

Desenvolvimento com Motores de Jogos II

Aula 07 - Jogo Polygonal Rescue - Parte 5 -
Gizmos Personalizados

Apresentação

Olá, pessoal! Sejam bem-vindos à aula 7 da disciplina de Motores de Jogos II. Na aula passada aprendemos a criar algumas fases para nosso jogo e também objetos a serem reutilizados entre as fases (cenas), chamados de Prefabs. Hoje aprenderemos a criar Gizmos personalizados no editor, pois isso ajuda a exibir as configurações dos nossos scripts de maneira visual dentro do Unity. Também vamos integrar no jogo o sistema de instanciar Prefabs, realizado na aula passada. Assim, deixaremos o nosso jogo ainda mais completo e divertido. Vamos lá?

Objetivos

Ao final desta aula, você deverá ser capaz de:

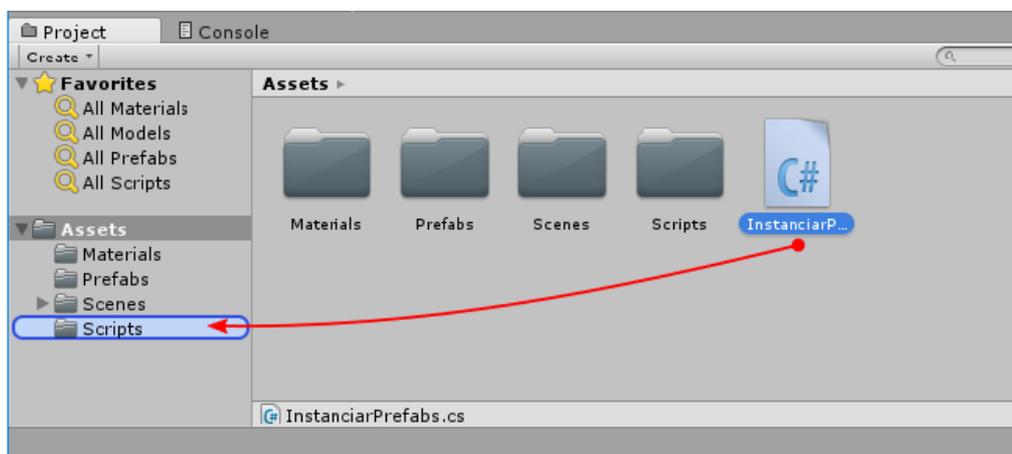
- Aprender a criar Gizmos personalizados;
- Integrar o sistema de instanciar Prefabs no jogo;
- Adicionar ícones em GameObjects.

Analizando o script InstanciarPrefabs

Na aula passada criamos um script chamado “InstanciarPrefabs” que serve para instanciar uma quantidade de Prefabs específicos, configurados em uma lista, dentro de determinada área cujo centro é o mesmo do objeto ao qual o script está anexado e se estende em uma distância específica.

Primeiramente vamos abrir nosso projeto *Polygonal Rescue* e mover o script InstanciarPrefab.cs para a pasta “Scripts” (se não já tiver feito isso). Veja esse processo na **Figura 1**.

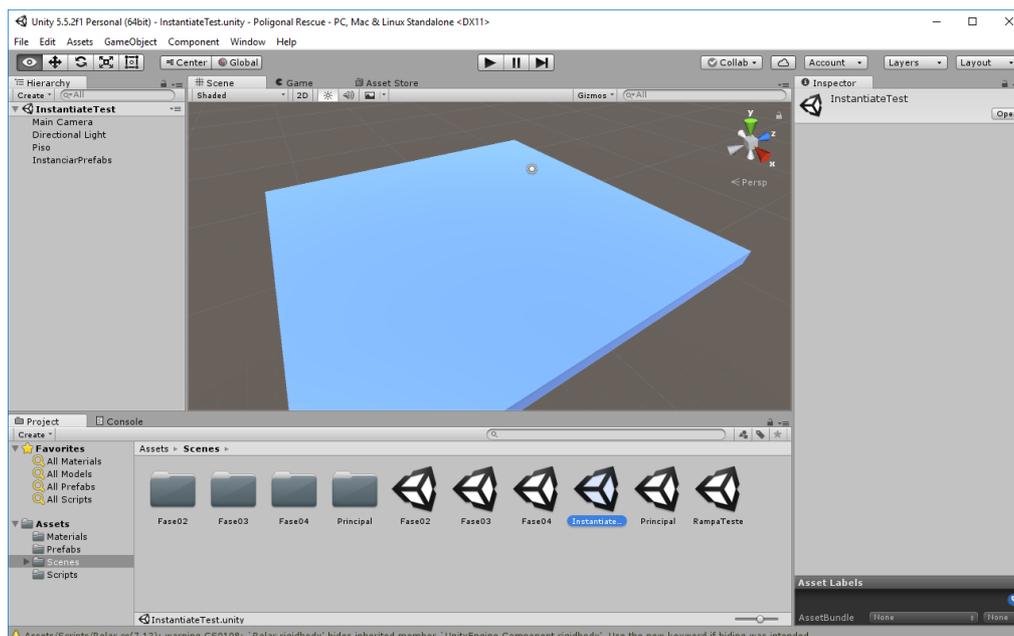
Figura 01 - Movendo InstanciarPrefabs.cs para a pasta Scripts.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Em seguida abra a cena (Que deve estar na pasta Scenes) InstantiateTest para ser exibida o Scene View, como na **Figura 2**.

Figura 02 - Cena InstantiateTest no Scene View.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora, localize o script InstanciarPrefabs.cs na pasta Scripts e o edite no Monodevelop. O seu conteúdo deve estar atualmente como na **Figura 3**.

Figura 03 - Script InstanciarPrefabs atual.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class InstanciarPrefabs : MonoBehaviour {
6     public List<GameObject> prefabs_base; // Lista dos prefabs base para a criação das instâncias
7     public float distancia_maxima; // Distância máxima das instâncias até o centro desse GameObject
8     public int quantidade; // Quantidade de instâncias que serão criadas.
9
10    void Awake() {
11        // posição atual do GameObject que contém esse script
12        Vector3 centro = transform.position;
13
14        // laço com "quantidade" iterações (uma por instância)
15        for (int i = 0; i < quantidade; i++) {
16            // Escolhemos um índice aleatório para selecionar o prefab base que será usado.
17            // Esse índice deve ir de zero até prefabs_base.Count, que é o tamanho da lista de prefabs.
18            int indice_aleatorio = Random.Range (0, prefabs_base.Count);
19            GameObject prefab_escolhido = prefabs_base [indice_aleatorio];
20
21            Vector3 posicao_da_instancia = new Vector3 ();
22            // Valor de X variando em + ou - a distancia máxima tendo centro.x como base
23            posicao_da_instancia.x = Random.Range ((centro.x - distancia_maxima), (centro.x + distancia_maxima));
24            // Valor de Y igual ao centro.y já que não iremos variar a posição das instancias verticalmente.
25            posicao_da_instancia.y = centro.y;
26            // Valor de Z variando em + ou - a distancia máxima tendo centro.Z como base
27            posicao_da_instancia.z = Random.Range ((centro.z - distancia_maxima), (centro.z + distancia_maxima));
28
29            // Instanciando o prefab_escolhido na posicao_da_instancia com rotação padrão (Quaternion.identity)
30            Instantiate (prefab_escolhido, posicao_da_instancia, Quaternion.identity);
31        }
32    }
33 }
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

