem graça. A bola bate na plataforma e nada acontece. Queremos que ela salte (ou quicar). Para isso precisamos

adicionar um **Physics Material**. Que possui duas propriedades: fricção e **bounciness** (que traduz para “coeficiente de salto”, elasticidade). Configurando essas duas propriedades, fazemos a bola quicar ao bater em um GO com algum tipo de **Collider**.

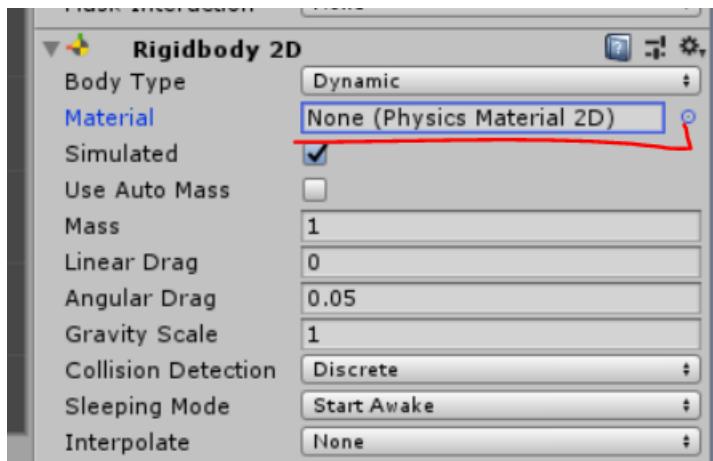
- Dentro da pasta **Materials**, adicione um **Physics2D Material** (pois nosso jogo é 2D): Botão Direito -> **Create > Physics2D Material**. Chame de **BolaFisicaMat**.



- Adicione esse **Physics2D Material** ao GO Bola. Isso é feito no campo **Material** do componente **BoxCollider2D**.

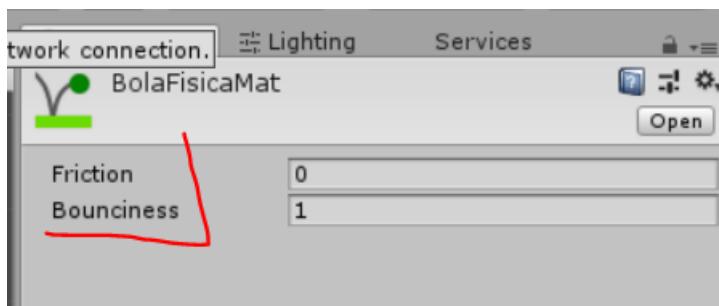


Por que não fizemos isso no campo **Material** do **RigidBody2D**?



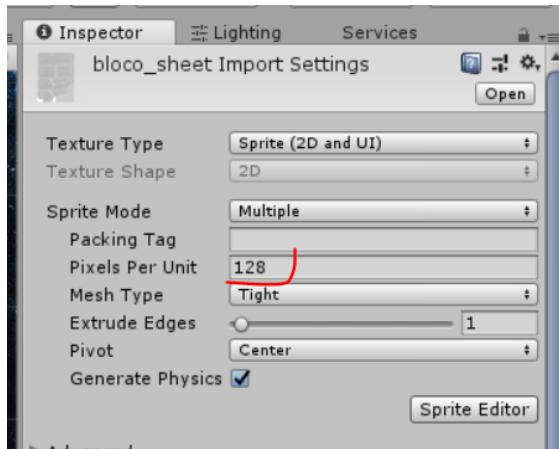
O **Material** do **RigidBody2D** serve para todos os **Colliders** anexado a este GO. Enquanto que o **Material** do **Collider** serve apenas para o **Collider** específico. Como temos apenas um **Collider**, não faz diferença. Mas caso você queira definir comportamentos diferente para outros **Colliders** no mesmo GO, você deverá usar o campo **Material** do próprio **Collider** (como fizemos).

- Coloque 0 (zero) no campo **friction** e 1 no campo **bounciness**. Isso quer dizer que 100% da energia será conservada na colisão, portanto a bola quicará para sempre.



Ajustando **ppu (pixels per unit)** da plataforma

Se você usar um software gráfico, como **Paint**, verá que o **sprite** “inteiro” possui 128pixels de largura. Nós desejamos que 1 bloco desse ocupe exatamente 1 **world unit**. Selecione o arquivo **bloco_sheet** (pois devemos mudar o **ppu** do arquivo todo) e na aba **Inspector**, mude o **ppu** para 128. Não esqueça de redimensionar o **BoxCollider2D** do GO **Plataforma**.



Movendo a Plataforma com o Mouse.

- Posicione a **Plataforma** em (8,0.5,0)
- Crie um **script** chamado **Plataforma** na pasta **Scripts**.

```

5   public class Plataforma : MonoBehaviour {
6
7     // Update is called once per frame
8     void Update () {
9       float mousePosWorldUnitX =
10      ((Input.mousePosition.x) / Screen.width * 16);
11      Vector2 plataformaPos =
12      new Vector2(0, transform.position.y);
13      plataformaPos.x = Mathf.Clamp(mousePosWorldUnitX, 0.5f, 15.5f);
14      transform.position = plataformaPos;
15    }

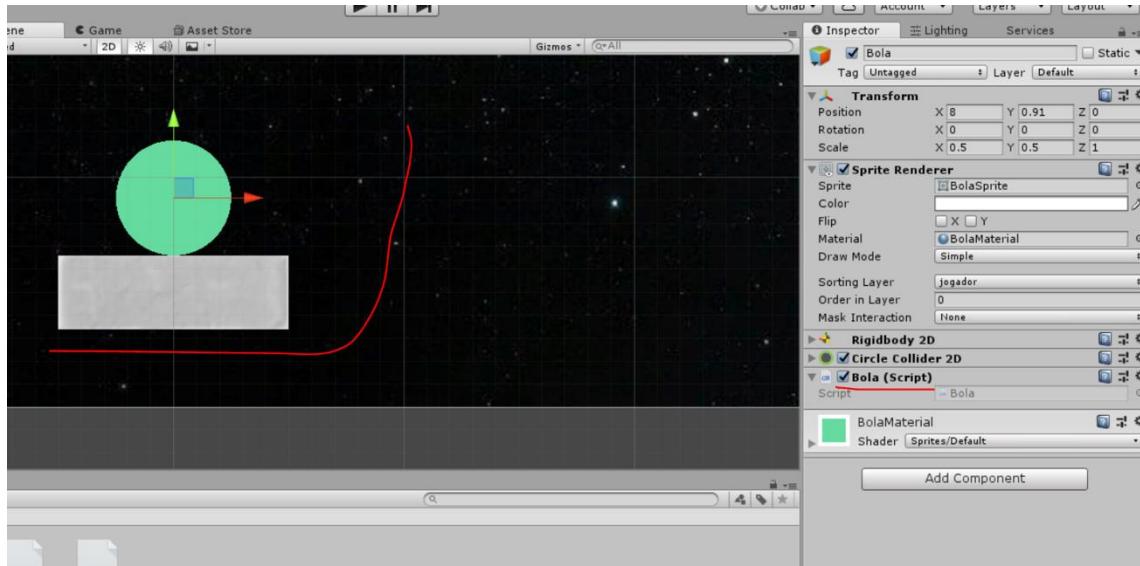
```

Na linha 9 fazemos a conversão da posição do mouse em **pixels** para **world unit**. Na linha 13 garantimos que esse valor não ultrapasse a faixa de 0.5f – 15.5 através da função **Mathf.Clamp()**. E finalmente atualizamos a posição da plataforma na linha 14.

- Associe esse **Script** ao GO **Plataforma** e dê **play**. Faça movimentos com o mouse e observe o resultado.

6 – Fazendo o lançamento da bola com clique do mouse.

- Crie um **script** chamado **Bola**
- Associe ao GO **Bola**
- Posicione o GO **Bola** bem próximo a **Plataforma** conforme Figura abaixo



Essa será considerada a posição padrão. Queremos que a Bola fique travada na Plataforma nessa posição e se movimente apenas no eixo X, isso dela ser lançada para iniciar o jogo.

- Abra o **script** Bola

```

5     public class Bola : MonoBehaviour {
6
7         [SerializeField]
8         private Plataforma plataforma;
9
10        private Vector3 plataformaBolaDis;
11
12        void Start() {
13            plataformaBolaDis = transform.position
14            - plataforma.transform.position;
15        }
16
17        void Update() {
18            transform.position = plataforma.transform.position +
19                            plataformaBolaDis;
20        }
21

```

Na linha 8 pegamos uma referência para a **Plataforma**. Na linha 13 calculamos a distância entre a **Bola** e a **Plataforma**. A ideia é manter a altura constante, pois queremos que a **Bola** se desloque apenas no eixo X nesse momento.

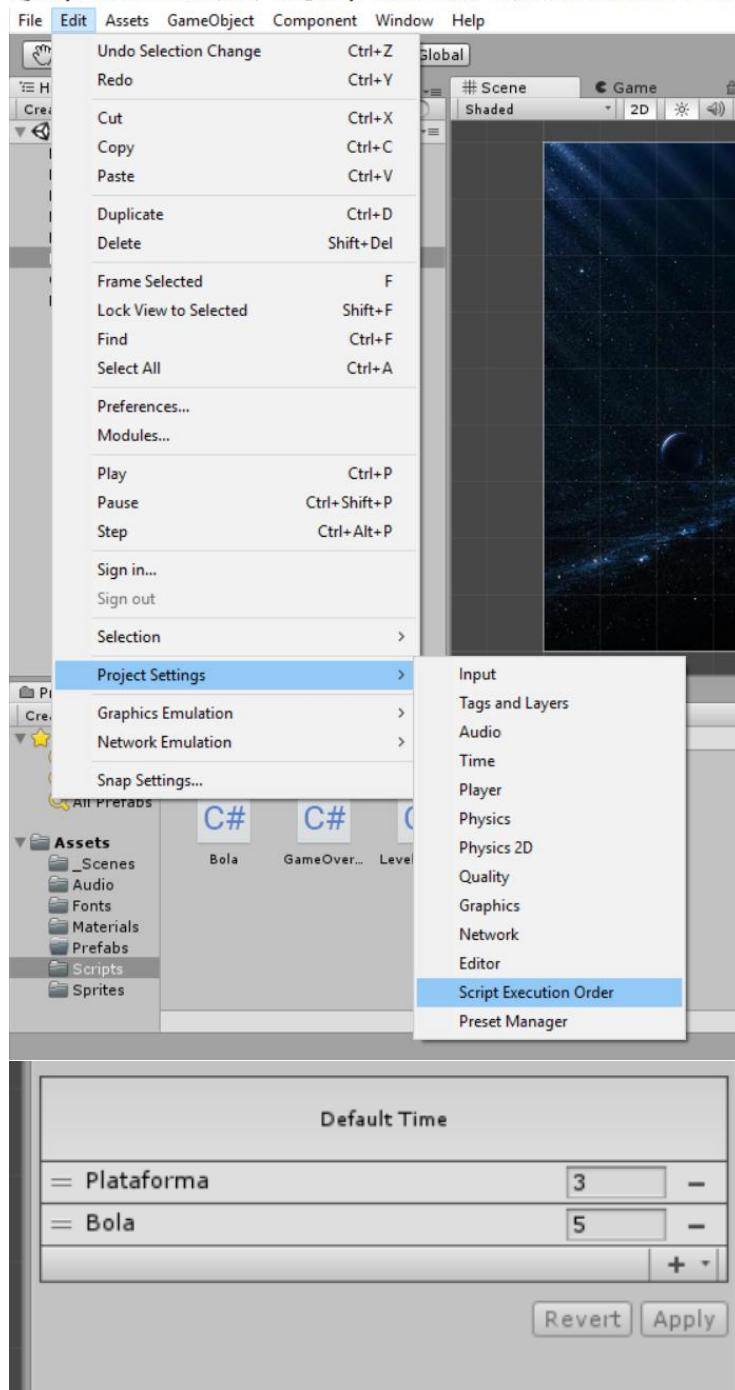
Dentro do **Update()** ficamos constantemente atualizando a posição da bola para que esta se mantenha constantemente presa a plataforma.

