

Desenvolvimento com Motores de Jogos II

Aula 06 - Jogo Polygonal Rescue - Parte 4 -
Prefabs e Mudan as de Fases

Apresentação

Olá, pessoal! Sejam bem-vindos à Aula 6 da disciplina de Motores de Jogos III! Nas aulas anteriores, criamos uma versão do jogo *Polygonal Rescue* na qual o jogador pode já resgatar os polígonos no labirinto enquanto o resolve até o seu fim. Além disso, implementamos uma simples HUD (Heads-up display) para o jogo, a qual mostra a pontuação atual do jogador, isto é, quantos polígonos ele já resgatou.

Entretanto, nosso jogo limita-se atualmente a apenas um cenário, ou seja, somente uma fase, um labirinto para se jogar, o que não o torna muito atrativo.

Na aula de hoje, avançaremos ainda mais. Aprenderemos a criar outras fases para nosso jogo, assim como objetos que podem ser reutilizados entre as fases (cenas), chamados de Prefabs. Preparados?

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Criar objetos reutilizáveis com Prefabs;
- Criar instâncias de Prefabs na cena;
- Instanciar Prefabs utilizando scripts;
- Criar um sistema de instanciação aleatória de Prefabs ao redor de uma posição desejada.

Prefabs

No Unity, Prefabs são GameObjects pré-configurados que você cria na cena e armazena no seu projeto. Assim, eles podem ser clonados em diversas cenas do projeto, criando-se, dessa maneira, **instâncias** do Prefab com as mesmas características do original. Isso possibilita que GameObjects, como o personagem e os polígonos coletáveis de determinado tipo, por exemplo, possam ser criados somente uma vez e reutilizados em todas as cenas (ou fases) do jogo.

Você pode se perguntar o seguinte: mas não basta copiar e colar o personagem de uma cena para outra a fim de obter o mesmo efeito de quando um Prefab é usado? A resposta para essa pergunta é sim e não. É verdade que, se o desenvolvedor copiar um GameObject de uma cena e colar em outra, essa cópia terá as mesmas características do original, entretanto, essas duas cópias do personagem não são instâncias de um mesmo GameObject, pois são duplicatas totalmente independentes. Isso significa que, se você fizer uma mudança em uma delas, essa mudança não será refletida na outra, te obrigando a realizar a mesma mudança em todas as cópias do personagem. Imagine fazer isso em um jogo com 50 fases! Não é nada prático. Os Prefabs ajudam a resolver esse problema, permitindo que você “transforme” o personagem original em um Prefab e utilize esse Prefab para criar as outras instâncias dele em outras fases. Assim, uma mudança feita em um dos personagens pode se refletir para todos os outros, sem a necessidade de se alterar manualmente todos eles.

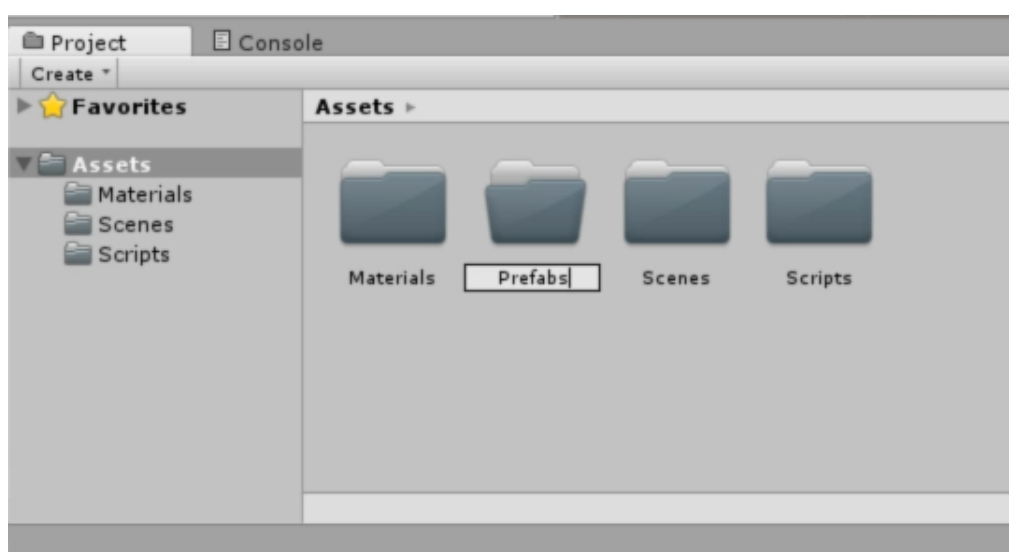
Outra funcionalidade dos Prefabs é a possibilidade de se criar instâncias deles no jogo durante a sua execução, ou seja, enquanto o jogo está rodando você pode, através de comandos nos scripts, criar instâncias de Prefabs na cena atual. Isso é muito útil em casos que você não sabe previamente quantos objetos de um determinado tipo precisam ser criados nem o local dessa criação. Imagine um jogo de tiro, por exemplo, no qual o jogador deseja realizar um disparo. Você não pode criar infinitos projéteis na sua arma, pois isso, além de impossível, deixaria o jogo muito pesado. Alternativamente, você pode criar um Prefab que representa um projétil e, no momento de cada disparo, criar uma nova instância dele e configurar

sua velocidade e direção de acordo com a posição e orientação da arma. Existem muitos casos nos quais o uso de Prefabs é bastante útil, tanto para a criação de instâncias no editor do Unity como durante a execução do jogo.

Criando o Primeiro Prefab

Antes de criar um novo Prefab, criaremos uma nova pasta no nosso projeto, a qual terá o sugestivo nome de “Prefabs”. Veja a **Figura 1**.

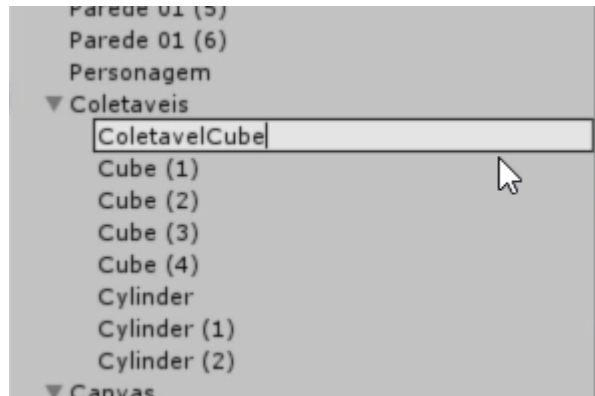
Figura 01 - Criação da pasta para os Prefabs.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Depois de criar essa pasta, criaremos nosso primeiro Prefab. Começaremos com os nossos polígonos resgatáveis. Temos atualmente no nosso jogo dois tipos de polígonos resgatáveis, os Cubos e os Cilindros, dos quais criamos várias cópias na nossa cena e ainda precisaremos criar várias outras cópias nas demais cenas (fases). Sendo assim, esses GameObjects são excelentes candidatos para se transformarem em Prefabs. Selecione, no Hierarchy, o primeiro polígono do tipo Cube, que está com o nome “Cube”, e renomeie para “ColetavelCube”, como pode ser observado na **Figura 2**.