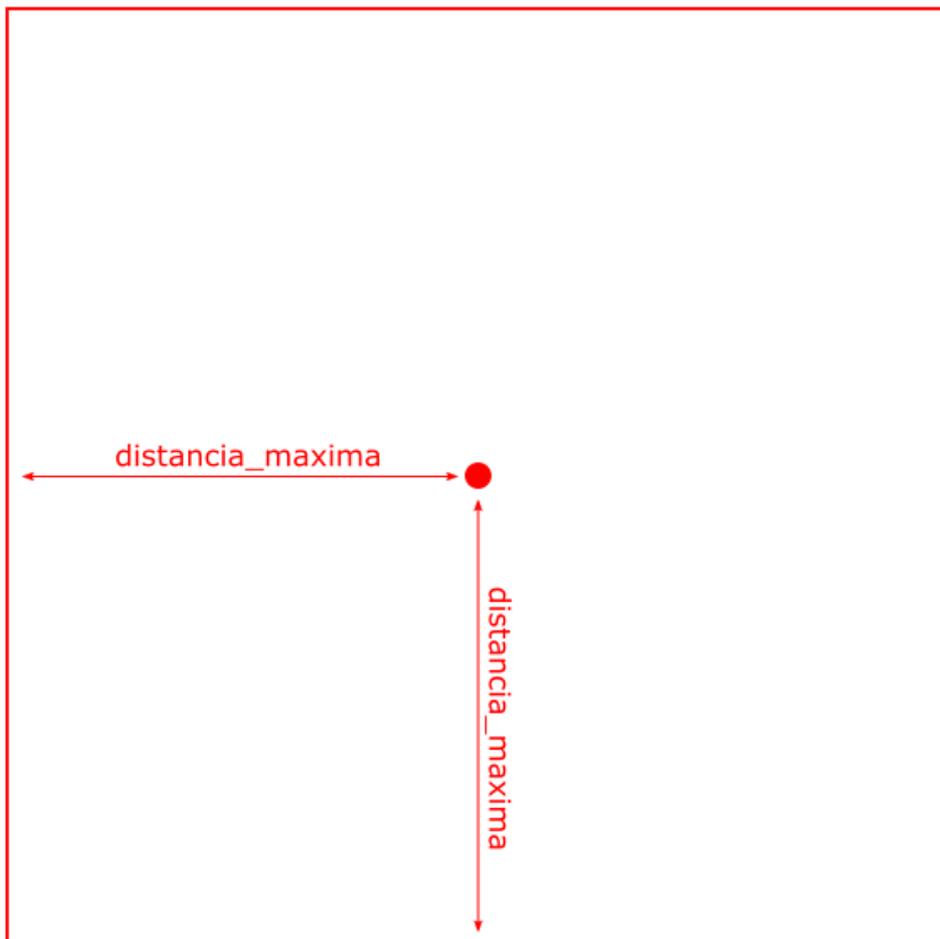
Revisando o script, repare que temos três variáveis públicas:

- List<GameObject> **prefabs_base**: utilizada para armazenar uma lista de Prefabs que podem ser instanciados pelo script.
- float **distancia_maxima**: indica qual a máxima distância dos eixos X e Z do centro que os prefabs podem ser instanciados.
- Int **quantidade**: a quantidade total de Prefabs que serão instanciados.

Nesse script (executado antes do Start), o que fazemos no método Awake é um laço *for* para que uma **quantidade** de prefabs da lista **prefabs_base** seja criada em posições aleatórias. Cada prefab instanciado é criado com a coordenada Y (altura) igual ao do gameobject que o script está adicionado enquanto as coordenadas X e Z são configuradas com uma variação de + ou - **distancia_maxima**. Isso faz com que os prefabs fiquem na mesma altura do objeto com o script, entretanto suas posições no plano X,Z são aleatórias podendo no se distanciar do centro no máximo com o valor de **distancia_maxima**.

Vale observar que essa distância máxima é aplicada individualmente entre as coordenadas X e Y, e se você analisar com cuidado descobrirá que se a posição final do prefab gerado for próximo ao máximo possível em X e Y ele ficará em um local com uma distância linear ao centro maior que a distancia_maxima. Isso foi feito de forma proposital, pois assim a área possível de geração dos prefabs é um quadrado e não um círculo. A **Figura 4** ilustra essa área detalhadamente.

Figura 04 - Área de possível criação de prefabs do script InstanciarPrefabs.



Fonte: Elaborado pelo autor (2017)

A seguir vamos fazer duas mudanças nesse script antes de usá-lo em nosso jogo:

1. Permitir que a área de criação dos prefabs seja um retângulo de qualquer dimensão no eixo X ou eixo Y, pois atualmente é um quadrado de largura e altura idênticas (2 vezes a variável `distancia_maxima`)
2. Criar um método para desenhar um Gizmo personalizado de modo que essa área seja exibida no editor do Unity, isso facilitará bastante a sua configuração.

Adaptando o InstanciarPrefabs para uma Área Retangular

Para adaptar o nosso InstanciarPrefabs.cs para que a área de geração seja retangular, precisamos inicialmente trocar a variável float distancia_maxima por duas outras:

- public float **largura**: Largura do retângulo (eixo X).
- public float **comprimento**: Comprimento do retângulo (eixo Y).

Assim poderemos configurar essas duas variáveis separadamente. Para isso realize os seguintes passos:

1. Remova a variável distancia_maxima;
2. Adicione as variáveis largura e comprimento (ambas do tipo float);

A declaração de suas variáveis deve ficar como na **Figura 5**.

Figura 05 - Novas variáveis do script InstanciarPrefabs.

```
public class InstanciarPrefabs : MonoBehaviour {  
    public List<GameObject> prefabs_base; // Lista dos prefabs base para a criação das instâncias  
    public float largura; // Largura do retângulo (eixo X)  
    public float comprimento; // Comprimento do retângulo (eixo Y)  
    public int quantidade; // Quantidade de instâncias que serão criadas.
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora vamos modificar a nossa função Awake para usar a largura e o comprimento no lugar de distancia_maxima. Não é algo muito difícil, pois o código está bem organizado. Repare que distancia_maxima é atualmente utilizada para a definição da posição da instância do prefab a ser criado (ver **Figura 6**).

Figura 06 - Uso atual da variável distancia_maxima para definir a posição da instância do prefab.

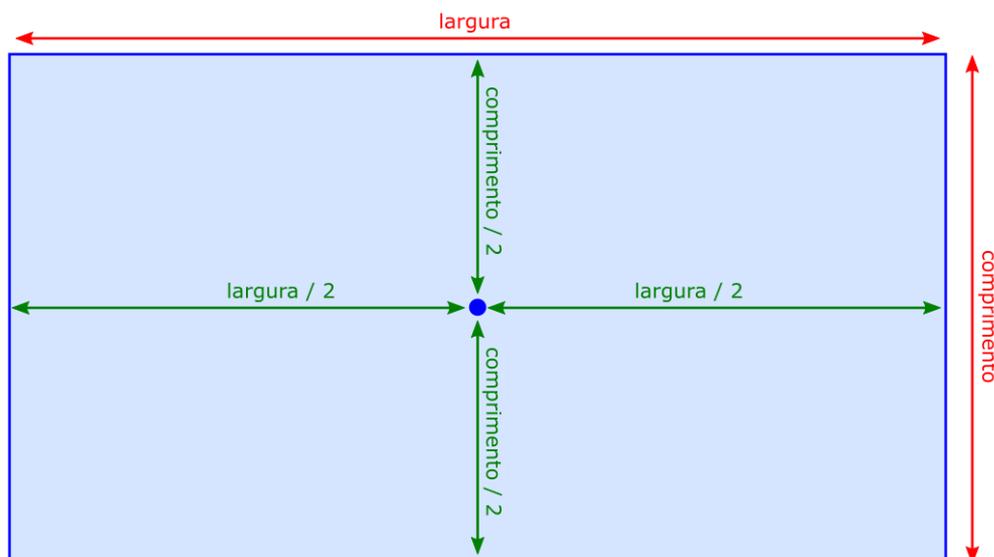
```
Vector3 posicao_da_instancia = new Vector3 ();  
// Valor de X variando em + ou - a distancia máxima tendo centro.x como base  
posicao_da_instancia.x = Random.Range ((centro.x - distancia_maxima), (centro.x + distancia_maxima));  
// Valor de Y igual ao centro.y já que não iremos variar a posição das instancias verticalmente.  
posicao_da_instancia.y = centro.y;  
// Valor de Z variando em + ou - a distancia máxima tendo centro.z como base  
posicao_da_instancia.z = Random.Range ((centro.z - distancia_maxima), (centro.z + distancia_maxima));
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Repare na **Figura 6** que a variável `distancia_maxima` está destacada pelo Monodevelop. Isso ocorre pelo fato de que ela não existe mais, já que a removemos.

Trocamos as duas ocorrências de `distancia_maxima` por $(largura/2)$ na primeira linha em que ela aparece. Por que dividir largura por 2? Isso se dá pelo fato de a posição X variar do centro da área + para um lado e - para o outro, ou seja, para cada um desses lados precisamos variar a metade da largura. A mesma coisa para a variável comprimento e o eixo Y. A **Figura 7** mostra em azul a área de geração retangular que estamos querendo criar, assim como a representação das variáveis largura e comprimento em vermelho e a metade do valor dessas variáveis de cada lado do centro em verde.