- Dê **play**. Movimente o **mouse** e veja o que está ocorrendo.

Dependendo da velocidade que você movimentar o mouse, talvez veja a bola deslocada para fora da plataforma e logo em seguida ela se reposicione. Isso pode ocorrer, caso sua máquina seja mais lenta e o script da Bola esteja sendo executado antes do script da Plataforma. O Unity permite termos controle na ordem de execução baseado no tipo da classe.

- Vá em **Edit->Project Settings -> Script Execution Order**

Unity 2018.1.0f2 Personal (64bit) - Fase_1.unity - Quebra-Quebra - PC, Mac & Linux Standalone* <DX1



- Inverta a Bola com a Plataforma e observe o resultado.
- Volte ao script Bola e faça as modificações abaixo.

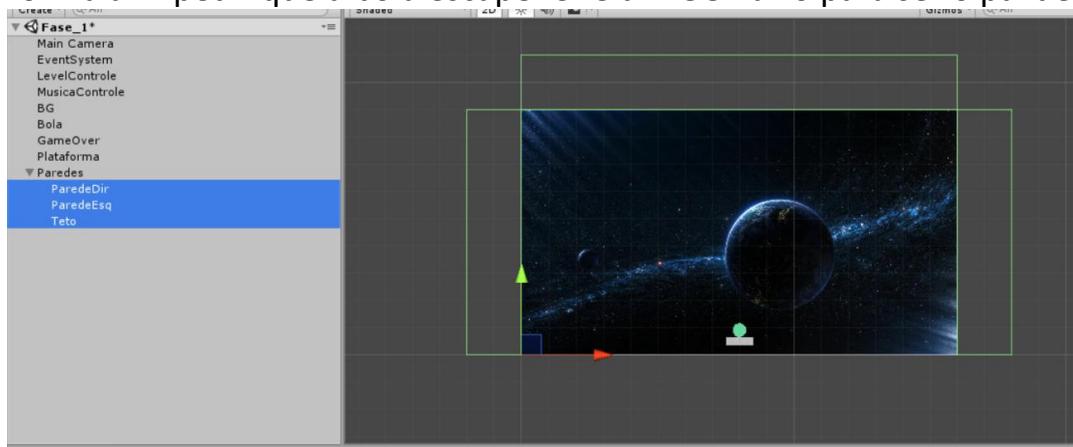
```

12     private Rigidbody2D rb2D;
13
14     private bool jogoComecou = false;
15
16     void Start() {
17         rb2D = GetComponent<Rigidbody2D>();
18         plataformaBolaDis = transform.position
19             - plataforma.transform.position;
20     }
21     void Update() {
22         if (!jogoComecou) {
23             transform.position = plataforma.transform.position +
24                 plataformaBolaDis;
25             if (Input.GetMouseButtonDown(0)) {
26                 rb2D.velocity = new Vector2(2f, 10f);
27                 jogoComecou = true;
28             }
29         }
30     }

```

Na linha 12 temos o acesso ao ***RigidBody2d***. Na linha 14 uma variável para controlar se o jogo já começou (isso irá acontecer depois que clicarmos no mouse). Na linha 18, a variável ***plataformaBolaDis*** é do tipo ***Vector3***. Dentro do ***Update()*** verificamos se o jogo já começou. Enquanto isso não ocorrer a bola ficará presa a plataforma. Assim que o botão do mouse for pressionado aplicamos uma velocidade a Bola e marcamos ***jogoComecou*** como ***true***. Se continuar jogando, irá observar que a bola pode sumir pelas laterais da tela. Vamos resolver isso criando paredes invisíveis como ***Colliders***.

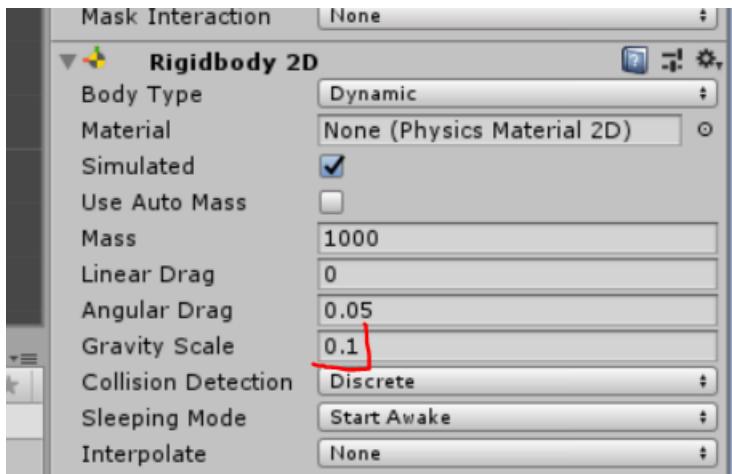
- Crie 3 (três) GOs vazios. Adicione um ***BoxCollider2D*** em cada uma de forma a impedir que a bola escape. Crie um GO vazio para ser o pai desses



- Teste o jogo

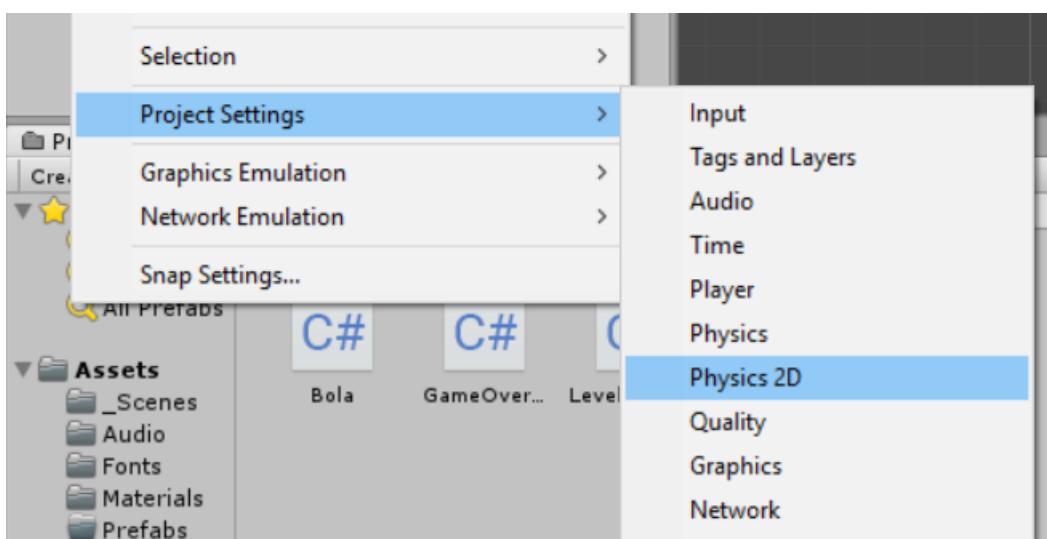
Aumentando a altura que a bola atinge

Queremos que a bola chegue até o teto. Uma forma pode ser alterarmos como a gravidade está influenciando a bola.

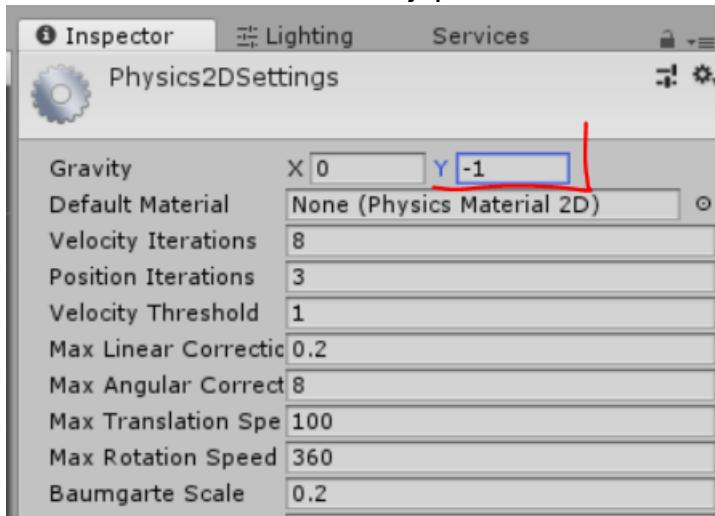


- Coloque o valor em 0.1 e o valor 10 e veja o que ocorre.
- Irá observar que deixando em 0.1 a bola salta bem mais alto. Isso por que a gravidade está exercendo menos força na bola. O problema dessa abordagem é que objetos diferentes irão acelerar de forma diferente. Isso pode trazer certos desequilíbrios para o jogo. Mas continua sendo uma opção. Outra abordagem é alterar a gravidade para o jogo todo.

- Vá em **Edit -> Project Settings -> Physics2D**



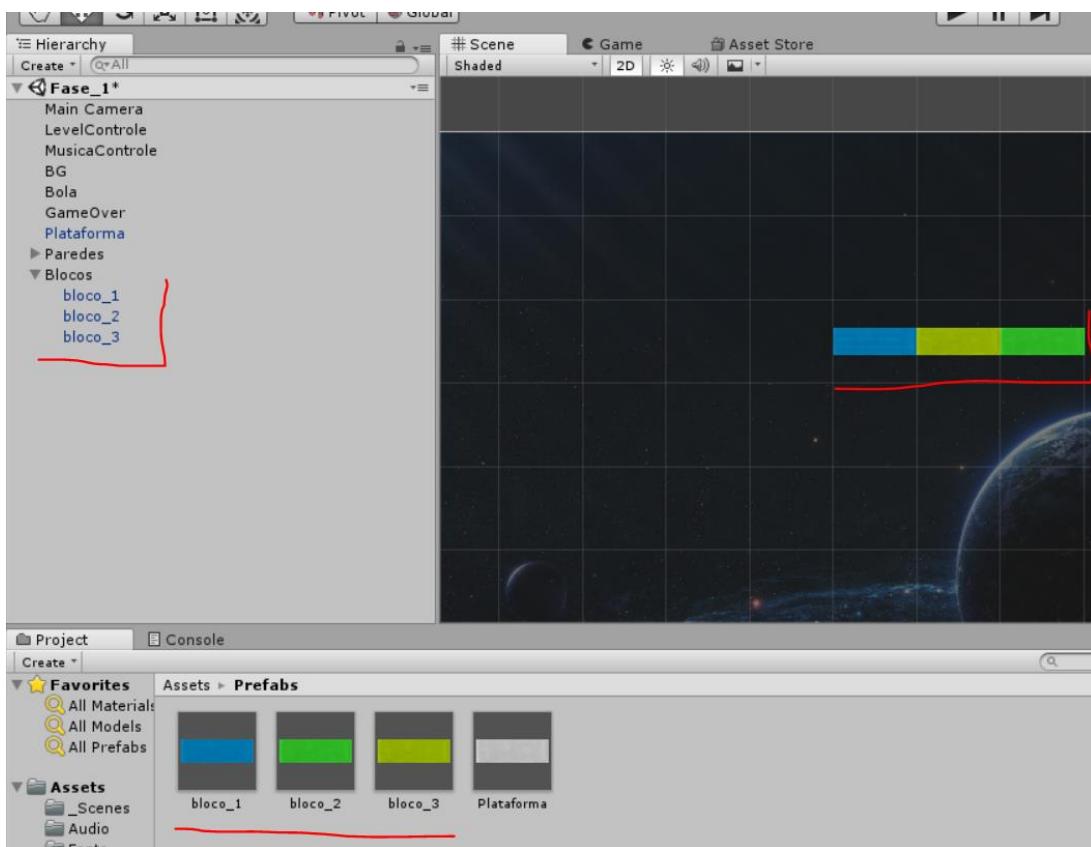
- Mude o valor Y de **Gravity** para -1



- Teste o jogo.