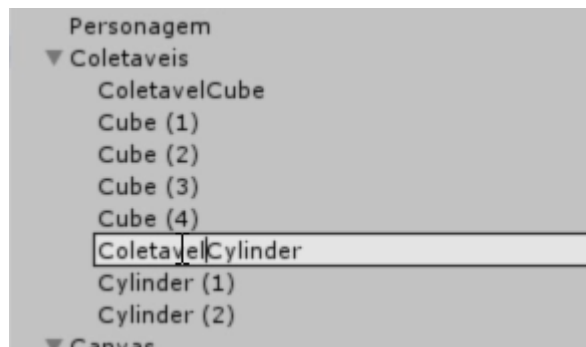
Figura 02 - Renomeando o primeiro polígono do tipo Cube para “ColetavelCube”.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Esse nome é importante, pois será o nome do nosso Prefab. Faça o mesmo para o primeiro polígono do tipo Cylinder, renomeando-o para “ColetavelCylinder”, como na **Figura 3**.

Figura 03 - Renomeando o primeiro polígono do tipo Cylinder para “ColetavelCylinder”.



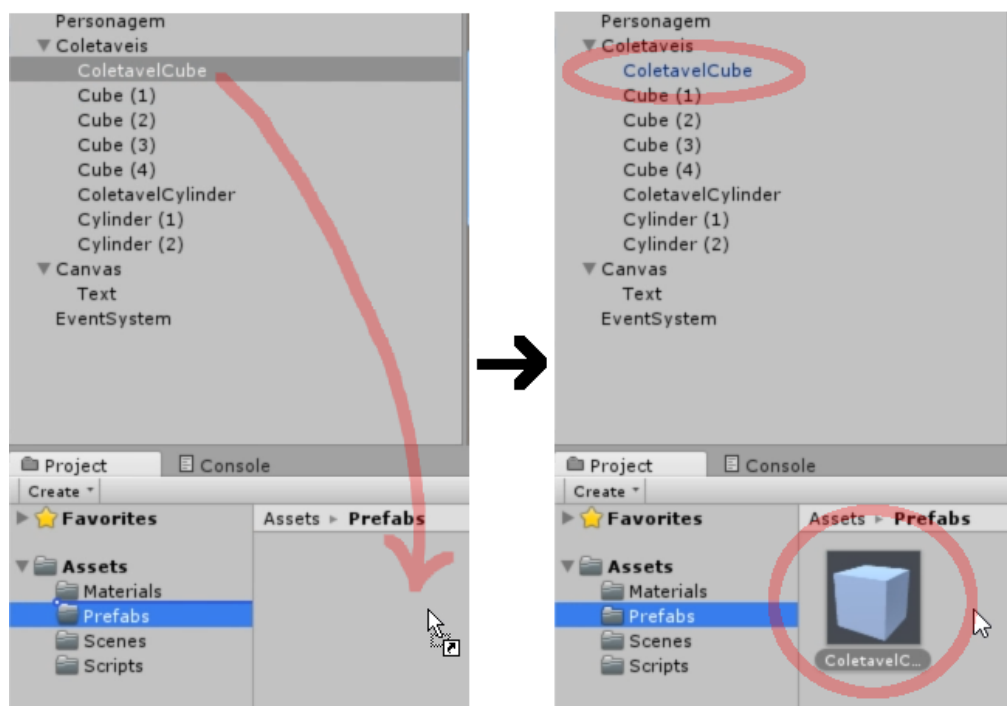
Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Temos agora dois dos nossos polígonos, um de cada tipo, renomeados e prontos para transformarem-se em Prefabs. Por enquanto, mantenha os outros polígonos no Hierarchy como estão, mas saiba que eles serão removidos e substituídos por instâncias dos últimos Prefabs criados.

Começando com o ColetavelCube, arraste esse GameObject para dentro da pasta Prefabs que você criou. Pronto, simples assim! Você acabou de realizar duas ações com esse simples arrastar e soltar. A primeira ação foi criar um Prefab com o mesmo nome do GameObject arrastado (ColetavelCube). Em seguida, o Unity automaticamente removeu o GameObject ColetavelCube da sua cena e o substituiu por uma **instância** desse Prefab. Repare que, agora, existe um novo Prefab na

pasta, como esperado, e no Hierarchy o ColetavelCube que antes era exibido com um texto de cor preta, agora é um texto de cor azul. Essa cor azul significa ser ele um GameObject, o qual na verdade é uma instância de um Prefab. Veja a **Figura 4** detalhando o processo.

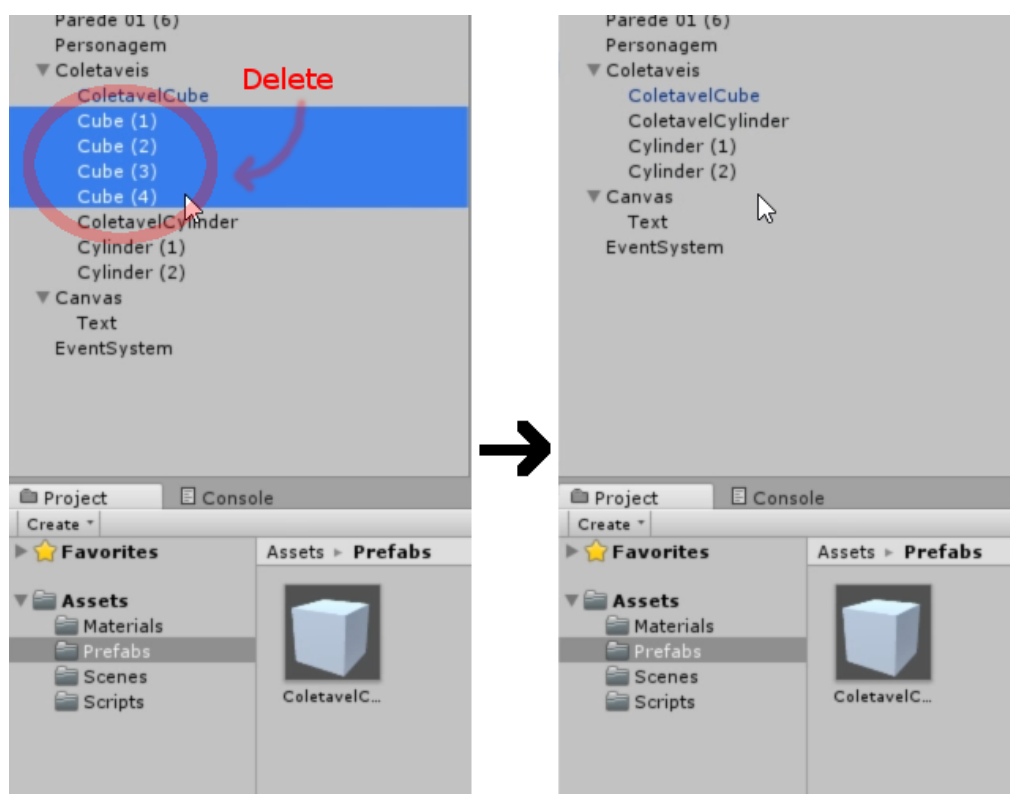
Figura 04 - Criação de um Prefab baseado em um GameObject da cena.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Agora, como temos um Prefab chamado ColetavelCube, podemos remover os outros polígonos do tipo Cube da cena os quais não são ainda uma instância desse Prefab e, então, substituí-los por instâncias dele. Selecione os outros Cubes da cena (exceto do que já é uma instância do Prefab ColetavelCube) e pressione a tecla DELETE para removê-los. O processo está detalhado na **Figura 5**.