Figura 07 - Área retangular de geração dos prefabs com as variáveis largura e comprimento.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Para utilizar a variável largura modifique a linha:

```
1 posicao_da_instancia.x = Random.Range ((centro.x - distancia_maxima), (centro.x + distancia_maxima));
```

Para que fique assim:

```
1 posicao_da_instancia.x = Random.Range ((centro.x - largura/2), (centro.x + largura/2));
```

Para utilizar a variável comprimento, modifique a linha:

```
1 posicao_da_instancia.z = Random.Range ((centro.z - distancia_maxima), (centro.z + distancia_maxima));
```

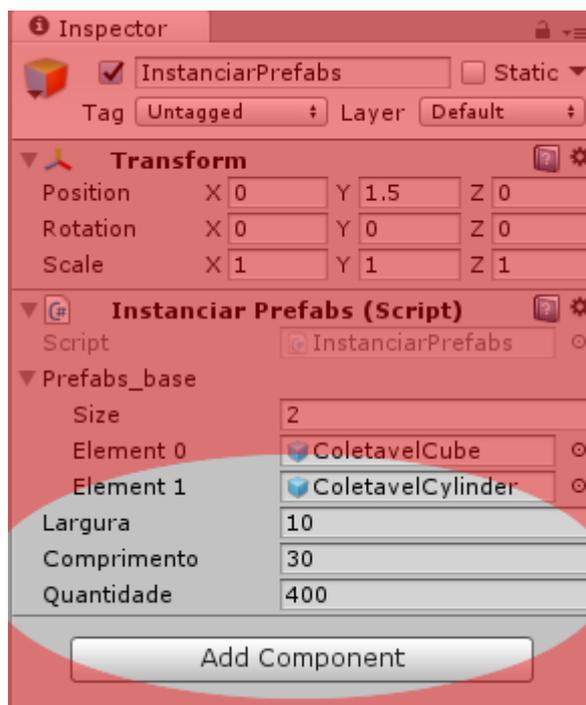
Para que fique assim:

```
1 posicao_da_instancia.z = Random.Range ((centro.z - comprimento/2), (centro.z + comprimento/2));
```

E aí, tudo bem até aqui? É basicamente essa a única mudança no script para que ele instancie prefabs em uma área retangular. Lembre-se que qualquer dificuldade ou questionamentos podem ser esclarecidos nos encontros presenciais ou através da criação de tópicos no Moodle para discutir sobre o assunto.

Salve seu script e volte ao Unity. Clique no GameObject InstanciarPrefabs no Hierarchy e observe no Inspector que o script anexado "Instanciar Prefabs (Script)" não tem mais a variável distancia_maxima, agora em seu lugar existem dois valores a serem preenchidos: Largura e Comprimento, assim como planejamos. Esses valores estão com o valor padrão zero, então modifique-os para Largura=10 e comprimento=30, aproveite e modifique o valor de Quantidade para 400. Usaremos uma quantidade alta de prefabs (400) para tentar preencher bastante a área de geração, pois isso facilita verificar se está funcionando. Veja na **Figura 8** como ficou.

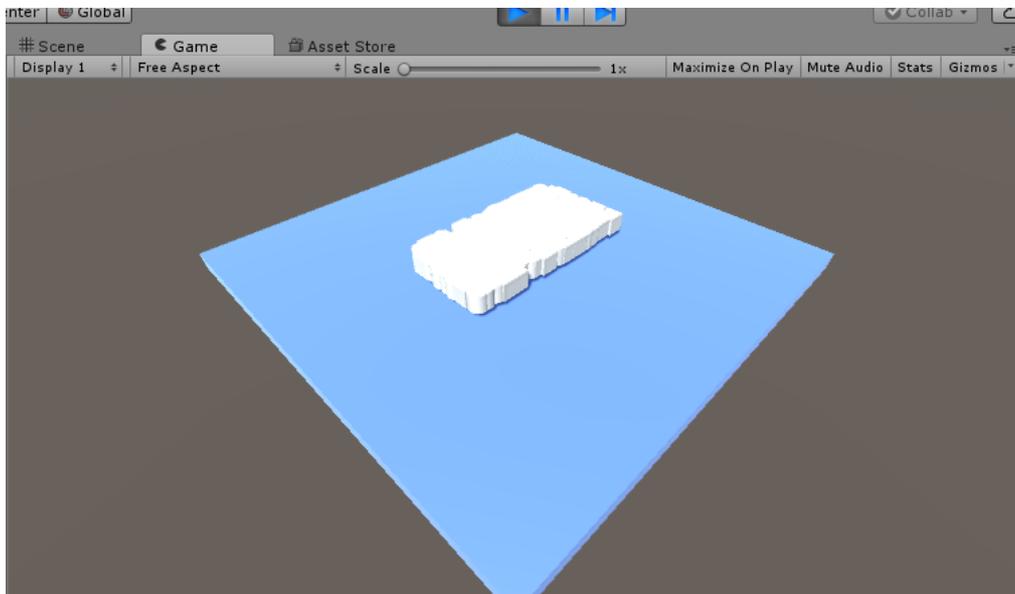
Figura 08 - Novas propriedades do script InstanciarPrefabs anexado no GameObject.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora é com você! Execute o seu jogo e veja como ele ficou! A **Figura 9** nos dá o exemplo aproximado de como ele deve ficar!

Figura 09 - Cena InstantiateTest rodando exibindo prefabs 400 criados em uma área retangular.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Bom Trabalho! Veja que funcionou! Podemos agora determinar a área de geração dos prefabs do nosso script com qualquer largura e comprimento, sem ficarmos presos a um quadrado.

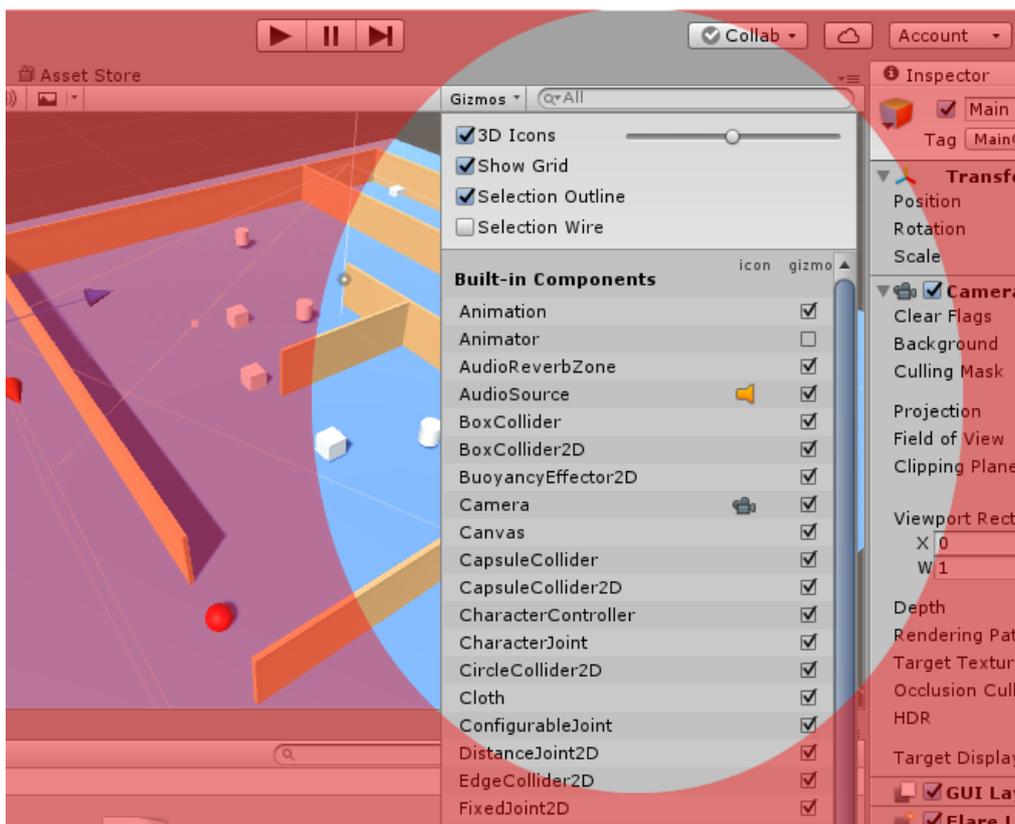
Entretanto, as coisas ainda não estão perfeitas! Repare que a única forma atual de verificar a área coberta pelo InstanciarPrefabs é aumentando o valor quantidade para um número absurdo e executando o jogo. Não seria ideal se houvesse uma forma de visualizar os limites do retângulo no editor sem precisar executar o jogo? Pois existe e é justamente utilizando Gizmos que será possível desenhar esses limites.