7 – Criando blocos

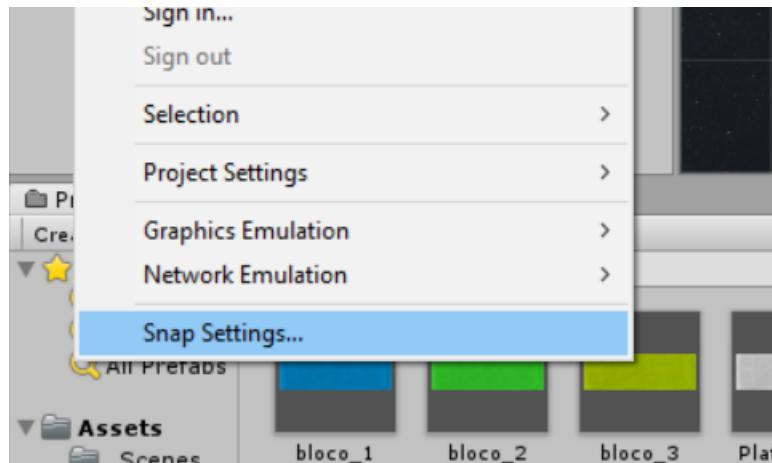
- Duplique o GO Plataforma. Chame de Bloco_1. E o torne um **prefab**.
- Mude a cor para algo de sua preferência.
- Crie outros dois GOs – Bloco_2 e Bloco_3 e os transforme em **prefab**.
- Mude a cor destes dois últimos.



Usando *Snap Settings* para mover GOs

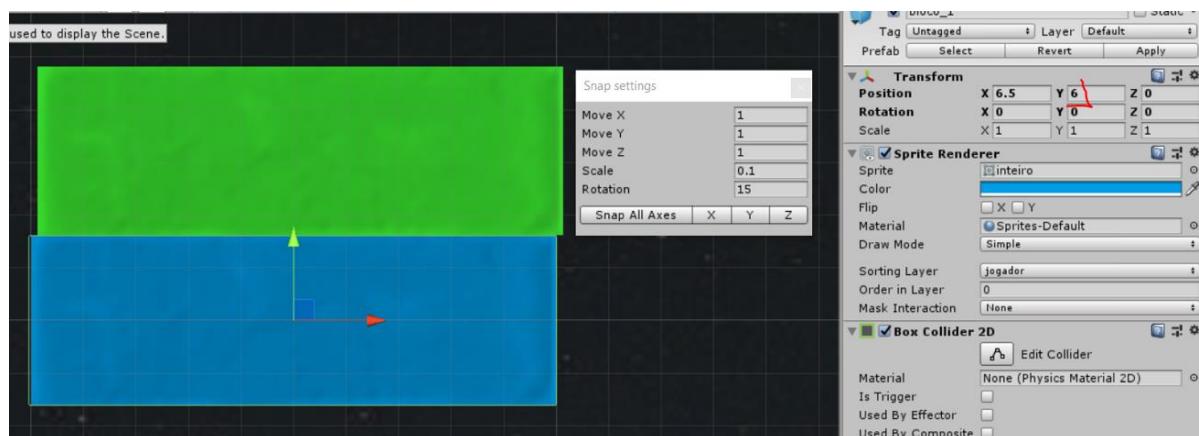
Você deve ter notado que as vezes mover os GOs, principalmente esses blocos, o movimento possui granularidade muito fina e sempre temos que colocar os valores manualmente no componente **Transform**. O desejável seria mover os blocos, na aba **Scene**, de acordo com um *grid*.

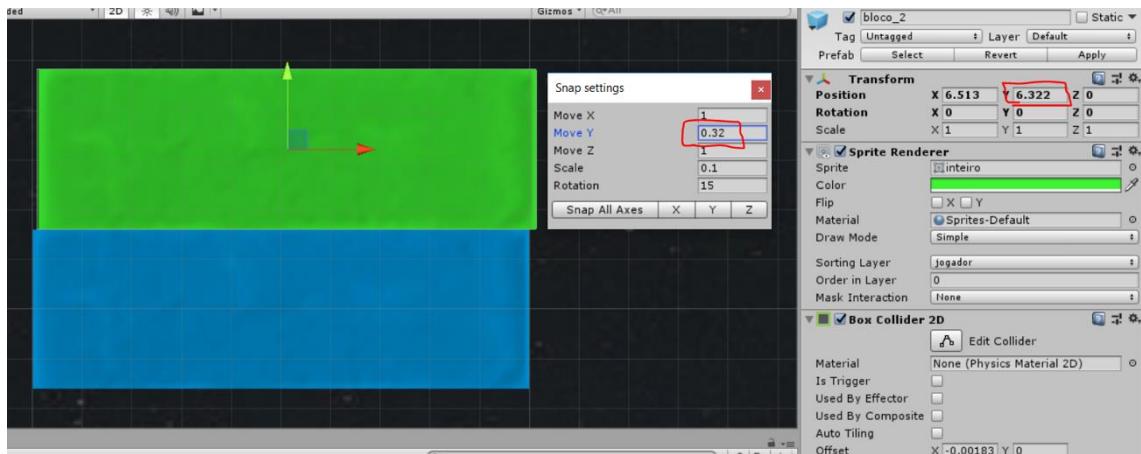
- Vá em **Edit -> Snap Settings...**



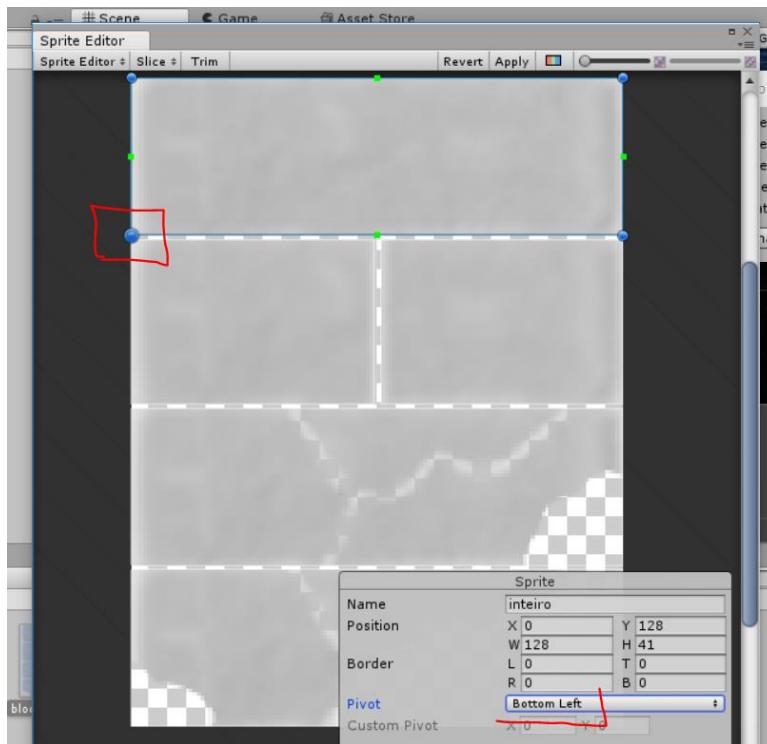
A tela **Snap Settings** possui dois campos importantes para: **Move X** e **Move Y**. No primeiro vamos deixar 1 (um). Pois queremos que o bloco move 1 **world unit** na horizontal.

Queremos que o bloco move na vertical 1 unidade de bloco. Para ver que valor é esse podemos usar a seguinte técnica. Observe na Figura abaixo que o Bloco_1 está em Y=6. E o Bloco_2 está em 6.322. Isso sugere que a altura do Bloco_2 é 0.32 **world units**. Assim, na tela **Snap Settings** coloque **Move Y** para 0.32

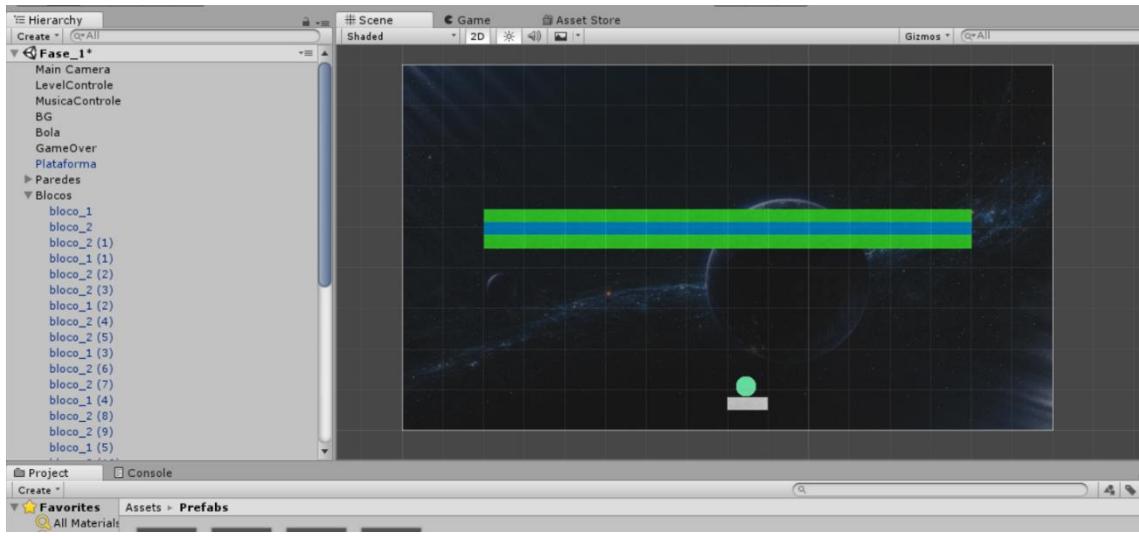




- Com os Blocos selecionados, clique em ***Snap All Axes***. Isso colocará os Blocos nos locais adequados para esse ***grid***.
- Mova os blocos, mas segurando o botão ***Ctrl***. Isso faz com que os blocos movam dentro do ***grid*** criado pelo ***Snap Settings***.
- Se ainda não tiver feito, mude o pivot do ***Sprite*** “inteiro” para o canto esquerdo inferior. Não esqueça de resetar o ***Collider*** dos Blocos e clicar em ***Apply*** para que as modificações espalhem para o ***prefab***.
- No ***script Plataforma***, altere o argumento do ***Mathf.Clamp()*** para 0.0f e 15.0f. Isso é necessário pois mudamos o ***pivot point*** do ***sprite*** “inteiro”.



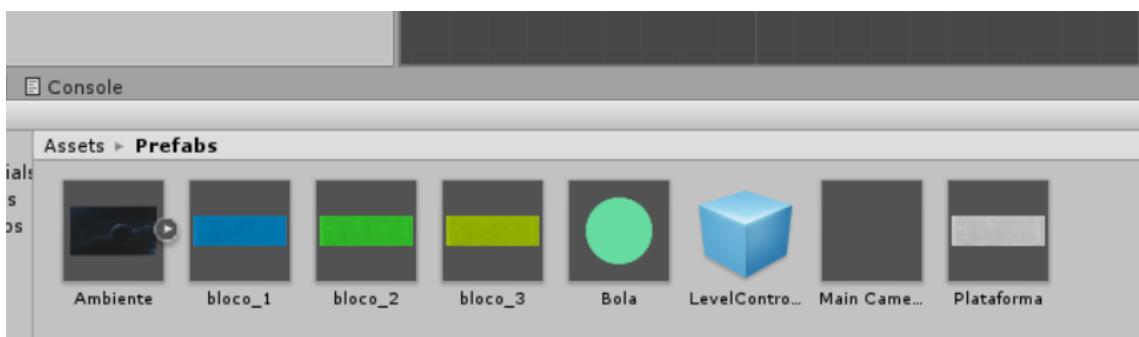
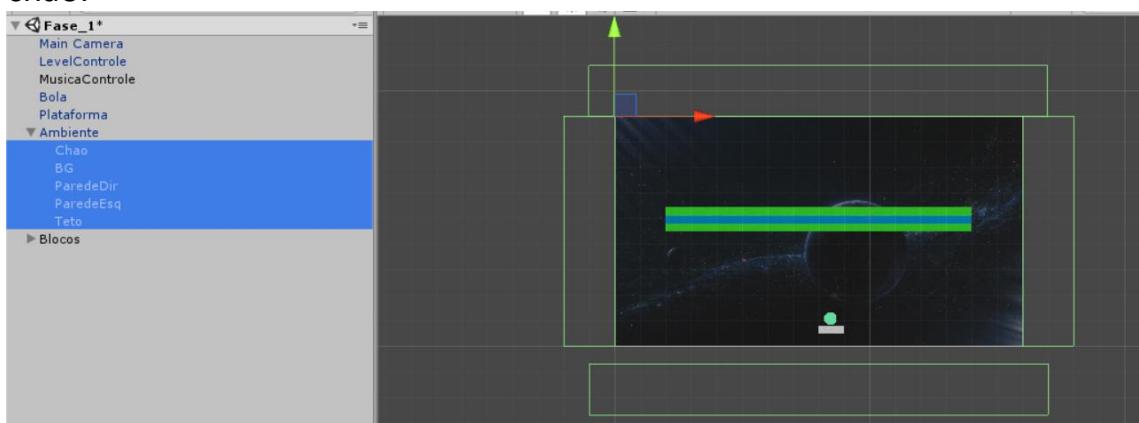
- Adicione mais blocos, criando um ***scene*** completo.