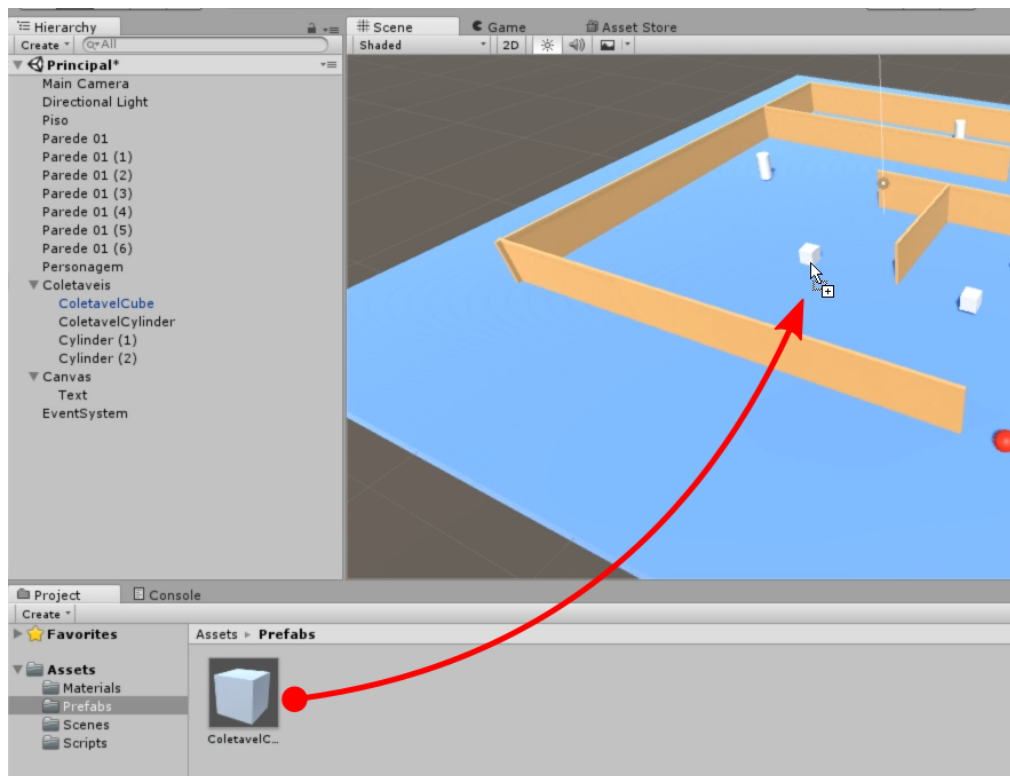
Figura 05 - Remoção dos polígonos do tipo Cube que ainda não são instâncias do Prefab ColetavelCube.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Depois, a fim de criar as instâncias do ColetavelCube na cena, basta arrastar o Prefab com o mesmo nome para o local onde deseja criar a instância. Crie algumas instâncias nos locais desejados. A **Figura 6** detalha o processo.

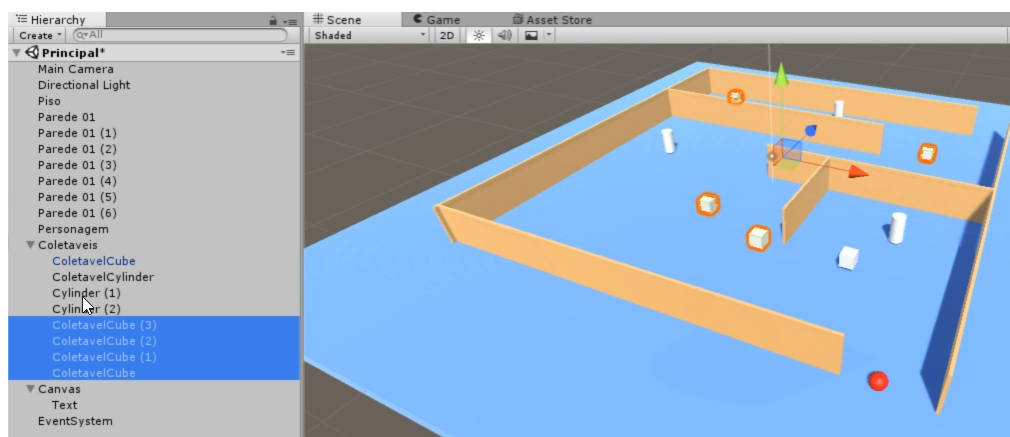
Figura 06 - Criação de uma instância do Prefab ColetavelCube.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Repare que as instâncias criadas têm as mesmas propriedades do Prefab original, já com a Tag ItemColetavel configurada e com a opção Is Trigger marcada no Box Collider. Ao criar as instâncias arrastando o Prefab para a cena, também as cria no Hierarchy, naturalmente. Entretanto, é necessário que você as coloque dentro do GameObject “Coletaveis”, como filhas dele, para manter a organização planejada. Faça isso e teremos algo como na **Figura 7**.