Gizmos

Gizmos são elementos utilizados para permitir uma depuração (Debug) visual do seu jogo ou realizar configurações de forma gráfica na scene view. Os Gizmos são por padrão exibidos somente no editor e não quando o seu jogo está rodando, seja no Game View ou em algum dispositivo.

Já existem Gizmos prontos no Unity usados, por exemplo, para mostrar os manipuladores da translate, rotate, scale, e também exibir ícones no Scene View para a câmera, contorno de objetos selecionados, etc. Você pode escolher quais Gizmos são exibidos na Scene View clicando no botão “Gizmos” no canto superior direito dessa janela e marcando os que deseja exibir, veja na **Figura 10**.

Figura 10 - Gizmos padrão do Unity.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora, assistiremos a um vídeo que mostra mais detalhes de como são exibidos os Gizmos no Unity, quais seus principais tipos e como fazemos para habilitar ou desabilitar sua exibição na Scene View.



Vídeo 01 - Gizmos

Gizmos Personalizados

Além dos Gizmos presentes no Unity, é possível criar Gizmos personalizados através de métodos nos seus scripts, permitindo uma melhor experiência quando utilizá-los nos objetos da cena.

O uso de Gizmos é muito útil para vários casos. Por meio da utilização de Gizmos você poderá instruir o Unity a desenhar uma esfera (dentre outros tipos de desenhos) com o tamanho desejado, assim aparecerá exatamente a área de detecção enquanto cria o seu jogo no editor, além disso quando você executar o jogo essa esfera não aparecerá.

Nosso objetivo é utilizar o Gizmo para desenhar a área retangular de geração dos prefabs do script InstanciarPrefabs no editor para que você possa ter uma representação visual dessa área sem a necessidade de executar o jogo.

Ainda com a cena InstantiateTest aberta, edite o script InstanciarPrefabs.cs no Monodevelop. Atualmente ele está conforme a **Figura 11**.

Figura 11 - Script InstanciarPrefabs atual, com a área de geração retangular implementada.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class InstanciarPrefabs : MonoBehaviour {
6     public List<GameObject> prefabs_base; // Lista dos prefabs base para a criação das instâncias
7     public float largura; // Largura do retângulo (eixo X)
8     public float comprimento; // Comprimento do retângulo (eixo Y)
9     public int quantidade; // Quantidade de instâncias que serão criadas.
10
11     void Awake() {
12         // posição atual do GameObject que contém esse script
13         Vector3 centro = transform.position;
14
15         // laço com "quantidade" iterações (uma por instância)
16         for (int i = 0; i < quantidade; i++) {
17             // Escolhemos um índice aleatório para selecionar o prefab base que será usado.
18             // Esse índice deve ir de zero até prefabs_base.Count, que é o tamanho da lista de prefabs.
19             int indice_aleatorio = Random.Range(0, prefabs_base.Count);
20             GameObject prefab_escolhido = prefabs_base[indice_aleatorio];
21
22             Vector3 posicao_da_instancia = new Vector3 ();
23             // Valor de X variando em + ou - a distância máxima tendo centro.x como base
24             posicao_da_instancia.x = Random.Range((centro.x - largura/2), (centro.x + largura/2));
25             // Valor de Y igual ao centro.y já que não iremos variar a posição das instancias verticalmente.
26             posicao_da_instancia.y = centro.y;
27             // Valor de Z variando em + ou - a distância máxima tendo centro.z como base
28             posicao_da_instancia.z = Random.Range((centro.z - comprimento/2), (centro.z + comprimento/2));
29
30             // Instanciando o prefab_escolhido na posicao_da_instancia com rotação padrão (Quaternion.identity)
31             Instantiate (prefab_escolhido, posicao_da_instancia, Quaternion.identity);
32         }
33     }
34 }
35
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Portanto, já estamos utilizando as variáveis **largura** e **comprimento**, porém somente no método Awake que são executadas no início da execução do jogo.

Agora criaremos um novo método chamado “OnDrawGizmos”, executado apenas quando estamos no editor do Unity.

Você deve implementar o OnDrawGizmos em um script sempre que desejar desenhar Gizmos, os quais podem inclusive ser manipulados com o mouse. Nesta aula utilizaremos Gizmos para desenhar quatro linhas, as quais correspondem às bordas do retângulo representando a área de geração dos prefabs. É importante observar que essa função não é executada se o script ao qual essa área pertence não está expandido no Inspector.

O esqueleto padrão do método OnDrawGizmos é assim:

```
1 void OnDrawGizmos() {
2     // desenho dos Gizmos aqui...
3 }
```

O funcionamento dos Gizmos segue o seguinte padrão: inicialmente você modifica a cor dos Gizmos e em seguida utiliza os comandos para desenhá-los na tela com a cor selecionada. Se deseja desenhar vários Gizmos com muitas cores, basta modificar a cor novamente, imediatamente, antes de desenhar cada Gizmo.

Para mudar a cor atual dos Gizmos você deve usar o seguinte comando:

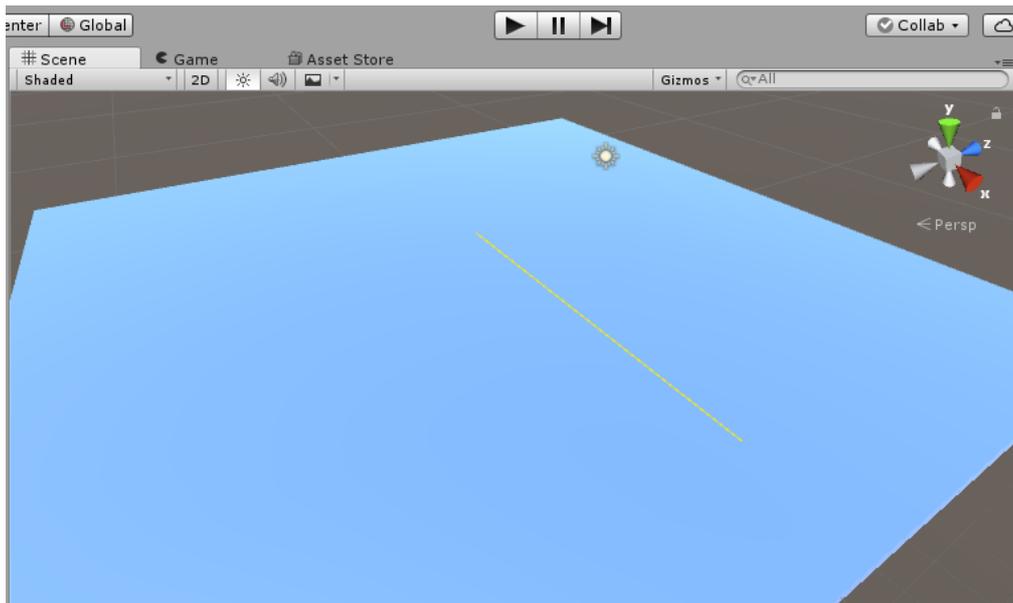
```
1 Gizmos.color = Color.yellow;
```

Isso mudará a cor dos Gizmos desenhados abaixo desse comando para amarelo, por exemplo. Vamos testar agora o desenho de uma simples linha amarela no editor a partir do centro até um ponto a 10 unidades de distância no eixo X, ou seja, X+10. A implementação ficará assim:

```
1 void OnDrawGizmos() {  
2     Gizmos.color = Color.yellow;  
3     Vector3 centro = transform.position;  
4     Vector3 final_da_linha = new Vector3 (centro.x + 10, centro.y, centro.z);  
5     Gizmos.DrawLine (centro, final_da_linha);  
6 }
```

Repare que o centro é a posição atual do GameObject com o script anexado (`transform.position`) e a variável `final_da_linha` é criada como um novo `Vector3` com as mesmas posições do centro, porém com o valor do eixo X adicionado de 10 unidades. Logo após usamos o comando `Gizmos.DrawLine(centro, final_da_linha)` para desenhar uma linha entre esses dois pontos. Salve o script e volte ao Unity. Você verá uma linha amarela desenhada a partir do centro do `IntanciarPrefabs` com 10 unidades de tamanho, mesmo sem executar o jogo, como na **Figura 12**. Legal, não é mesmo?