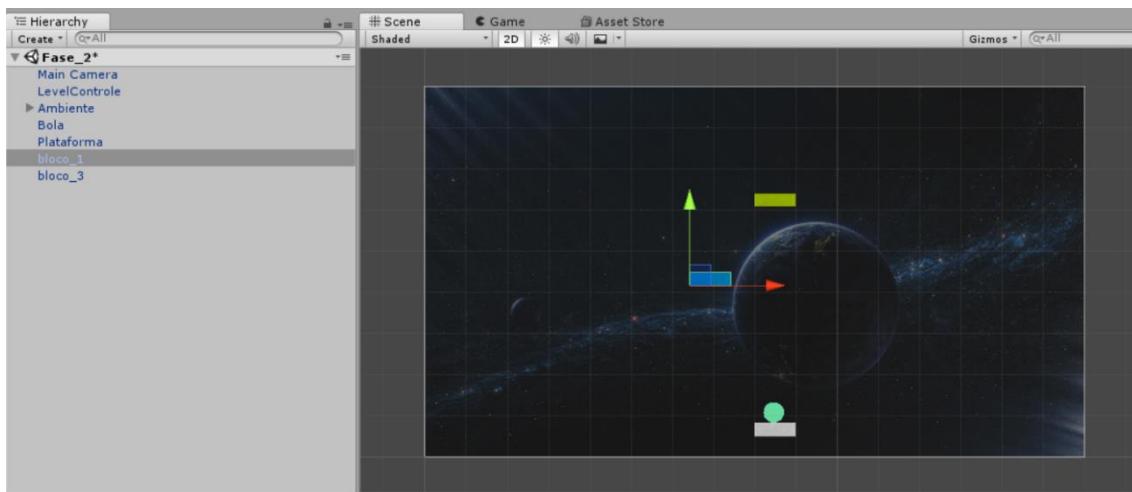
8 – Criar a Fase_2 a partir de prefabs

Nessa etapa iremos transformar quase tudo em ***prefabs***, e iremos criar uma fase adicional

- Transforme a **Main Camera**, **Bola**, **Ambiente** e o **LevelControle** em um ***prefab***. Se você não tiver o GO **Ambiente**, ele é o pai das paredes, teto e chão.



- Crie uma **Scene** chamada Fase_2
- Coloque os prefabs **Main Camera**, **Bola**, **Plataforma**, **LevelControle**, **Ambiente** e posicione os **Blocos** a seu critério.



Se você testar a fase, observará alguns erros. O primeiro é o **NullPointerException** que se dá pelo fato da Bola não achar a Plataforma.

- Abra o script **Bola** e faça as seguintes modificações.

```

 7   private Plataforma plataforma;
 8
 9   private Vector3 plataformaBolaDis;
10
11  private Rigidbody2D rb2D;
12
13  private bool jogoComecou = false;
14
15  void Start() {
16      rb2D = GetComponent<Rigidbody2D>();
17      plataforma = FindObjectOfType<Plataforma>();
18      plataformaBolaDis = transform.position
19                  - plataforma.transform.position;
20  }
  ...

```

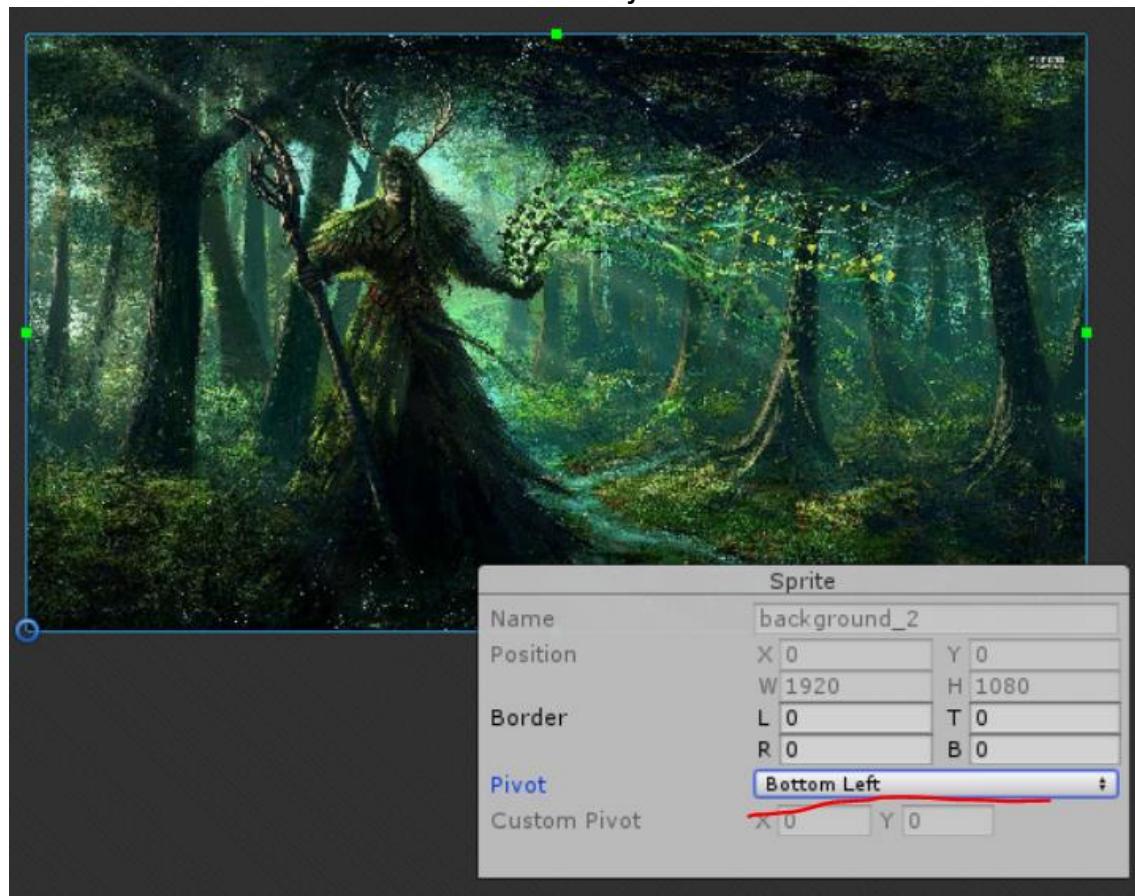
Não exponha a Plataforma no **Editor** (removendo o [Serialized]) e use **FindObjectOfType** para buscar a plataforma na **scene**.

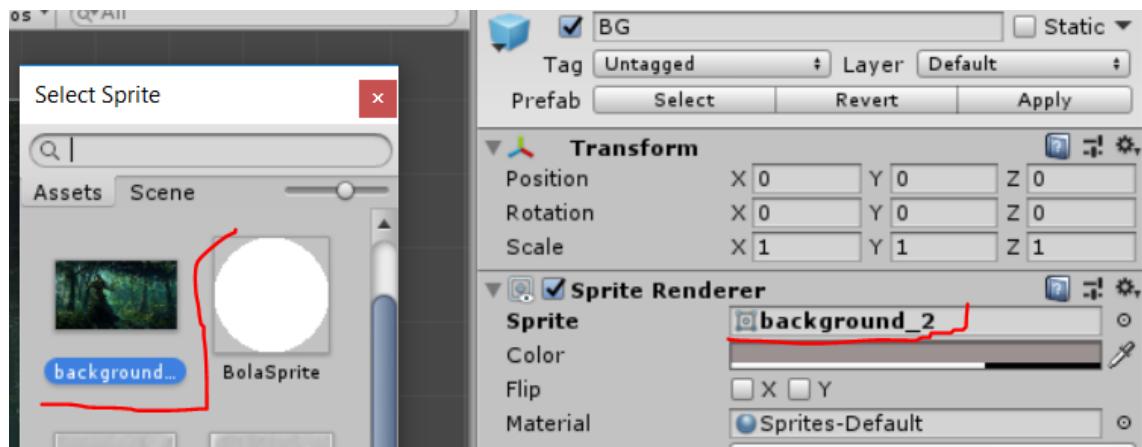
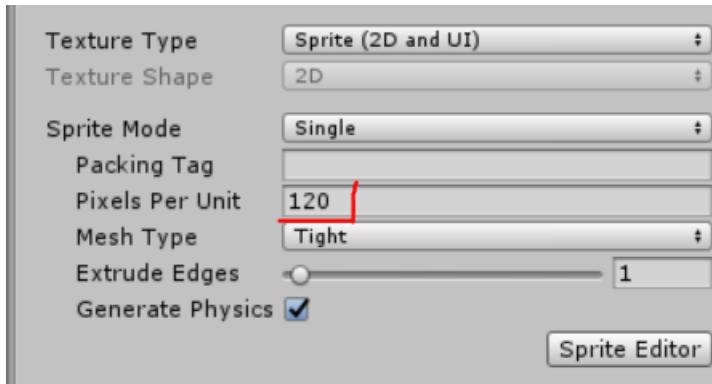
- Faça a mesma coisa para o GO **Chao** (responsável por detectar fim do jogo). Busque o **LevelControle** usando **FindObjectOfType()**.

```

5  public class GameOverControle : MonoBehaviour {
6
7      private LevelControle levelControle;
8
9      private void Start() {
10         levelControle = GameObject.
11             FindObjectOfType<LevelControle>();
12     }
13
14     private void OnTriggerEnter2D(Collider2D collision) {
15
16         levelControle.CarregaLevel("TelaVitoria");
17     }
18 }
```

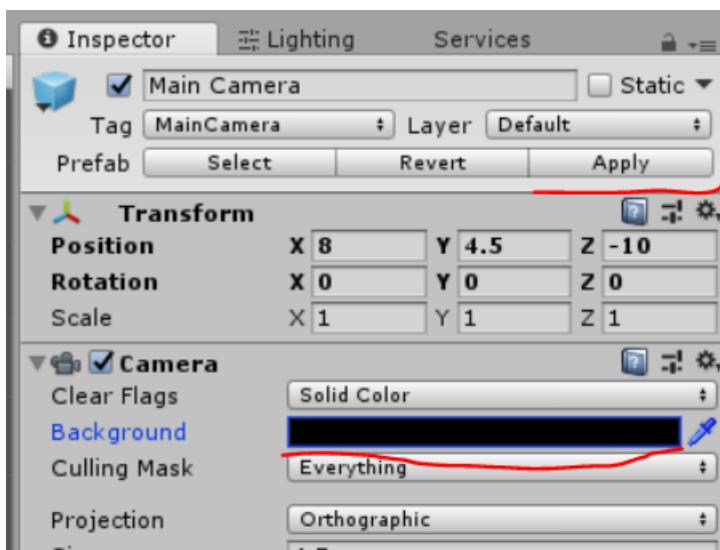
- Na Fase_2 mude o BG para a imagem ***background_2*** ou alguma de sua escolha. Lembre-se de fazer os devidos ajustes de **PPU** e **Pivot**.