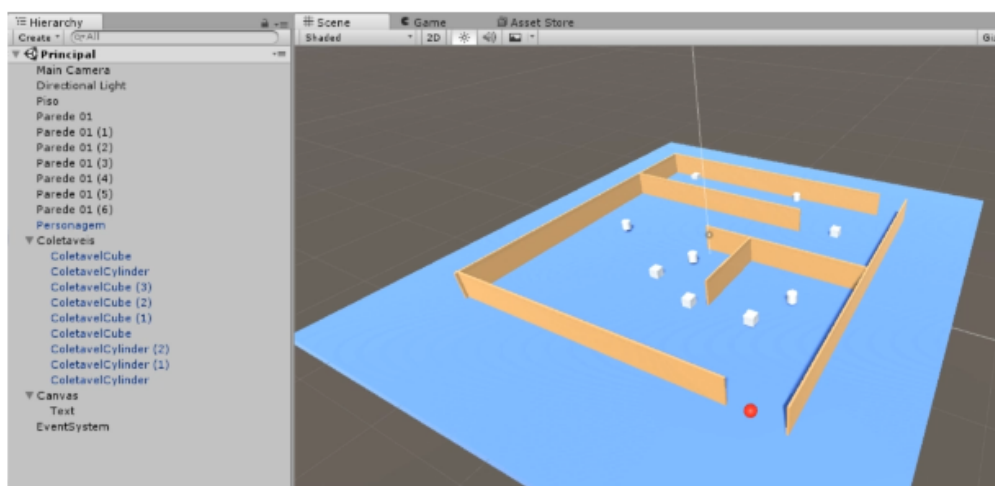
Figura 07 - Várias instâncias do Prefab ColetavelCube criadas na cena e organizadas no Hierarchy como filhas do GameObject Coletaveis.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Agora, repetiremos o processo para o Cylinder, criando seu Prefab e substituindo os GameObjects do tipo Cylinder por instâncias desse Prefab. Renomeie o primeiro GameObject do tipo Cylinder para ColetavelCylinder, arraste esse GameObject para a pasta Prefabs, remova as outras cópias do Cylinder que ainda não são instâncias do Prefab e arraste algumas vezes o Prefab novo ColetavelCylinder para a cena, posicionando nos locais desejados e não esquecendo de organizá-los como filhos do GameObject Coletaveis. Você deve acabar com uma cena na qual todos os polígonos resgatáveis são instâncias ou do Prefab ColetavelCube ou de ColetavelCylinder, similar ao que é mostrado na **Figura 8**.

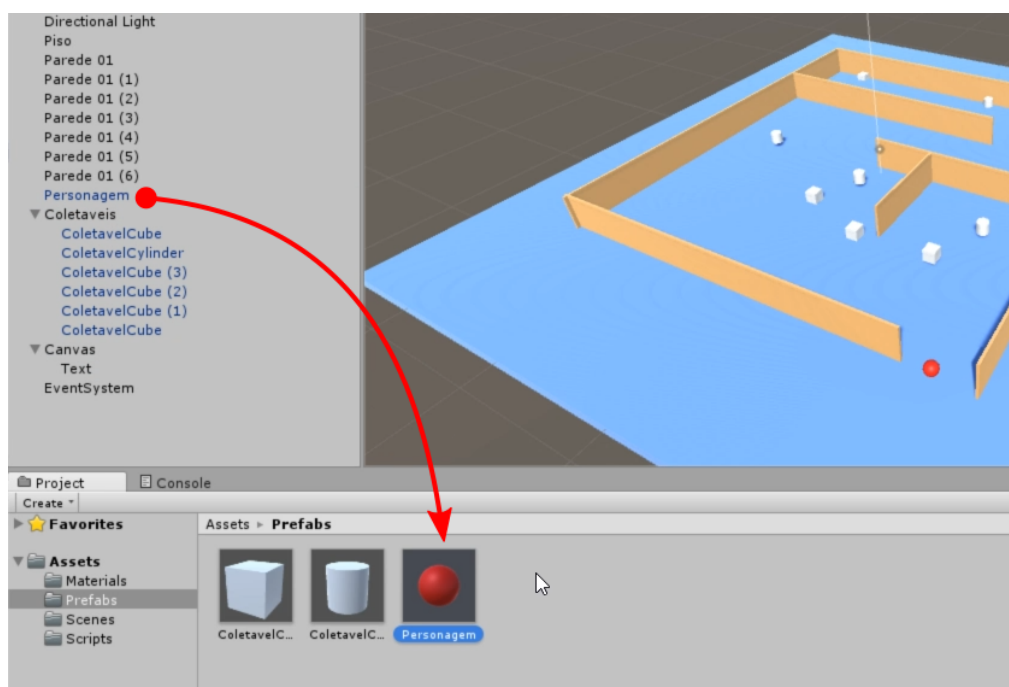
Figura 08 - Polígonos resgatáveis como instâncias dos Prefabs ColetavelCube ou do ColetavelCylinder.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Agora que você já criou dois Prefabs para os polígonos e já está utilizando-os na cena, faremos o mesmo para o nosso personagem. Selecione o personagem no Hierarchy e arraste para a pasta Prefabs. Como só temos uma instância do personagem na cena (e só precisamos de uma), não precisa fazer mais nada, pois o personagem da cena já é uma instância do Prefab criado. Veja a **Figura 9**.

Figura 09 - Prefab do personagem criado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Muito bem! Nosso personagem agora já é um Prefab. Você pode estar se perguntando o porquê de estarmos escolhendo esses GameObjects, até então o personagem e os polígonos, para criar Prefabs. Lembre-se que estamos criando Prefabs dos GameObjects que serão reutilizados, tanto na mesma cena como em outras cenas. No caso dos polígonos, várias instâncias deles são usadas em uma única cena, bem como também serão criados em outras fases do jogo (cenas). No caso do personagem, apenas existirá um por fase, mas ele será usado em todas as fases. Portanto esse é o motivo da escolha desses GameObjects para a criação de Prefabs. No caso do piso, da câmera e das paredes, por exemplo, cada fase terá a sua com tamanho e configurações diferentes, então eles não são candidatos para a criação de Prefabs e, se fossem, todas as fases teriam o mesmo labirinto, ou mesmo diferentes eles teriam de usar paredes e pisos de mesmo tamanho, mas isso não é o que queremos.

Existe um último GameObject o qual utilizaremos em todas as fases do jogo que pode também ser um Prefab. Trata-se da nossa UI, tendo como GameObject raiz um Canvas. O UI representa o HUD (Hheads-up display) do nosso jogo, que é responsável por exibir a pontuação. Ele será o mesmo em todas as fases do mapa, somente exibindo valores diferentes para a pontuação, mas isso é alterado por um script.

Então vamos criar o Prefab do nosso UI, inicialmente selecione o GameObject Canvas e renomeie para UI, como na Figura 10. Isso dá um nome mais apropriado para o GameObject e consequentemente para o Prefab a ser criado.