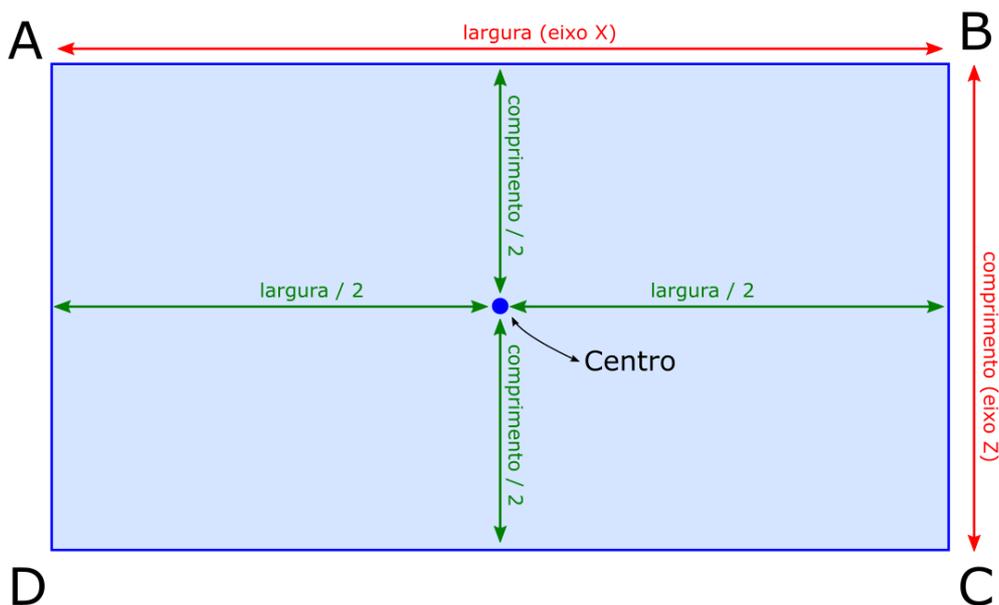
Figura 12 - Linha desenhada com Gizmos.DrawLine.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora ficou bem mais fácil! O que queremos é desenhar um retângulo, e cada linha dessa tem dois pontos, de início e fim. Um retângulo é composto por quatro linhas que ligam também quatro pontos (os cantos do retângulo). Primeiramente criaremos as variáveis A,B, C e D, as quais serão cada uma delas um Vector3 para representar esses cantos do retângulo. Observe que mesmo o retângulo se tratando de uma figura 2D, precisamos de coordenadas em 3D (Vector3) para representar a posição dos seus cantos. Temos a variável centro, que representa o **centro** do retângulo, e também temos **largura** e **comprimento**, representando suas dimensões. Como conseguir os valores de A, B, C e D baseado nessas variáveis? Veja a **Figura 13** para ficar mais claro.

Figura 13 - Cantos do retângulo.



Fonte: Elaborado pelo autor (2017)

Observando a **Figura 13**, podemos observar que as coordenadas das variáveis A, B, C e D são:

```
1 Vector3 A = new Vector3 (centro.x - largura/2, centro.y, centro.z - comprimento/2);  
2 Vector3 B = new Vector3 (centro.x + largura/2, centro.y, centro.z - comprimento/2);  
3 Vector3 C = new Vector3 (centro.x + largura/2, centro.y, centro.z + comprimento/2);  
4 Vector3 D = new Vector3 (centro.x - largura/2, centro.y, centro.z + comprimento/2);
```

Um Vector3 novo é criado para cada variável com o comando “new” e os parâmetros de entrada são as posições x, y e z. Repare que para cada valor de x e z de cada Vector3 é passado o centro.x e centro.z adicionando ou subtraindo a metade dos valores das larguras e comprimento, fazendo com que cada um deles represente um canto do retângulo. O valor de Y (altura) sempre é mantido o mesmo do centro, já que o retângulo é um plano que ficará nessa mesma altura.

Depois de definir os valores de A, B, C e D, agora desenharemos as quatro linhas ligando esses pontos e formando um retângulo, assim:

```
1 Gizmos.DrawLine (A, B);  
2 Gizmos.DrawLine (B, C);  
3 Gizmos.DrawLine (C, D);  
4 Gizmos.DrawLine (D, A);
```

Juntando tudo, nosso método OnDrawGizmos ficará como na **Figura 14**.