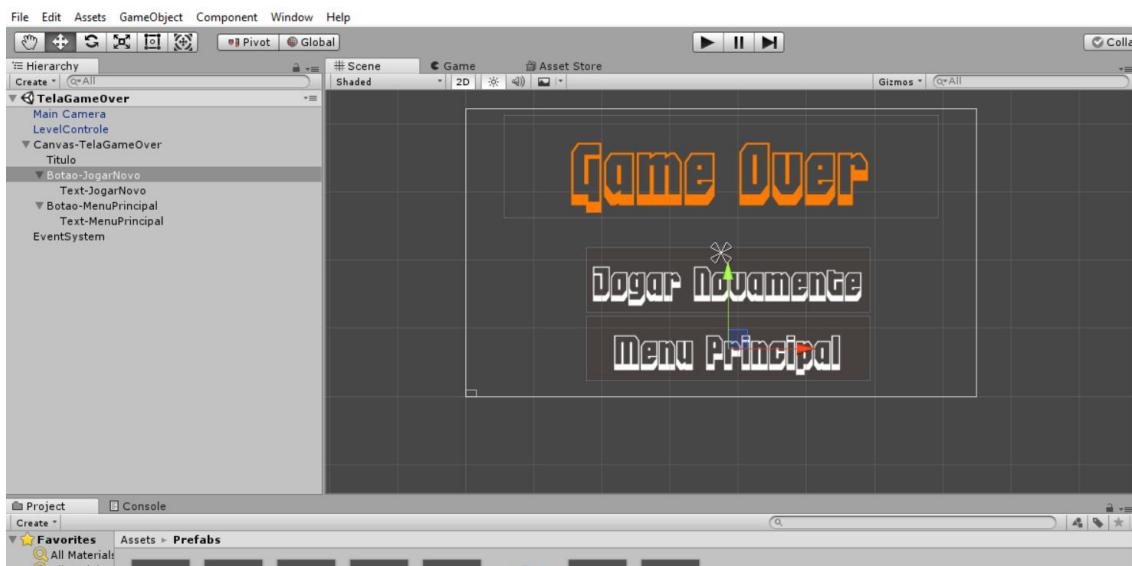
O **background** está com um tom azul, pois a câmera possui uma cor azul. Vamos mudar isso.

- Vá no componente **Main Camera** e mude a cor para preto, ou algo que lhe agrade. Como agora a câmera é **prefabs**, clique em um **apply** para a modificação persistir por todos as **scenes** que usam essa câmera.



9 – Criar a Tela *GameOver* (de verdade)

- Duplique a **scene** TelaVitoria. Renomeie para TelaGameOver
- Faça os ajustes de texto e adicione um botão para decidir entre o MenuPrincipal e Jogar Novamente

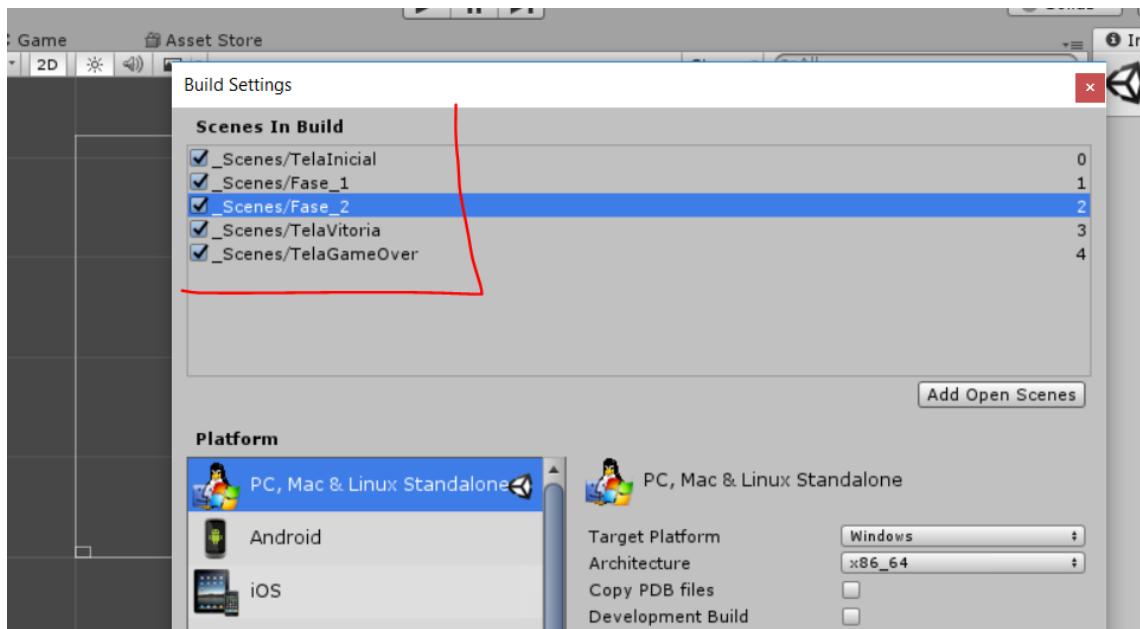


- Atualize o Script GameOverControle para que a TelaGameOver seja chamada.

```
5     public class GameOverControle : MonoBehaviour {
6
7         private LevelControle levelControle;
8
9         private void Start() {
10             levelControle = GameObject.
11                 FindObjectOfType<LevelControle>();
12         }
13
14         private void OnTriggerEnter2D(Collider2D collision) {
15
16             levelControle.CarregaLevel("TelaGameOver");
17
18         }
19 }
```

Vamos adicionar as telas na Build. A ordem será importante.

- Abra a pasta **Scenes** na aba **Projects**
- Vá em **File->Build Settings**.
- Adicione as Scenes seguindo a ordem abaixo



- Teste se a Fase_1 e Fase_2 progridem para a TelaGameOver.
- Teste os botões “Jogar Novamente” e “Menu Principal” da TelaGameOver

Observe que ainda não existe condição de vitória, estamos apenas testando o Game Over.

Vamos criar uma condição falsa de vitória agora, para testar o fluxo de tela. Para isso não é necessário pensar em toda a mecânica do jogo.

- Vá até o *script* LevelControle e crie o método CarregaProxLevel

```

6  public class LevelControle : MonoBehaviour {
7
8      public void CarregaLevel(string levelNome) {
9          SceneManager.LoadScene(levelNome);
10     }
11
12     public void CarregaProxLevel() {
13         SceneManager.LoadScene(SceneManager.
14             GetActiveScene().buildIndex + 1);
15     }
16 }
```

Esse método invoca o próximo ***level*** (ou ***scene***) fazendo o incremento do **Build ID**. Por isso foi importante colocar as Scenes na ordem adequada. Este método incrementa o ID atual para chamar a próxima fase.

- Crie um script chamado Bloco
- Associe aos prefabs Bloco_1, Bloco_2 e Bloco_3

```
5  public class Bloco : MonoBehaviour {  
6      LevelControle levelControle;  
7  
8  
9  [Warning] 9 void Start () {  
10         levelControle = FindObjectOfType<LevelControle>();  
11     }  
12  
13  [Warning] 13 private void OnCollisionEnter2D(Collision2D collision) {  
14         VitoriaFake();  
15     }  
16  
17  [Warning] 17 void VitoriaFake() {  
18         levelControle.CarregaProxLevel();  
19     }  
20
```

Assim que a Bola colidir com algum Bloco, a próxima ***scene*** será carregada.

- Certifique que o script está associado corretamente ao ***prefab Bloco*** e faça o teste.

10 – Destruindo os blocos

Nessa etapa iniciaremos o processo de destruição dos blocos.

- Abra o ***script Bloco***

```
5  public class Bloco : MonoBehaviour {
6
7      [SerializeField]
8      private int maxHits;
9      private int numHits;
10
11     LevelControle levelControle;
12
13     void Start () {
14         numHits = 0;
15         levelControle = FindObjectOfType<LevelControle>();
16     }
17
18     private void OnCollisionEnter2D(Collision2D collision) {
19         numHits++;
20         if(numHits >= maxHits) {
21             Destroy(gameObject);
22         }
23     }
}
```

Na linha 8 temos a variável **numHits** servindo de contador para o número de vezes que a bola acerta esse bloco. E a variável **maxHits** determina o número máximo de acertos que esse bloco suporta antes de quebrar.

Dentro do método **OnCollision**, na linha 18, verificamos se o limiar foi ultrapassado. Se tiver sido, destruímos o bloco.

- Na pasta prefab coloque valores diferentes de **maxHits** para os blocos e teste isso no jogo. No meu exemplo para o Bloco_1 coloquei maxHit = 1. Para o Bloco_2 coloquei 2 no maxHit e 3 para o Bloco_3

10.2

Vamos trocar os **sprites** dos Blocos para darmos ao jogador um retorno do que está ocorrendo.

- Abra o Script **Bloco**.

```

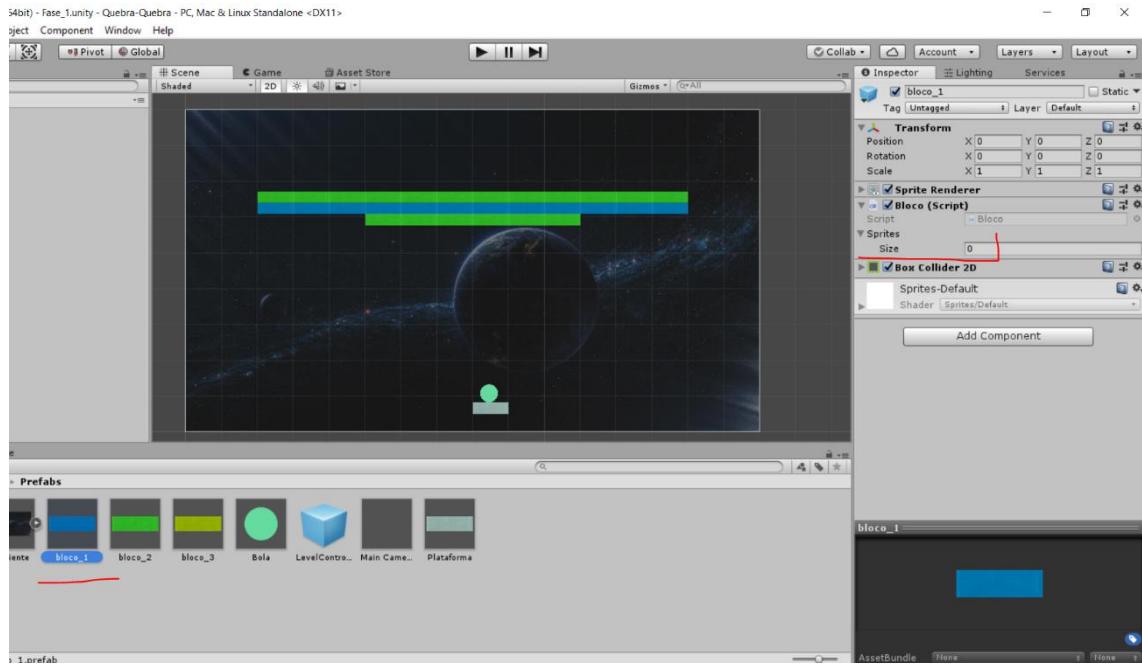
8     [SerializeField]
9     private Sprite[] sprites;
10    private SpriteRenderer spriteRenderer;
11    private int numHits;
12    private LevelControle levelControle;
13
14    void Start () {
15        numHits = 0;
16        levelControle = FindObjectOfType<LevelControle>();
17        spriteRenderer = GetComponent<SpriteRenderer>();
18    }
19
20    private void OnCollisionEnter2D(Collision2D collision) {
21        numHits++;
22        int maxHits = sprites.Length + 1;
23        if(numHits >= maxHits) {
24            Destroy(gameObject);
25        } else {
26            CarregaSprite();
27        }
28    }
29
30    private void CarregaSprite() {
31        int spriteIndex = numHits - 1;
32        if (sprites[spriteIndex]) {
33            spriteRenderer.sprite = sprites[spriteIndex];
34        }

```

Na linha 9 criamos um **array** do tipo **Sprite**, pois usaremos para salvar os sprites dos blocos quebrado. Na linha 11 fazemos acesso ao componente **Sprite Renderer** do bloco. No tratamento de colisão, caso o bloco não seja destruído, chamamos o método **CarregaSprite()**. Este por sua vez é responsável por carregar os sprites que estão no **array**. Para carregar usamos o componente **SpriteRenderer**.