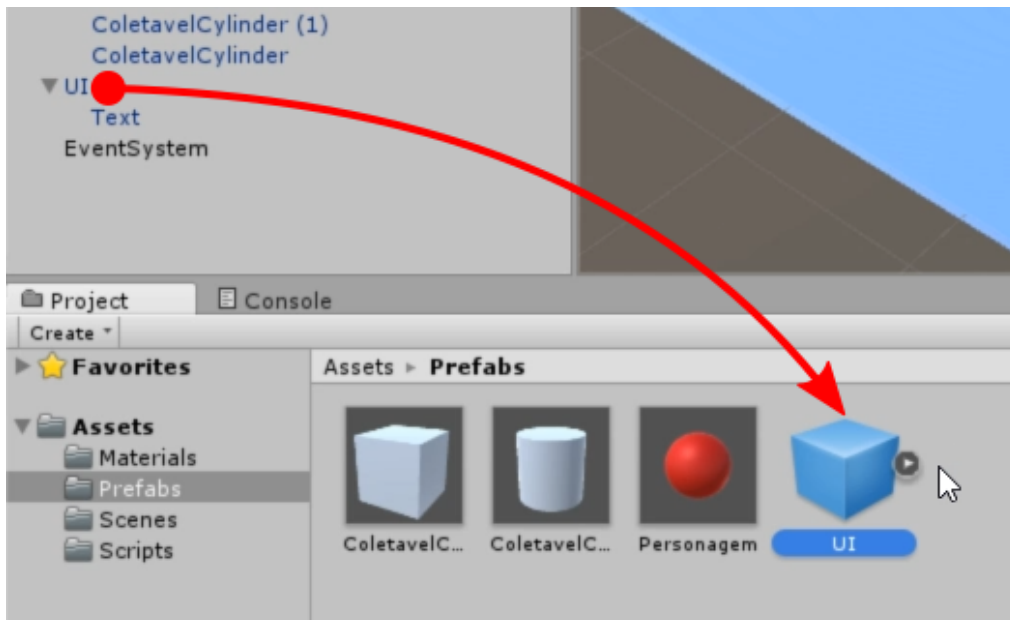
Figura 10 - Renomeando o Canvas para UI, deixando-o preparado para a criação do Prefab.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Com o HUD renomeado de Canvas para UI, vamos agora criar o Prefab dele arrastando esse GameObject para a pasta Prefabs. Ver **Figura 11**.

Figura 11 - Criando o Prefab para o UI que contém o nosso HUD.



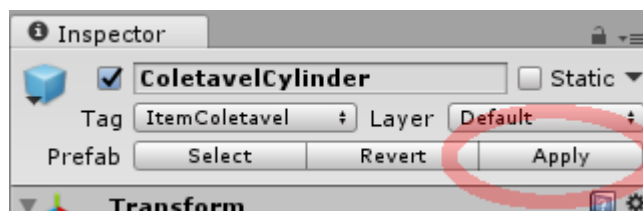
Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Ótimo! Temos agora todos os Prefabs necessários para a criação de novas fases para o nosso jogo de uma maneira organizada e produtiva. Criaremos novas fases na nossa próxima aula, por enquanto, vamos aprender uma forma mais poderosa de se criar instâncias de Prefabs, que é através de scripts! Lembrando que nos encontros presenciais podemos tirar as dúvidas e esclarecer algum questionamento.

Alterando um Prefab

Como vimos, as instâncias dos Prefabs no Hierarchy ficam com a cor azul e eles são criados com todas as propriedades do Prefab original. Entretanto você pode realizar mudanças nos Prefabs depois de serem criados facilmente. Para isso selecione uma instância do Prefab que você deseja alterar (se ele não existe crie arrastando o prefab para a cena) e depois modifique alguma de suas propriedades, por exemplo a sua escala. Faça isso com uma instância do ColetavelCylinder tornando-o ligeiramente mais alto (aumentando o valor do Scale Y). Verá que somente essa instância teve sua altura modificada, pois até então somente ela foi modificada e não o Prefab em si. Para aplicar a mudança no Prefab basta selecionar a instância que foi alterada e, no Inspector (na seção “Prefab”), clique no botão “Apply”. Isso fará com que as alterações dessa instância sejam salvas no Prefab e assim todas as outras instâncias desse Prefab no jogo ficarão com a mudança realizada. Ver **Figura 12**.

Figura 12 - Opção de aplicar as mudanças de uma instância para que elas sejam salvas no prefab.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Instâncias de Prefabs em Scripts

Já criamos nossos Prefabs e também suas instâncias usando o editor do Unity, posicionando-os nos locais desejados. Entretanto, no Unity é possível criar Prefabs através de scripts durante a execução do jogo. Isso é muito útil quando queremos criar GameObject instâncias de um Prefab em locais aleatórios, por exemplo, ou quando a sua criação depende do acontecimento de algum evento no jogo, como a

criação de um novo projétil quando um jogador realiza um disparo, a criação de fragmentos de vidro quando o jogador quebra um espelho, etc. São várias as aplicações.

No nosso jogo criamos instâncias dos Prefabs arrastando-os para a cena, escolhendo uma posição. Não alteramos a sua rotação, deixando-a como 0,0,0 no Transform, que é seu valor padrão, mas também poderíamos ter mudado esse valor depois de arrastar o Prefab.

A criação de Prefabs por scripts é realizada com o método **Instantiate** e ele pode ser chamado de algumas formas distintas, com parâmetros diferentes. Essas formas são:

- `public static Object Instantiate(Object original);`
- `public static Object Instantiate(Object original, Transform parent);`
- `public static Object Instantiate(Object original, Transform parent, bool instantiateInWorldSpace);`
- `public static Object Instantiate(Object original, Vector3 position, Quaternion rotation);`
- `public static Object Instantiate(Object original, Vector3 position, Quaternion rotation, Transform parent);`

Os parâmetros significam:

original	Um objeto existente que você deseja fazer a cópia (O Prefab).
position	Posição da nova instância.
rotation	Rotação da nova instância.
parent	O GameObject “pai” que a nova instância será criada como filha.

InstantiateInWorldSpace

Se quando associado ao pai a posição original relativa à cena será mantida.

Tabela 1 - Parâmetros do Instantiate.