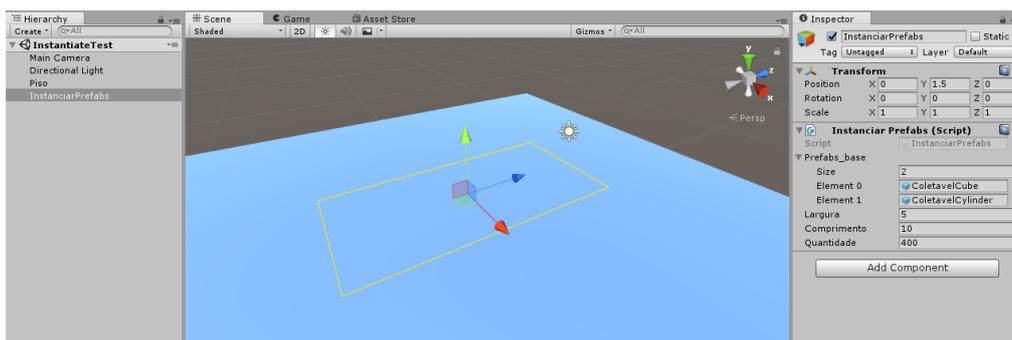
Figura 14 - Método OnDrawGizmos final.

```
35 void OnDrawGizmos() {
36     Gizmos.color = Color.yellow;
37     Vector3 centro = transform.position;
38     Vector3 A = new Vector3 (centro.x - largura/2, centro.y, centro.z - comprimento/2);
39     Vector3 B = new Vector3 (centro.x + largura/2, centro.y, centro.z - comprimento/2);
40     Vector3 C = new Vector3 (centro.x + largura/2, centro.y, centro.z + comprimento/2);
41     Vector3 D = new Vector3 (centro.x - largura/2, centro.y, centro.z + comprimento/2);
42
43     Gizmos.DrawLine (A, B);
44     Gizmos.DrawLine (B, C);
45     Gizmos.DrawLine (C, D);
46     Gizmos.DrawLine (D, A);
47 }
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Salve o seu script e retorne ao Unity. Você verá que um retângulo amarelo está exatamente no centro do InstanciarPrefabs, mostrando precisamente os limites da área de geração dos prefabs sem a necessidade de executar o jogo. Veja a **Figura 15**.

Figura 15 - InstanciarPrefabs com o Gizmo personalizado que criamos.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Selecionando o InstanciarPrefabs no Hierarchy e observando o seu Inspector, altere os valores de largura e comprimento e verá que o retângulo amarelo muda em tempo real, acompanhando suas mudanças. Do mesma maneira, se você mover o GameObject InstanciarPrefabs o retângulo acompanha o movimento se posicionando no local correto.

Criamos, assim, o nosso primeiro Gizmo personalizado! Foi fácil, não é mesmo? Salve seu jogo antes de continuar.

Tipos de Desenhos com Gizmos

Existem diversos tipos de primitivas que podem ser desenhadas com Gizmos além do DrawLine. Veja na **Tabela 1** os métodos que você pode utilizar:

Método	Função
DrawCube	Desenha um cubo sólido.
DrawFrustum	Desenha um campo de visão de câmera (chamado de Frustum) usando a propriedade Gizmos.matrix configurada anteriormente para sua localização e rotação.
DrawGUITexture	Desenha uma textura na cena.
DrawIcon	Desenha um ícone em uma posição na Scene View.
DrawLine	Desenha uma linha de um ponto a outro.
DrawMesh	Desenha um Mesh 3D na cena.
DrawRay	Desenha um raio de um ponto seguindo uma direção.
DrawSphere	Desenha uma esfera sólida com um centro e um raio.
DrawWireCube	Desenha um cubo vazado exibindo somente suas arestas (wireframe) em um centro e de determinado tamanho.
DrawWireMesh	Desenha um Mesh 3D com somente suas arestas (wireframe).
DrawWireSphere	Desenha uma esfera com somente suas arestas (wireframe).

Tabela 1 -Métodos de desenho com Gizmos.

Atividade 01

1. Implemente outra versão do `OnDrawGizmos()`, mas em vez de utilizar o `DrawLine` para criar as bordas do retângulo, utilize o `DrawCube` para criar um cubo com as mesmas dimensões do retângulo (largura e comprimento), porém com a altura bem baixa (valor 0,1).
2. Em vez de usar o comando `Gizmos.color = Color.yellow;` para definir a cor do Gizmo, utilize o comando `new Color(...)`; para criar uma nova cor. Cores são criadas com os componentes Red, Green, Blue e, opcionalmente, Alpha (opacidade). Crie uma cor amarelo translúcido (50%) para o seu plano. O resultado deve ficar como na figura abaixo:



Para verificar suas respostas, clique [aqui](#).

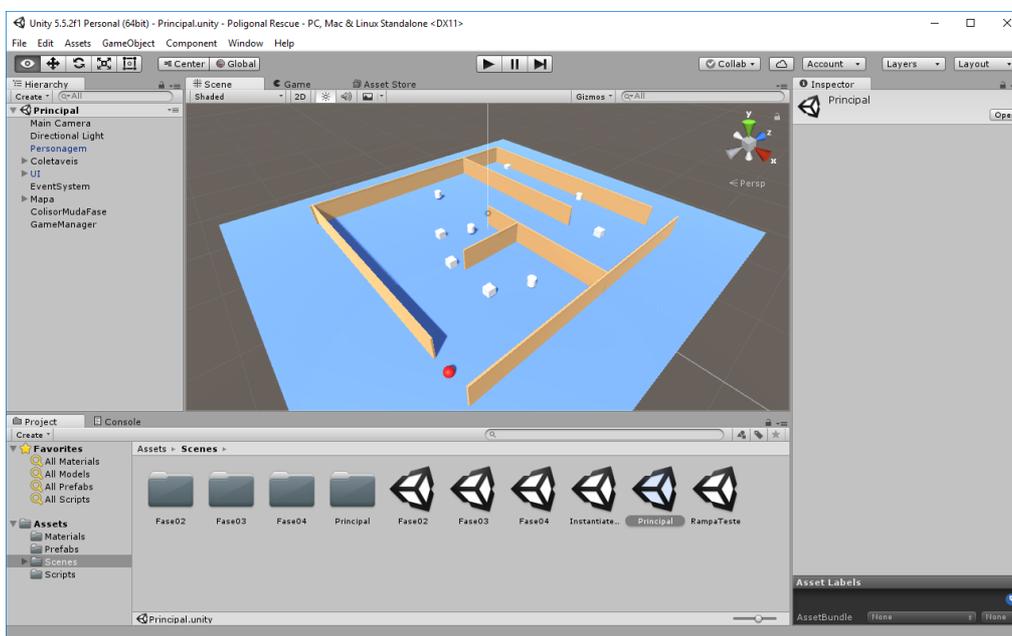
```
void OnDrawGizmos() {
    Gizmos.color = new Color (255f, 255f, 0f, 0.5f); // Cor amarela translúcida
    Vector3 centro = transform.position;
    Gizmos.DrawCube(centro, new Vector3(largura, 0.1f, comprimento));
}
```

Integrando o InstanciarPrefabs no Jogo

Até então criamos o InstanciarPrefabs e o testamos em uma cena separada. Vamos agora integrá-lo em uma cena do *Polygonal Rescue*.

Já temos algumas cenas (fases) criadas, escolha a primeira cena, chamada aqui de Principal, e dê um duplo clique para abri-la no editor. Você deverá ver o primeiro labirinto como na **Figura 16**.

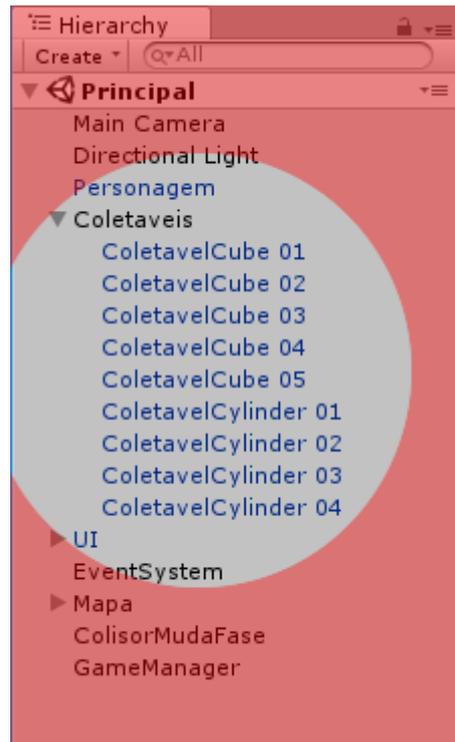
Figura 16 - Cena principal.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

No Hierarchy, expanda o GameObject **Coletaveis** e repare que todos os polígonos coletáveis estão criados abaixo dele, como na **Figura 17**.

Figura 17 - Polígonos coletáveis no Hierarchy.



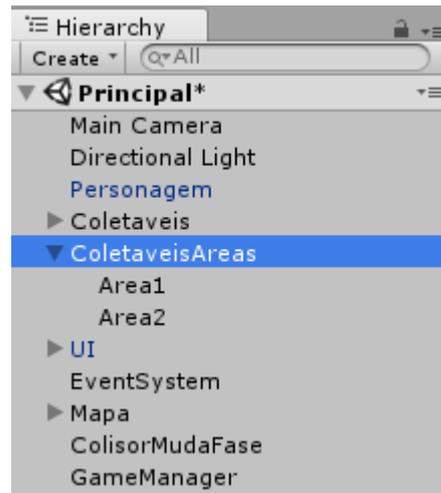
Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Atenção!

Lembre-se que esses polígonos são na verdade instâncias dos Prefabs ColetavelCylinder e ColetavelCube. Eles foram adicionados na cena simplesmente arrastando para uma região livre o seu prefab referente que está na nossa pasta “Prefabs”.