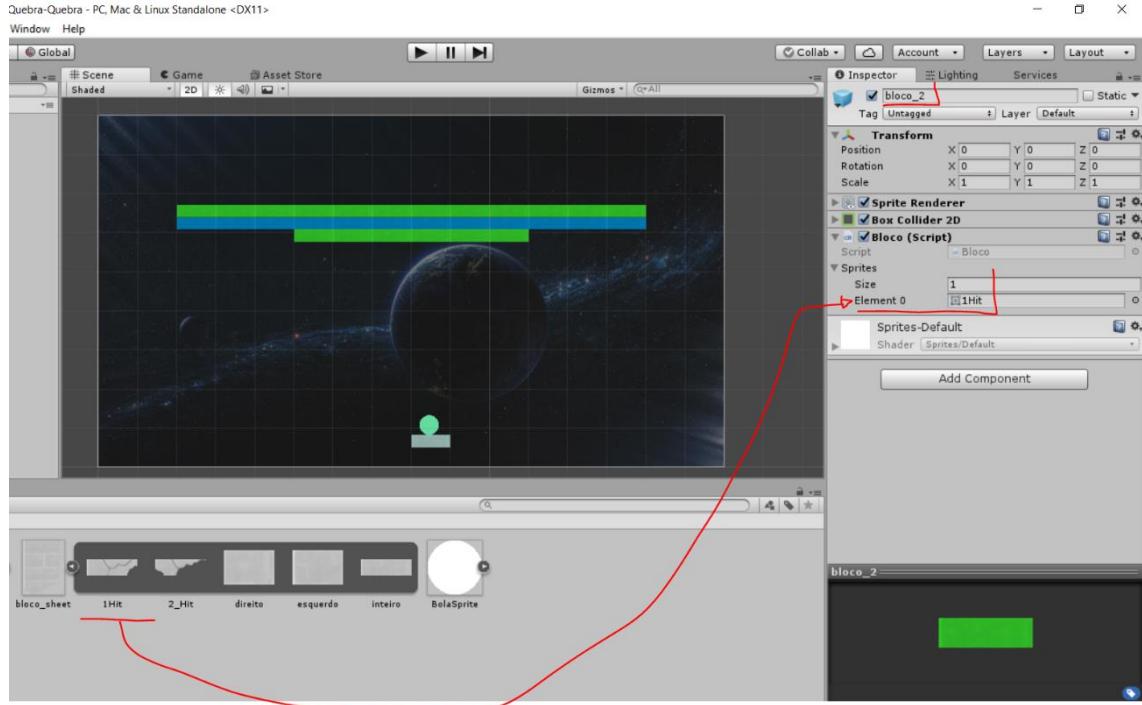
Observe que deixamos de expor a variável **maxHits** no editor, pois agora usamos o tamanho do **array** para inferir o seu valor.

- Va até o Unity. Escolha o bloco_1 na pasta **prefab**

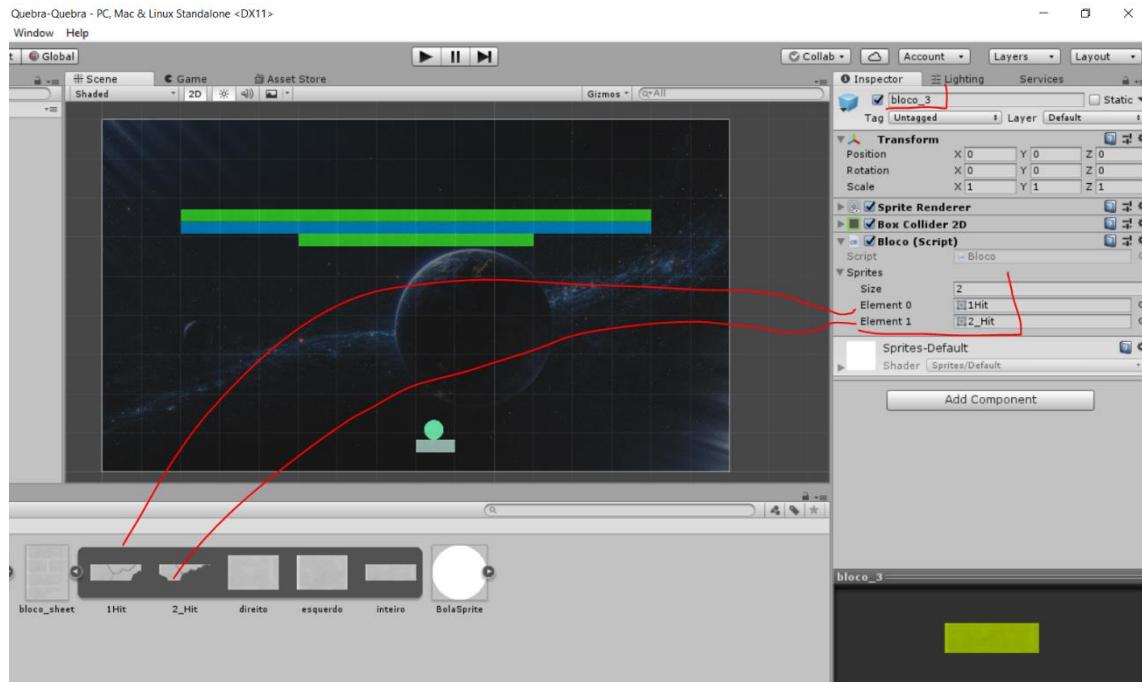


- Esse bloco é destruído com apenas um acerto. Portanto não precisamos alterar o **array**.

- Selecione o **prefab** bloco_2
- Arraste o Sprite “1 Hit” para o campo Element 0.

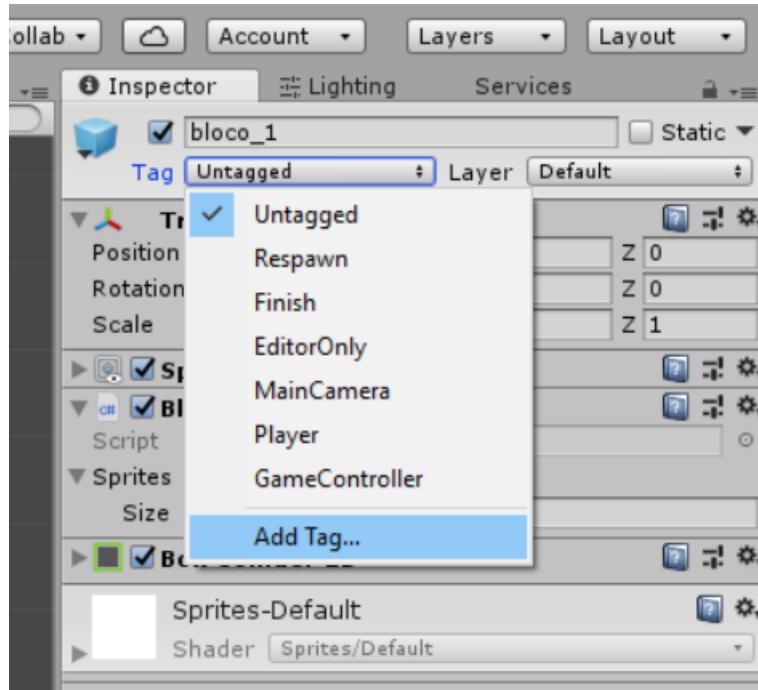


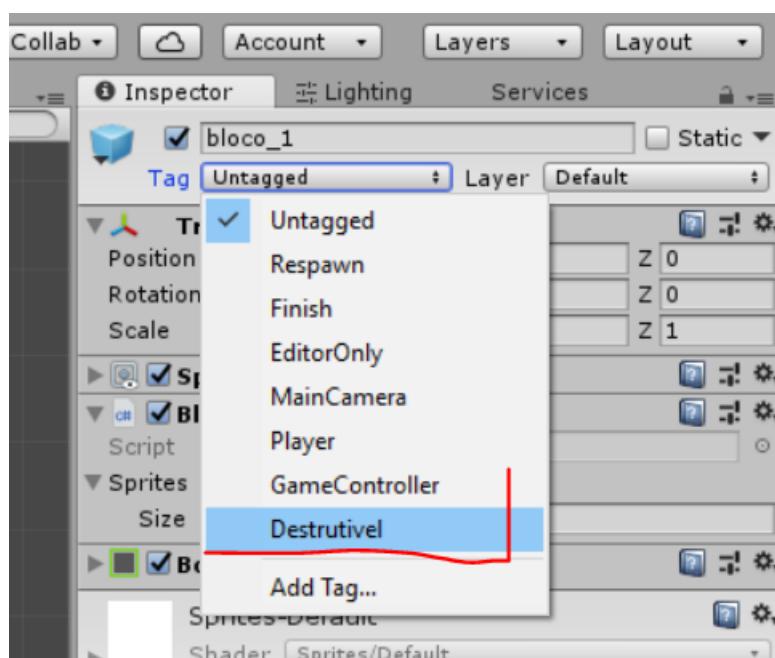
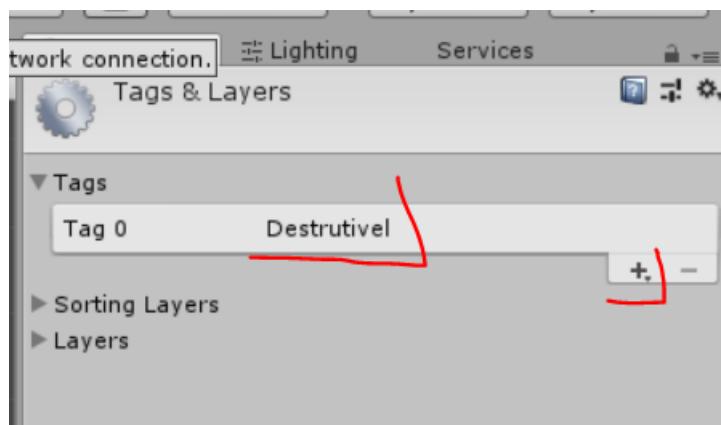
- Repita para o **prefab** bloco_3



Vamos criar um bloco indestrutível

- Crie uma tag “Destruível” para definir se os blocos são destrutíveis. Adicione essa tag nos **prefab** bloco_1, _2 e _3