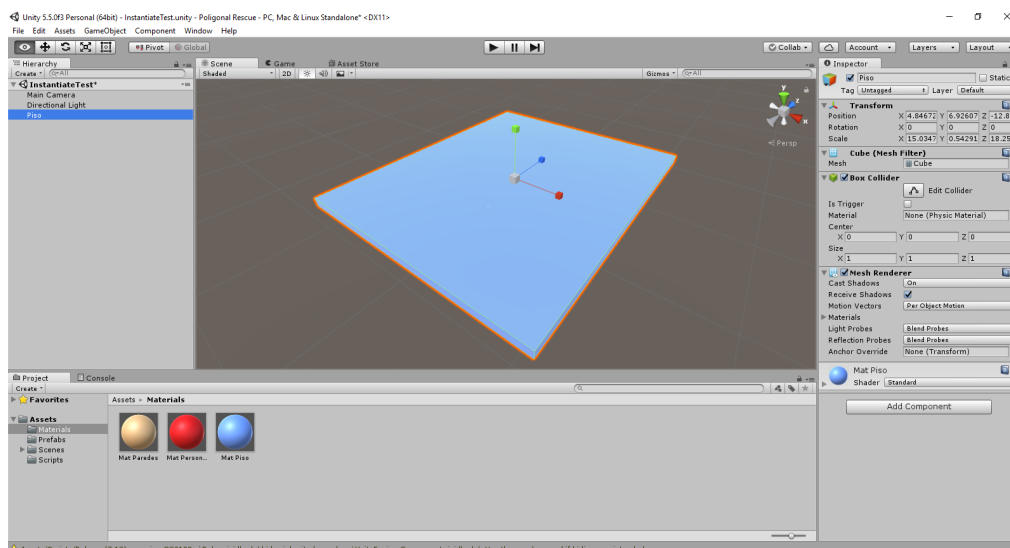
O método **Instantiate** retorna por padrão uma variável genérica do tipo `Object`, porém ela é uma referência para a instância recém-criada por ele. Caso você deseje manipular essa instância criada é muito comum armazená-la em uma variável do tipo `GameObject` (que é diferente de `Object`), e um modo de fazer isso é realizando uma conversão de tipos em C# adicionando “as `GameObject`” no final da chamada do método. Exemplo:

1	<code>GameObject novalInstancia = Instantiate(prefab, posicao, rotacao) as GameObject;</code>
---	---

Testando a Criação de Instâncias de Prefabs por Scripts em uma Nova Cena

Digamos que você deseja criar um sistema com a opção de escolher uma posição e no início do jogo uma determinada quantidade de polígonos (Cubos ou Cilindros) são criadas em locais aleatórios próximo dessa posição. Crie e salve uma nova cena chamada de “`InstantiateTeste`” para implementar esse mecanismo sem alterar nossa cena com o labirinto. Adicione um simples piso com o Material azul criado anteriormente. O resultado obtido pode ser observadona **Figura 13**.

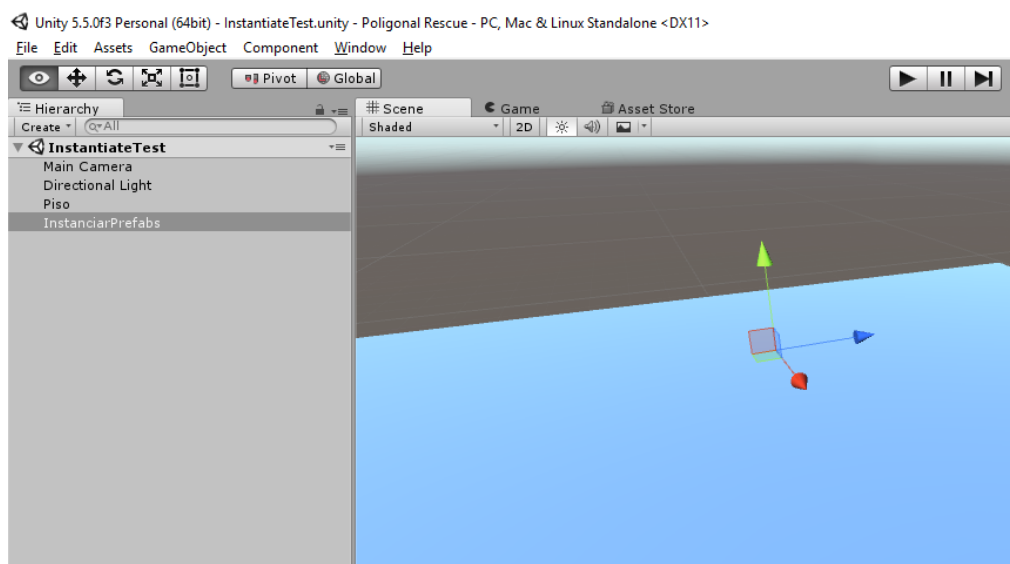
Figura 13 - Nova cena InstantiateTeste criada com um piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Crie agora um GameObject vazio, renomeie-o para “InstanciarPrefabs” e modifique sua posição para que fique sobre o piso, próximo ao centro. Ver **Figura 14**.

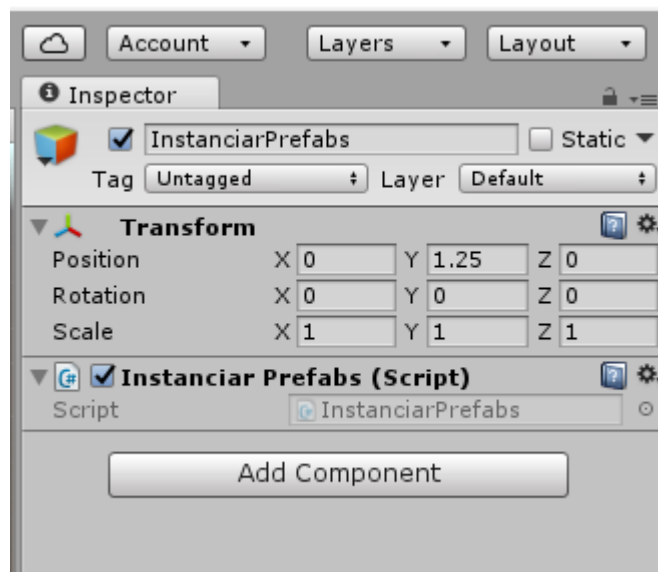
Figura 14 - GameObject InstanciarPrefabs posicionado sobre o piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Crie um novo script com o mesmo nome, “InstanciarPrefabs”, associando-o ao GameObject vazio InstanciarPrefabs, como na **Figura 15**.

Figura 15 - Script InstanciarPrefabs criado e associado ao seu GameObject.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

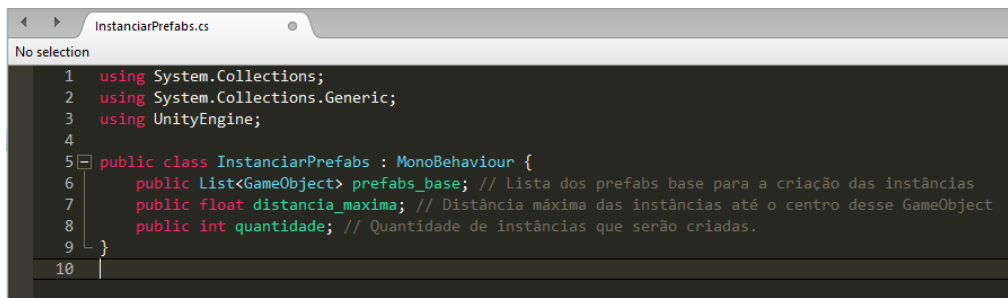
E aí? Tudo bem até aqui? Conseguiu entender o processo de criação de um novo script? Lembre-se que no encontro presencial você poderá esclarecer todas as suas dúvidas e questionamentos a respeito desse assunto.