Para efeitos de organização vamos manter o GameObject **Coletaveis** como o pai dos polígonos que foram adicionados um a um. Então crie na raiz do Hierarchy um novo GameObject vazio chamado **ColetaveisAreas**, o qual armazenará as áreas retangulares de geração de Prefabs utilizando o nosso script InstanciarPrefabs. Teremos várias áreas em cada cena, portanto é importante organizá-las todas como filhas desse GameObject. Depois de criado, adicione abaixo dele mais 2 GameObjects vazios, chamando-os de Area1 e Area2. Você deve ficar com o Hierarchy, como mostra a **Figura 18**.

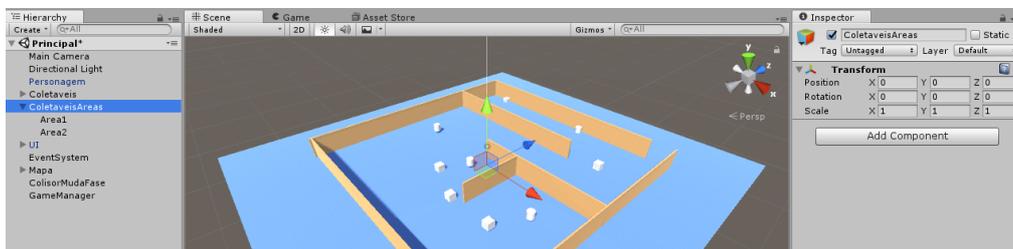
Figura 18 - Hierarchy com ColetaveisArea e duas áreas dentro dele.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora clique em ColetaveisAreas e centralize sua posição utilizando a função Reset Position no Inspector, como visto anteriormente. Isso deve trazer o Objeto para o centro da cena, facilitando sua localização. Veja na **Figura 19**.

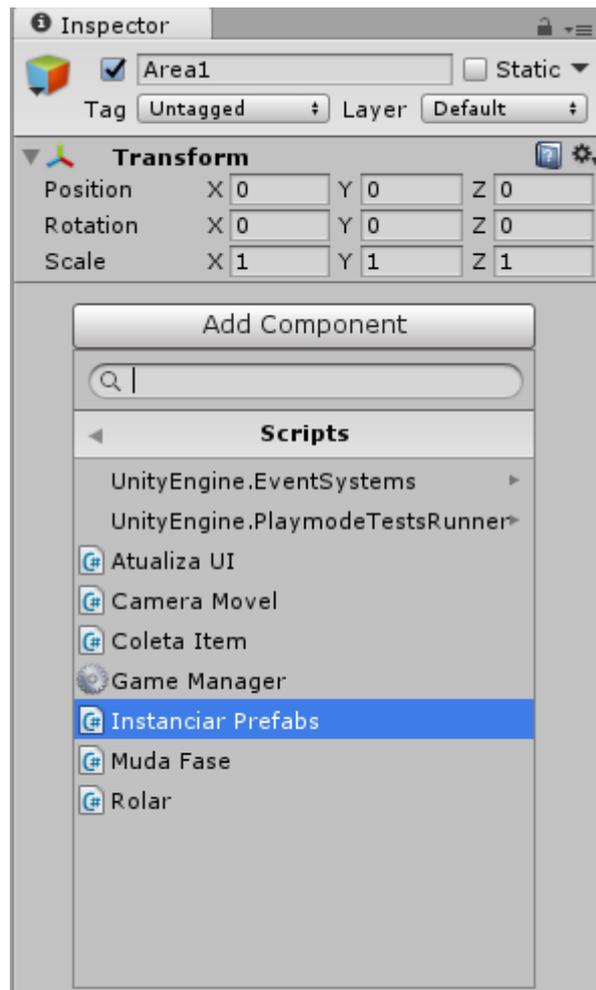
Figura 19 - ColetaveisAreas no centro da cena (Position; 0,0,0).



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora clique em Area1 e no inspector escolha a opção Add Component -> Scripts -> Instanciar Prefabs, como mostra a **Figura 20**.

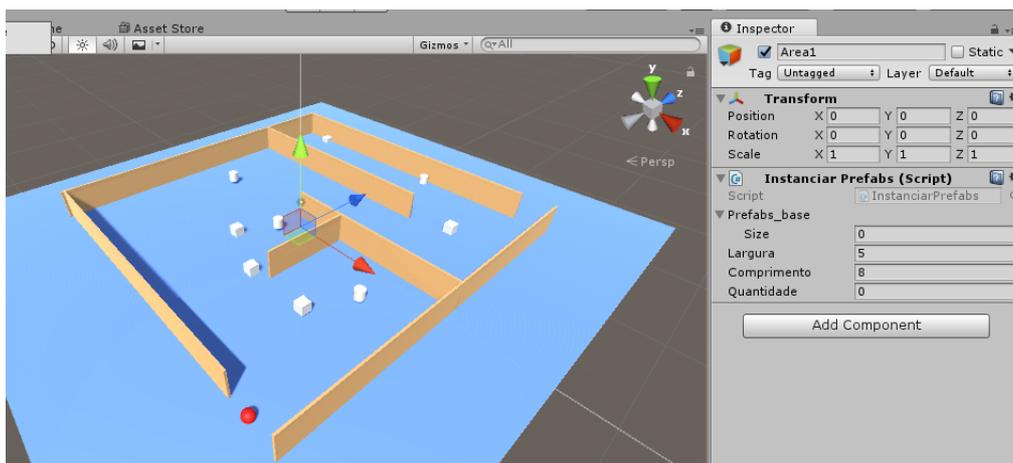
Figura 20 - Adicionando InstanciarPrefabs na Area1.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Você verá o novo componente InstanciarPrefabs adicionado no Inspector, porém com todos os valores zerados. Inicialmente modifique sua largura para 5 e comprimento para 8. Então ficará com algo parecido ao que se apresenta na **Figura 21**:

Figura 21 - Area1 com valores de largura e comprimento preenchidos.

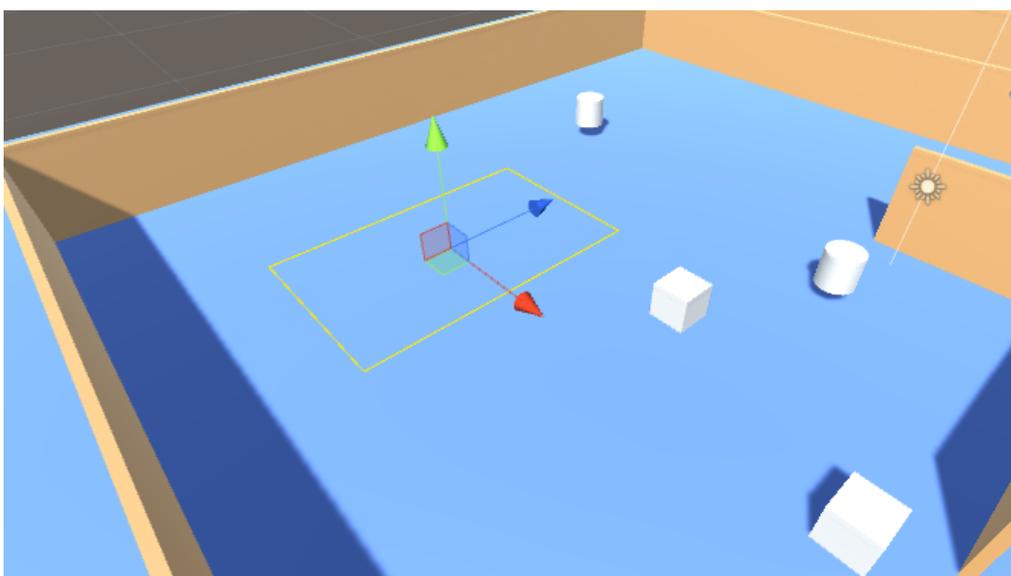


Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Observe que mesmo com os valores de largura e comprimento, o retângulo amarelo não está aparecendo na cena. O que houve?

Calma, acontece que como o GameObject Area1 está com o Position=(0,0,0), então ele está “dentro” do piso azul do labirinto, e o retângulo não está aparecendo porque está sendo desenhado dentro do piso. Para corrigir isso selecione Area1 e mova-o para cima no eixo Y até que você veja o retângulo amarelo. Aproveite e também já escolha o seu local final (nos eixos X e Z) para ocupar um espaço que ainda está sem polígonos, veja na **Figura 22**.

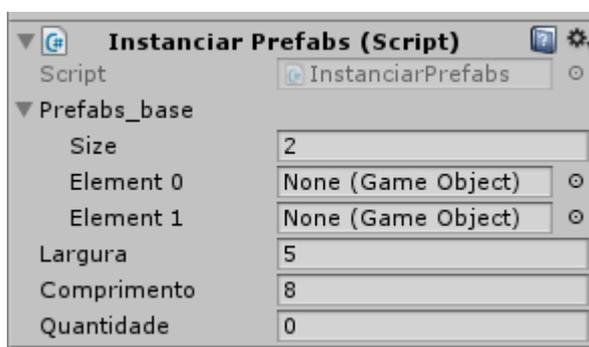
Figura 22 - Area1 reposicionada para que fique imediatamente acima do piso em um local sem polígonos.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Agora podemos conferir nosso retângulo que mostra onde serão criados os polígonos pelo script InstanciarPrefabs. Falta informar ao Area1 quais tipos de Prefabs queremos criar ali e a quantidade. Para isso clique em Area1 e no inspector abaixo de Prefabs_base preencha a propriedade Size=2, ou seja, queremos criar prefabs de 2 tipos já que temos somente 2 tipos de polígonos (ColetavelCylinder e ColetavelCube). Veja que logo depois disso surgiram mais duas novas propriedades: Element 0 e Element 1, como mostra a **Figura 23**. Essas propriedades estão com **None** como seu valor e devem ser preenchidas com os Prefabs dos tipos dos polígonos os quais desejamos criar.

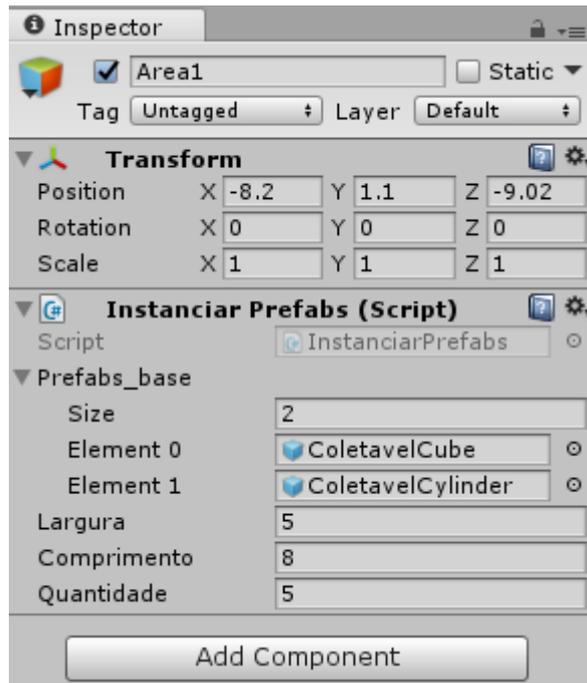
Figura 23 - Prefabs_base com 2 elementos ainda sem valor.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Localize na janela Project a pasta Prefabs, clique nela e ao exibir seu conteúdo arraste o prefab ColetavelCube para o Element 0 e arraste ColetavelCylinder para Element 1. Logo depois preencha a propriedade Quantidade com o valor 5, indicando que deseja criar 5 desses Prefabs nessa área retangular. Veja como ficou suas propriedades da Area1 no Inspector na **Figura 24**.

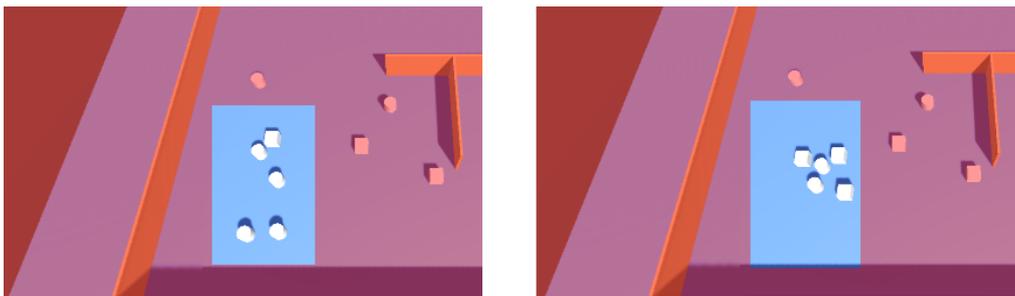
Figura 24 - Propriedades finais da Area1.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Pronto! Agora execute o jogo e veja que exatamente nessa área delimitada aparecerão 5 Prefabs em posições aleatórias os quais podem cada um ser do tipo ColetavelCube e ColetavelCylinder. Execute o jogo várias vezes e perceba que sempre as posições mudam, como na **Figura 25**. Legal, não acha?

Figura 25 - Duas execuções do jogo gerando polígonos na Area1 em posições distintas.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Atividade:

Agora é com você! Faça o mesmo procedimento para Area2 e posicione-a onde desejar no mapa. Você terá duas áreas de geração de Prefabs na sua cena. Bem prático, não é mesmo?

Ícones para GameObjects