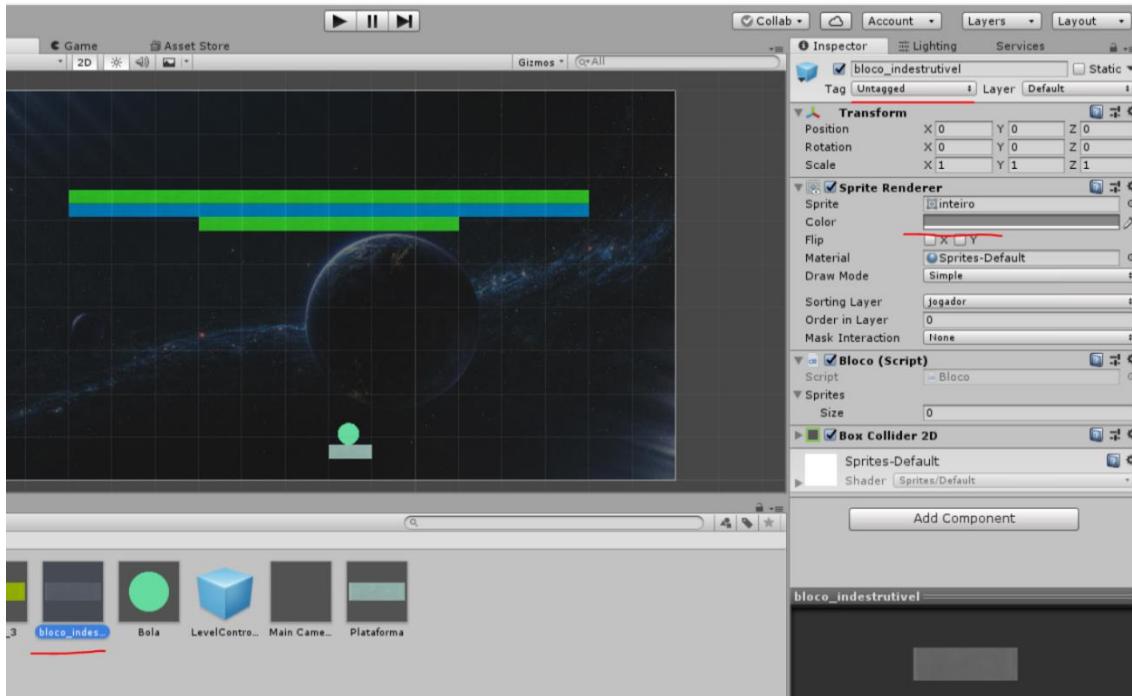
- Adicione a **tag** “Destruivel” nos blocos_2 e blocos_3
- Escolha um desses 3 blocos e duplique-o
- Chame de bloco_indestrutivel e remova a tag “Destruivel”.
- Coloque uma cor de sua preferência
- Faça a seguinte modificação no Script Bloco

```

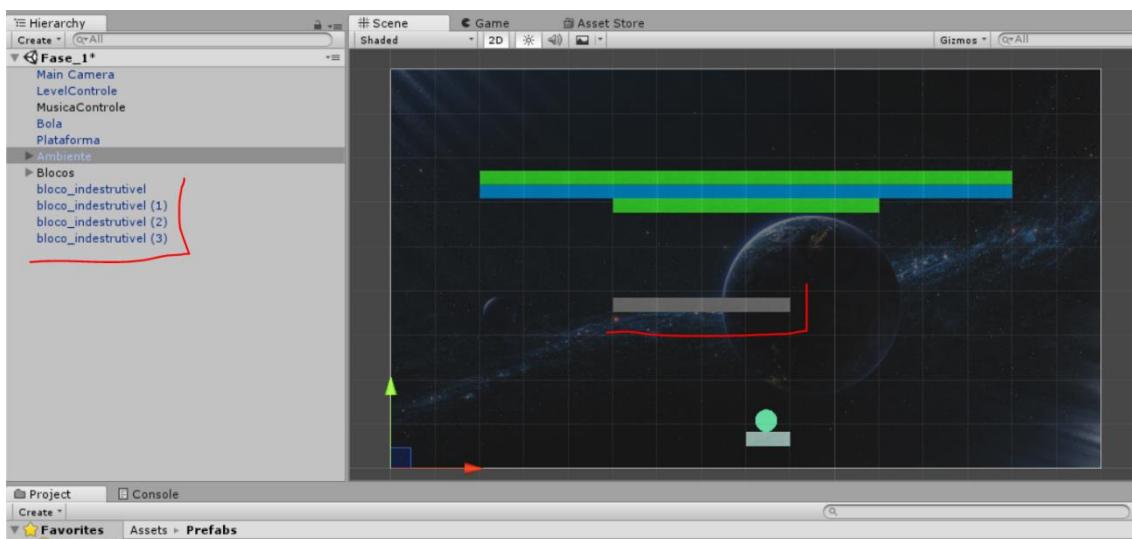
18     private void OnCollisionEnter2D(Collision2D collision) {
19         if (transform.CompareTag("Destruivel")) {
20             TratarDano();
21         }
22     }
23
24     private void TratarDano() {
25         numHits++;
26         int maxHits = sprites.Length + 1;
27         if (numHits >= maxHits) {
28             Destroy(gameObject);
29         } else {
30             CarregaSprite();
31         }
32     }

```

Criamos na linha 26 o método ***TratarDano()*** responsável por destruir e trocar os ***sprites*** dos blocos. E na linha 21 somente tratamos os acertos, mas apenas para blocos marcados como “Destruível”



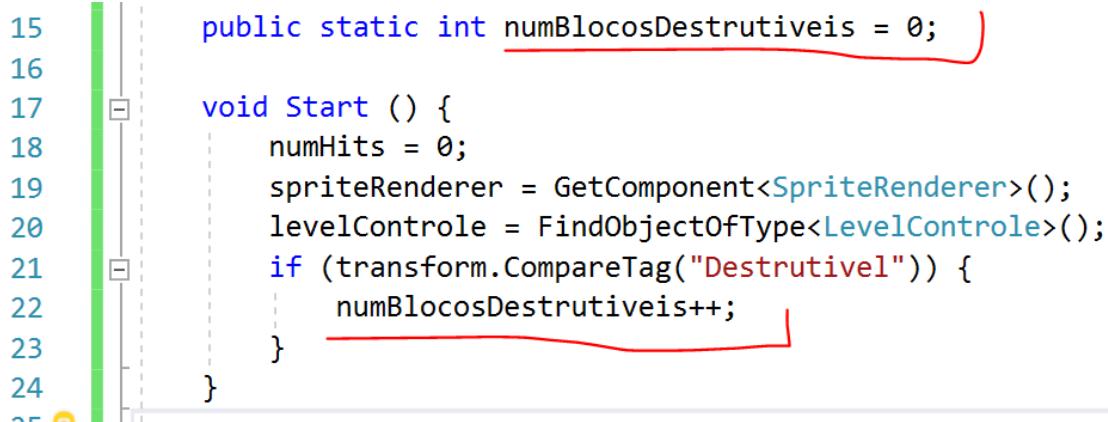
- Faça o teste com o bloco_indestrutivel



Criando uma condição de vitória

A fase é vencida quando não há mais blocos destrutíveis. Então precisamos de um marcador para isso. Vamos usar uma variável estática.

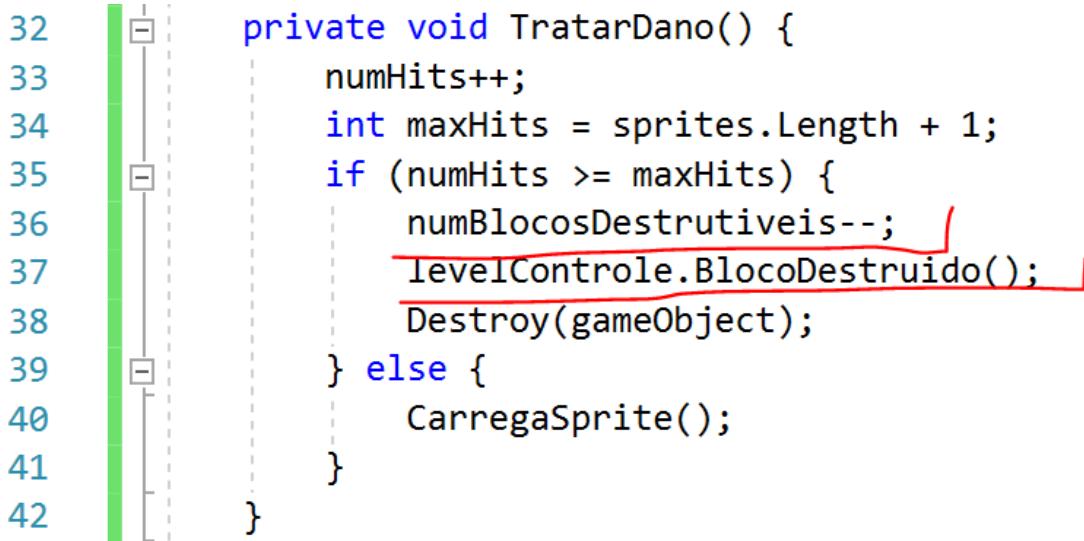
- Va no Script Bloco. Faças as modificações abaixo



```
15     public static int numBlocosDestruiveis = 0;
16
17     void Start () {
18         numHits = 0;
19         spriteRenderer = GetComponent<SpriteRenderer>();
20         levelControle = FindObjectOfType<LevelControle>();
21         if (transform.CompareTag("Destruivel")) {
22             numBlocosDestruiveis++;
23         }
24     }
25 
```

Na linha 15 temos a variável estática que irá controlar o número de blocos destrutíveis. Na linha 22 incrementamos esse número para bloco marcado com a tag “Destruivel”.

- Ainda no Script Bloco



```
32     private void TratarDano() {
33         numHits++;
34         int maxHits = sprites.Length + 1;
35         if (numHits >= maxHits) {
36             numBlocosDestruiveis--;
37             levelControle.BlocoDestruido();
38             Destroy(gameObject);
39         } else {
40             CarregaSprite();
41         }
42     }
43 
```

Na linha 36 decrementamos o número de blocos destrutíveis e na linha 37 chamamos o método BlocoDestruido (ainda não implementamos esse método) da classe **LevelControle**.

- Vá no **script LevelControle**

```

17     public void BlocoDestruido() {
18         if(Bloco.numBlocosDestruiveis <= 0) {
19             CarregaProxLevel();
20         }
21     }

```

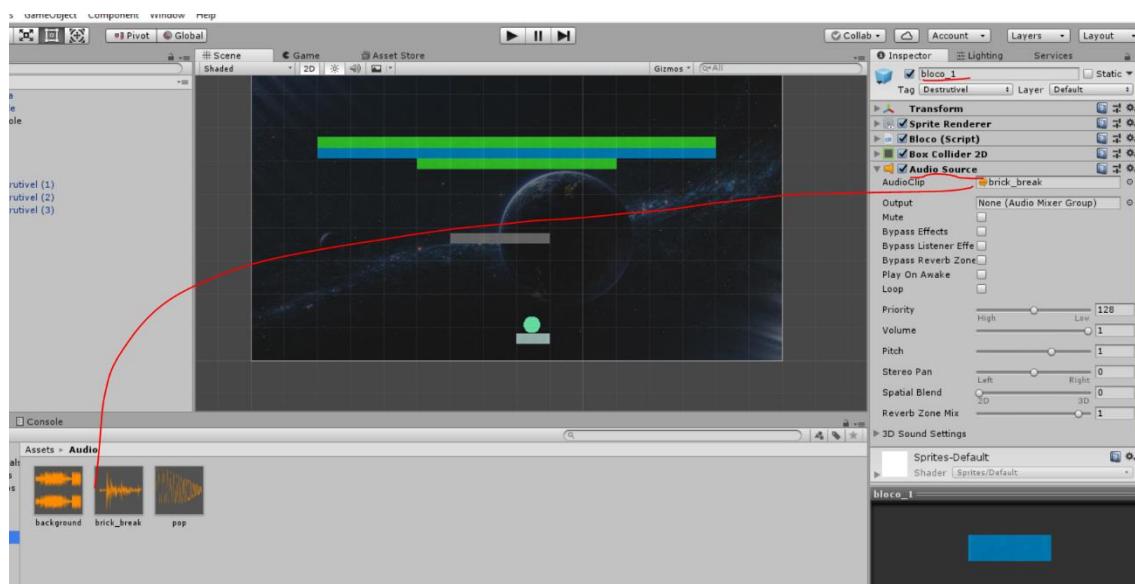
Esse método, **BlocoDestruido**, é chamado toda vez que um bloco é destruído. Ele verifica se o número de blocos destrutíveis na **Scene** é menor que zero, e carrega o próximo **Level** (ou fase, **Scene**....).

- Faça os testes. Verifique se o próximo **Level** é carregado corretamente.