Edite o script InstanciarPrefabs, apague os métodos Start e Update para iniciar com um script vazio e adicione as variáveis públicas:

- Uma lista de GameObject chamada “prefabs_base” que guardará os prefabs usados como base para a criação das instâncias aleatórias. Essa lista será posteriormente preenchida com os prefabs ColetavelCube e ColetavelCylinder.
- Uma variável float chamada “distancia_maxima” representando a distância máxima do centro em que a posição aleatória das instâncias dos prefabs serão criadas.
- Uma variável int chamada “quantidade” que indicará quantas instâncias aleatórias queremos criar.

Veja na **Figura 16** as variáveis adicionadas no nosso script até agora.

Figura 16 - Script InstanciarPrefabs com suas variáveis.



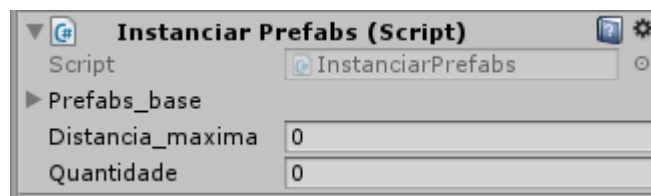
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class InstanciarPrefabs : MonoBehaviour {
6     public List<GameObject> prefabs_base; // Lista dos prefabs base para a criação das instâncias
7     public float distancia_maxima; // Distância máxima das instâncias até o centro desse GameObject
8     public int quantidade; // Quantidade de instâncias que serão criadas.
9 }
10
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Repare que nesse script a variável `prefabs_base` é do tipo `List`, ou seja, ela armazena vários objetos do tipo `GameObject`. Usaremos essa variável para armazenar nossos prefabs `ColetavelCube` e `ColetavelCylinder`, os quais serão usados para a criação das instâncias, porém esse sistema pode servir para qualquer Prefab em outro jogo. Se você observou bem, o tipo que a lista `prefabs_base` armazena é `GameObject` e não `ColetavelCube` ou `ColetavelCylinder`. Isso está correto, pois no Unity praticamente todos os objetos que são utilizados na criação de elementos visuais nos jogos usam classes no C# as quais herdam de `GameObject`, portanto podemos adicioná-los em uma `List` sem problemas. Não confunda essa lista `prefabs_base` com as instâncias que serão criadas. A lista `prefabs_base` só armazenará dois objetos (`ColetavelCube` e `ColetavelCylinder`). As instâncias criadas não serão armazenadas em variáveis, elas somente existirão no hierarchy depois de criadas pelo script.

Salve seu script e, voltando ao Unity, certifique-se de que o `GameObject` `InstanciarPrefabs` está selecionado, vá até seu inspector e veja que o script adicionado nele agora tem três propriedades, cada uma referente a uma das variáveis criadas, veja na **Figura 17**.

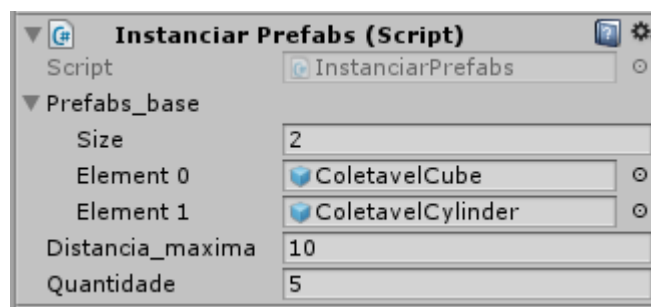
Figura 17 - Propriedades de InstanciarPrefabs.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Vamos alterar o valor dessas propriedades agora. Inicialmente vamos expandir a propriedade “Prefabs_base”, mostrando um item chamado “Size” o qual está com o valor zero. Modifique esse valor para 2, pois temos dois prefabs para usarmos como base (ColetavelCube e ColetavelCylinder). Verifique que, depois disso, dois novos campos foram mostrados, um para cada Prefab. Arraste o Prefab ColetavelCube da sua pasta para o primeiro campo e o Prefab ColetavelCylinder para o segundo campo. Após isso digite na propriedade Distancia_maxima o valor 10 e em Quantidade o valor 5. Isso indica que queremos criar cinco instâncias no total, sendo elas escolhidas aleatoriamente entre os prefabs ColetavelCube e ColetavelCylinder, todas em posições aleatórias, porém no máximo a uma distância de 10 unidades do centro do GameObject InstanciarPrefabs. A **Figura 18** mostra como as propriedades do script ficarão depois dessas mudanças.

Figura 18 - Script InstanciarPrefabs com as propriedades configuradas e com os dois prefabs adicionados na lista prefabs_base.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

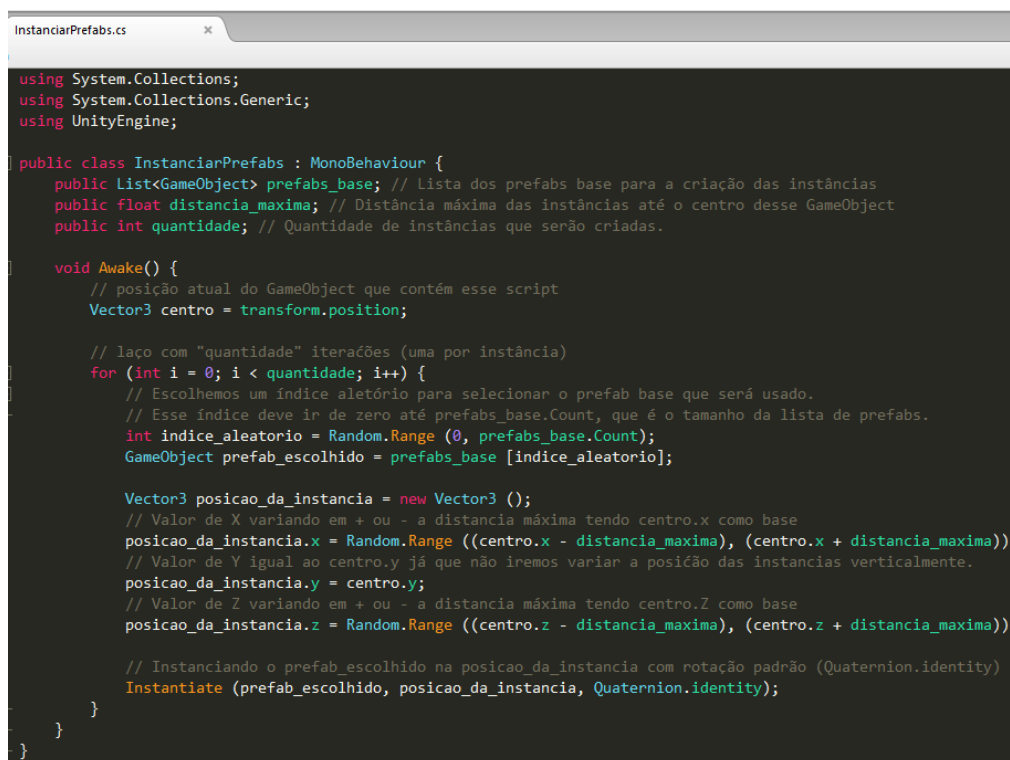
Edite novamente o script InstanciarPrefabs.cs e agora crie o método Awake() que será utilizado para instanciar os prefabs. O método Awake() é executado imediatamente antes do método Start(). Escolhemos esse método para criar as nossas instâncias, mas também poderia ser o Start() a funcionar normalmente. Nesse caso é apenas uma questão de preferência e organização, escrevemos o código que cria objetos iniciais no Awake e deixamos o Start para o código o qual considera que esses objetos já estão criados. Mas, repetindo, trata-se somente de organização de código, e isso, apesar de importante, é pessoal.