Uma ferramenta simples e útil no Unity é a possibilidade de adicionar ícones em GameObjects. Normalmente um GameObject em um jogo 3D ou 2D tem associado a ele um Sprite(2D) ou um Mesh(3D), que é exibido na Scene View, entretanto alguns GameObject não têm uma forma simples de serem exibidos na Scene View, por exemplo o GameObject Area1 com o InstanciarPrefabs como componente. Sim, é verdade que criamos um Gizmo personalizado para exibir o seu retângulo, entretanto imagine uma cena com várias áreas similares a essa, cada uma com tamanhos e posições diferentes. Ficaria complicado saber qual delas na Scene View é qual GameObject no Hierarchy, já que os Gizmos criados são apenas linhas e sua seleção a partir do Scene View é bem difícil com o Mouse, pois você precisa clicar precisamente em cima de uma das linhas. Para resolver esse problema rapidamente o Unity permite adicionar ícones em qualquer GameObject. Já esse ícone permite que o GameObject seja selecionado a partir da Scene View, mesmo que seja vazio ou sem um Sprite ou um Mesh associado.

Para adicionar um ícone em Area1, selecione-o no Hierarchy e observe no Inspector que existe um botão . No formato de um cubo com 3 cores logo ao lado do seu nome, como na **Figura 26**.

Figura 26 - Opção de seleção de ícones de um GameObject.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Clicando nesse ícone você verá uma lista de opções de cores e formatos para o ícone do GameObject, veja na **Figura 27**.

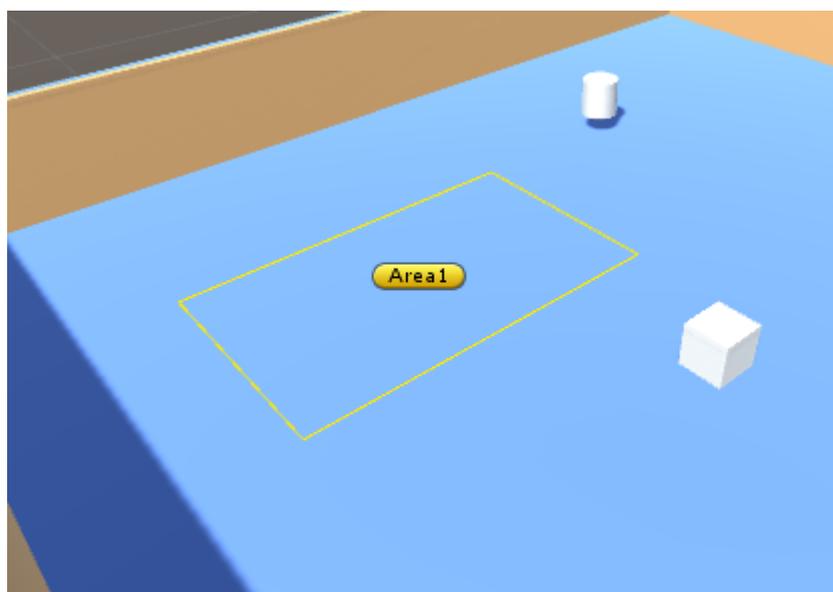
Figura 27 - Opções de cores e formatos para o ícone do GameObject.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Você pode, naturalmente, escolher qualquer uma dessas cores, mas existe uma diferença maior com relação aos formatos. Selecionando as opções similares a etiquetas, como esta amarela:  o ícone conterá o nome do GameObject no seu centro, enquanto escolher qualquer outro formato, como  ou  o ícone não terá o nome do GameObject no centro. Vamos escolher a opção de etiqueta amarela  para Area 1. Agora observe que na Scene View a Area1 tanto tem o nosso Gizmo retangular como um ícone amarelo no centro com o nome "Area1" e o mais legal é que se você clicar nesse ícone o objeto é selecionado, sem a necessidade de localizá-lo no Hierarchy. Veja na **Figura 28** como Area1 aparece na Scene View.

Figura 28 - Area1 com um ícone de etiqueta amarela.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 21 de março de 2017.

Esses pequenos detalhes ajudam muito na sua produtividade, então sempre estude sua ferramenta a fundo consultando a documentação oficial e também outras fontes de aprendizado.

Estamos chegando ao fim de mais uma aula! Espero que tenham gostado e, sobretudo, aprendido bastante. O nosso jogo está ficando ótimo, mas ainda há muito a ser feito! Na próxima aula daremos continuidade ao processo de desenvolvimento dele. Até lá!

Resumo

Nesta aula aprendemos a criar Gizmos personalizados no Unity, deixando a interação com scripts mais visual e representativa. Também integramos o nosso script de instanciar Prefabs no jogo, o que possibilitou a criação de zonas nas quais são instanciados os objetos coletáveis de forma aleatória. Além disso aprendemos a adicionar ícones em GameObjects, permitindo sua melhor visualização na cena mesmo quando se trata de um GameObject vazio.

Leitura Complementar

A criação de Gizmos personalizados é bastante extensa e poderosa, permitindo que você crie sistemas muito úteis para a manipulação das propriedades de objetos durante a edição. Seguem alguns links para estudar mais sobre o assunto:

- <https://docs.unity3d.com/Gizmos.html>
- <https://docs.unity3d.com/Object.Instantiate.html>
- <https://docs.unity3d.com/OnDrawGizmos.html>

Autoavaliação

1. Os Gizmos são bastante versáteis e customizáveis. Altere o método `OnDrawGizmos` do `InstanciarPrefabs` e modifique a cor dos Gizmos para uma diferente imediatamente antes de cada `DrawLine`. O que aconteceu?
2. Altere o ícone de `Area1` para um com formato menor, somente um ponto amarelo. Observe o que aconteceu na `Scene View`.
3. Tente imaginar como seria o desenho de um Gizmo personalizado para um script que instancia Prefabs em locais aleatórios, porém não só em um plano, mas em um volume de 3 dimensões. Qual das primitivas de desenho do Gizmo você usaria?

Referências

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Online Tutorials [online]. Disponível em: <https://unity3d.com/pt/learn/tutorials> [Acessado em 16 de novembro de 2016].

UNITY TECHNOLOGIES. 2016 (C). Unity Manual - Prefabs [online]. Disponível em: <https://docs.unity3d.com/Manual/Prefabs.html> [Acessado em 16 de novembro de 2016].

STOY, C. 2006. Game object component system. In Game Programming Gems 6, Charles River Media, M. Dickheiser, Ed., Páginas 393 a 403.

MARQUES, Paulo; PEDROSO, Hernâni - C# 2.0 . Lisboa: FCA, 2005. ISBN 978-972-722 208-8

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Manual [online]. Disponível em: <https://docs.unity3d.com/Manual/index.html> [Acessado em 16 de novembro de 2016].