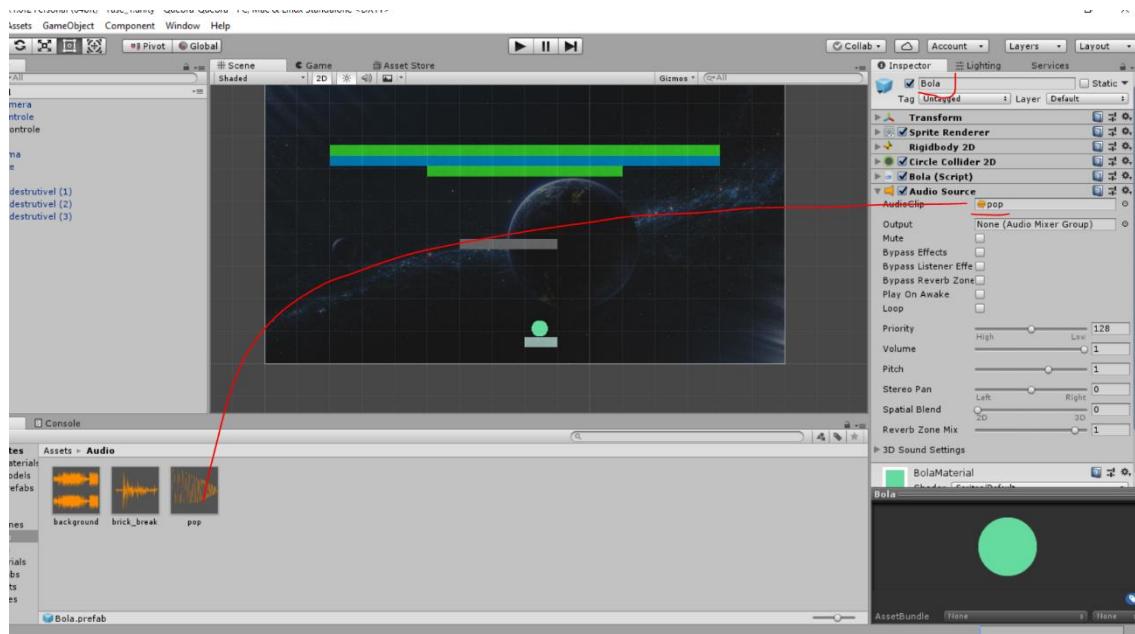
11 – Adicionando som

Vamos adicionar áudio a este jogo. Um som para o bloco se quebrando e outro som para a bola, quando esta colidir com a parede.

- Vá no **prefab** bloco_1. Adicione um **AudioSource**. Coloque o som “brick_break” (que se encontra na pasta Audio) no campo **AudioClip**.



- Repita para os demais blocos (**prefabs**)
- Vá no **prefab** Bola e adicione o áudio “pop”.



Agora vamos configurar os *scripts*

- Vá no **script** Bola e faça as seguintes alterações.

```

5  public class Bola : MonoBehaviour {
6  private Plataforma plataforma;
7  private Vector3 plataformaBolaDis;
8  private Rigidbody2D rb2D;
9  private AudioSource audioSource;
10
11  private bool jogoComecou = false;
12
13
14  void Start() {
15      rb2D = GetComponent<Rigidbody2D>();
16      plataforma = FindObjectOfType<Plataforma>();
17      plataformaBolaDis = transform.position
18          - plataforma.transform.position;
19      audioSource = GetComponent<

```

Primeiramente estamos buscando uma referencia para o Componente

- Permaneça nesse Script.

```

32     private void OnCollisionEnter2D(Collision2D collision) {
33
34         if (!collision.gameObject.CompareTag("Destruivel") && jogoComecou) {
35             audioSource.Play();
36         }
37     }

```

Iremos tocar o áudio apenas se o bloco não for destrutível e se o jogo já tiver começado.

- Agora vá no Script Bloco e faça as modificações abaixo.

```

28     private void OnCollisionEnter2D(Collision2D collision) {
29         if (transform.CompareTag("Destruivel")) {
30             AudioSource.PlayClipAtPoint(audioSource.clip, transform.position);
31             TratarDano();
32         }
33     }

```

Primeiramente, busque uma referência para o componente **. E na linha 30 estamos usando também a classe **AudioSource**. Essa classe possui um método, **PlayClipAtPoint()**, que permite posicionar um áudio em determinada posição da **scene**, tocar e logo em seguida esse áudio é destruído. Isso é importante, pois se usarmos a mesma forma como fiz para a Bola, o bloco será destruído e o áudio interrompido**

- Faça os testes e verifique se o áudio está tocando corretamente.

12 – Ajustando a velocidade

Para evitar que a bola fique presa em um loop, vamos adicionar um valor aleatório a cada colisão.

- Abra o script Bola e faça as modificações na linha 39 e 40, conforme mostrado abaixo.

```

36     private void OnCollisionEnter2D(Collision2D collision) {
37
38         if (!collision.gameObject.CompareTag("Destruivel") && jogoComecou) {
39             Vector2 velocidadeAjuste =
40                 new Vector2(Random.Range(0f, 0.2f), Random.Range(0f, 0.2f));
41             rb2D.velocity += velocidadeAjuste;
42             audioSource.Play();
43         }
44     }

```

13 – Playteste automático

Nessa etapa vamos inserir um modelo de playteste automático. Vamos movimentar a plataforma automaticamente com a bola. Assim, podemos fazer alguns testes de forma menos entediante.

- Abra o **script Plataforma**. Adicione uma referencia para Bola, e uma variável booleana para decidir se o jogo será manual ou automático.

```
5  public class Plataforma : MonoBehaviour {
6
7      [SerializeField]
8      private bool autoPlay = false;
9
10     private Bola bola;
11
12     void Start() {
13         bola = FindObjectOfType<Bola>();
14     }
15 }
```

- Faça um refatoração dentro do método **Update()**. Faça um método para movimento com o mouse (manual) ou movimento automático.

```
17     void Update () {
18         if (!autoPlay) {
19             MovimentoMouse();
20         } else {
21             MovimentoAutomatico();
22         }
23     }
24
25     private void MovimentoMouse() {
26         float mousePosWorldUnitX =
27             ((Input.mousePosition.x) / Screen.width * 16);
28         Vector2 plataformaPos =
29             new Vector2(0, transform.position.y);
30         plataformaPos.x = Mathf.Clamp(mousePosWorldUnitX, 0f, 15f);
31         transform.position = plataformaPos;
32     }
33 }
```

A diferença está em como obtemos a coordenada X da plataforma. No modo manual usamos o mouse, e no modo automático usamos a própria posição da bola. Para isso fazemos acesso ao componente **transform** do GO **Bola**.

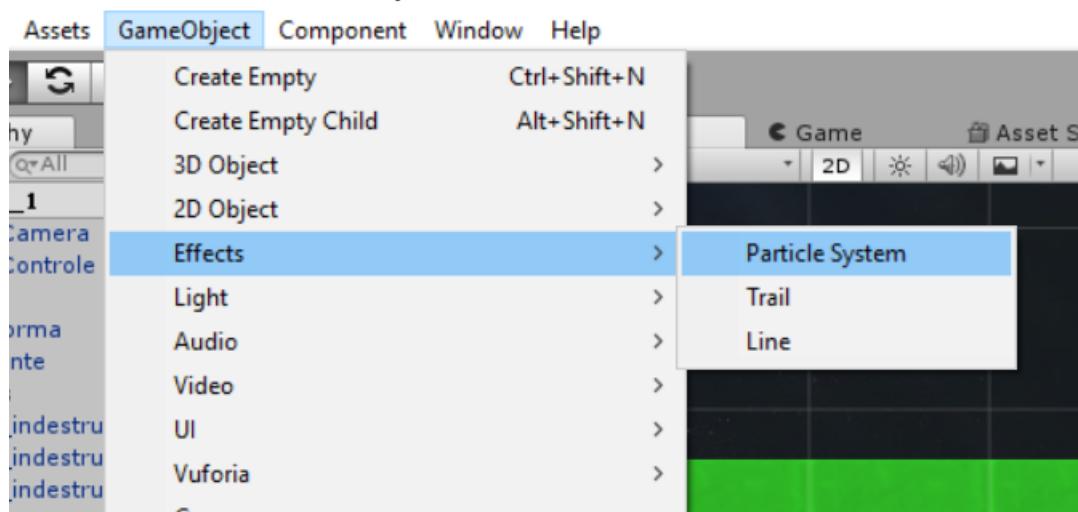
```
34     private void MovimentoAutomatico() {  
35         float bolaPosX =  
36             bola.transform.position.x;  
37         Vector2 plataformaPos =  
38             new Vector2(0, transform.position.y);  
39         plataformaPos.x = Mathf.Clamp(bolaPosX, 0f, 15f);  
40         transform.position = plataformaPos;  
41     }  
42 }
```

14 – Adicionando Efeitos de Explosão

Vamos usar o sistema de partículas do Unity para criar efeitos de explosão.

- Crie um **Particle System**. Chame de **explosaoEfeito**

18.1.0f2 Personal (64bit) - Fase_1.unity - Quebra-Quebra - PC, Mac & Linux Standalone <DX11>



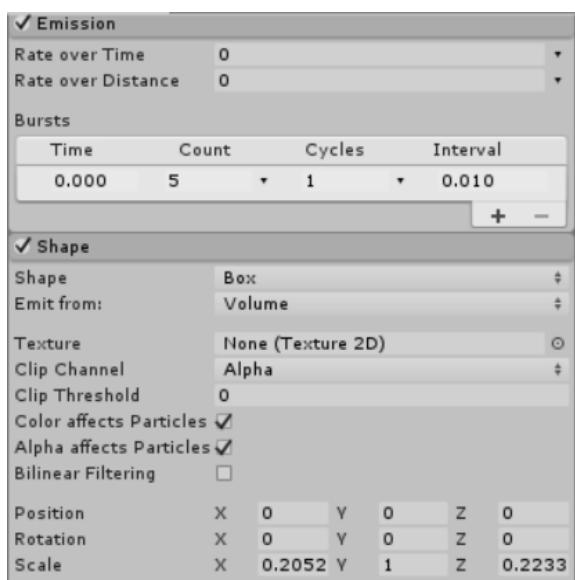
- Use as configurações abaixo:





É muito importante que você “brinque” com as configurações para criar a explosão a seu critério.

- Repita esse processo para criar uma explosão menor. Chame de **explosaoEfeitoIni**



O objetivo é usar a **explosaoEfeito** antes do bloco ser destruído. E a **explosaoEfeitoIni** é usada a cada colisão anterior.

- Vá até o **script Bloco**.

```
8 [SerializeField]
9 private Sprite[] sprites;
10 [SerializeField]
11 private ParticleSystem explosionIniFX;
12 [SerializeField]
13 private ParticleSystem explosionFimFX;
--
```

Colocamos duas referências, uma para cada tipo de explosão.

- Continue no **script Bloco**

```
40 private void TratarDano() {
41     numHits++;
42     int maxHits = sprites.Length + 1;
43     if (numHits >= maxHits) {
44         numBlocosDestruiveis--;
45         levelControle.BlocoDestruido();
46         EfeitoExplosao(explosionFimFX);
47         Destroy(gameObject);
48     } else {
49         EfeitoExplosao(explosionIniFX);
50         CarregaSprite();
51     }
52 }
```

Observe que chamamos uma função **EfeitoExplosão()** e passamos como parâmetro qual a explosão que iremos criar. Veja que antes do objeto ser destruído passamos como a parâmetro a **explosionFimFX**.

```
54     private void EfeitoExplosao(ParticleSystem expPS) {  
55         ParticleSystem explosionEffect =  
56             Instantiate<ParticleSystem>(expPS,  
57                 transform.position, Quaternion.identity);  
58         ParticleSystem.MainModule expMain = explosionEffect.main;  
59         expMain.startColor = transform.  
60             GetComponent<SpriteRenderer>().color;  
61     }  
62 }
```

A primeira que fazemos é instanciar o efeito (linha 55). Depois fazemos acesso ao módulo **Main** do sistema de partículas, linha 58. Tendo acesso a esse modo podemos mudar a cor das partículas. Na linha 59 deixamos as partículas com a mesma cor que o bloco. Para pegar a cor dos blocos, acessamos o componente **SpriteRenderer()** e depois pegamos a **color**.