O método Awake() fará as seguintes ações:

1. Crie uma variável Vector3 chamada **centro** que terá a posição do GameObject ao qual o script está associado (InstanciarPrefabs).
2. Executará um laço **for** com uma variável **i** de 0 até a **quantidade** escolhida.
3. Para cada iteração do laço um dos Prefabs da lista prebabs_base será escolhido (aleatoriamente) e armazenado na variável **prefab_escolhido**.
4. Ainda dentro do laço deve ser criado um novo Vector3 chamado **posicao_da_instancia** que terá o valor X igual a um valor aleatório entre **(centro.x - distancia_maxima)** e **(centro.x + distancia_maxima)**. O valor de Y de **posicao_da_instancia** será o mesmo do centro, já que não queremos variar a posição das instâncias verticalmente. O valor Z é igual a um valor aleatório entre **(centro.z - distancia_maxima)** e **(centro.z + distancia_maxima)**.
5. No laço ainda usamos o **Instantiate** para criar o **prefab_escolhido** na **posicao_da_instancia** e sem nenhuma rotação. No Unity, para indicar que não queremos nenhuma rotação, usamos o comando **Quaternion.identity** (veremos mais sobre Quaternions e rotações em outras aulas).
6. Finalize o laço for.

O código que instancia os prefabs no Awake() deve ficar conforme a **Figura 19**.

Figura 19 - Script com o código final de instanciação aleatória de prefabs no método Awake().



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InstanciarPrefabs : MonoBehaviour {
    public List<GameObject> prefabs_base; // Lista dos prefabs base para a criação das instâncias
    public float distancia_maxima; // Distância máxima das instâncias até o centro desse GameObject
    public int quantidade; // Quantidade de instâncias que serão criadas.

    void Awake() {
        // posição atual do GameObject que contém esse script
        Vector3 centro = transform.position;

        // laço com "quantidade" iterações (uma por instância)
        for (int i = 0; i < quantidade; i++) {
            // Escolhemos um índice aleatório para selecionar o prefab base que será usado.
            // Esse índice deve ir de zero até prefabs_base.Count, que é o tamanho da lista de prefabs.
            int indice_aleatorio = Random.Range(0, prefabs_base.Count);
            GameObject prefab_escolhido = prefabs_base[indice_aleatorio];

            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));

            // Instanciando o prefab_escolhido na posicao_da_instancia com rotação padrão (Quaternion.identity)
            Instantiate(prefab_escolhido, posicao_da_instancia, Quaternion.identity);
        }
    }
}
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Salve o script, volte ao Unity e faça as seguintes mudanças para que seja mais fácil a visualização do resultado:

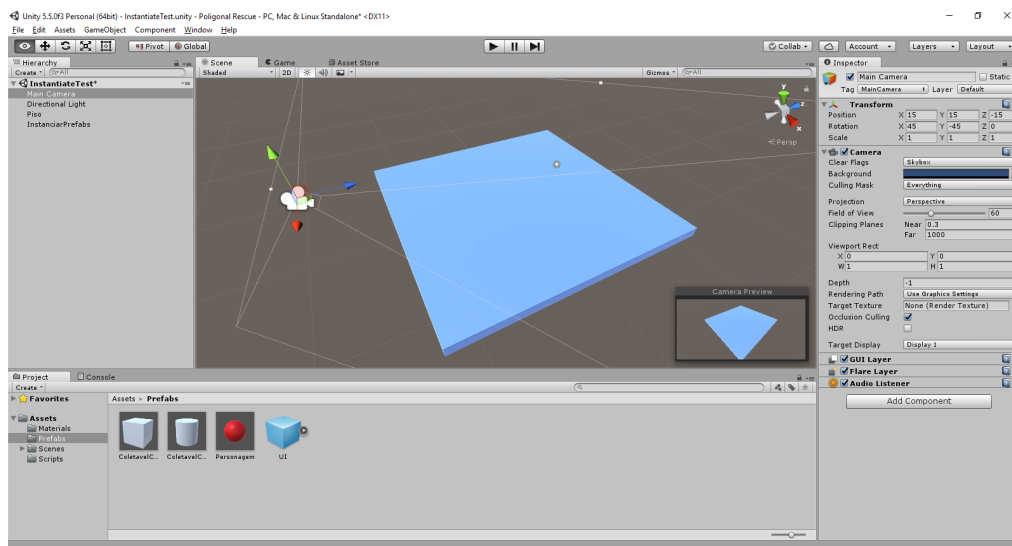
- Selecione o piso, reset sua posição e modifique sua escala para 25, 1, 25. Isso fará com que todas as instâncias fiquem sobre ele.
- Selecione a Main Camera e modifique a propriedade Position para 15, 15, -15 e a Rotation para 45, -45, 0. Assim ela deverá ficar de uma forma que o Camera Preview mostre todo o piso.
- Reposicione o GameObject InstanciarPrefab para a Position 0, 1.5, 0 para que fique no centro e um pouco acima do piso.
- Caso tenha criado essa cena com os objetos diferentes do recomendado, durante o texto realize os ajustes necessários para o seu caso. O importante é que a câmera mostre todo o piso no Preview.

Atenção!

Se você prestou atenção usamos o método `Random.Range()` para criar valores aleatórios entre um valor mínimo e máximo. O **`Random.Range(int min, int max)`** retorna um número float entre o parâmetro min (incluindo ele) e o parâmetro max (excluindo ele), ou seja, esse valor pode ser igual a min por coincidência, mas sempre será menor que max. O uso do Random associado ao Instantiate é muito útil quando queremos criar experiências distintas todas as vezes que um jogador visita uma cena do nosso jogo.

Se você realizou esses ajustes, sua cena deve estar como na **Figura 20**.

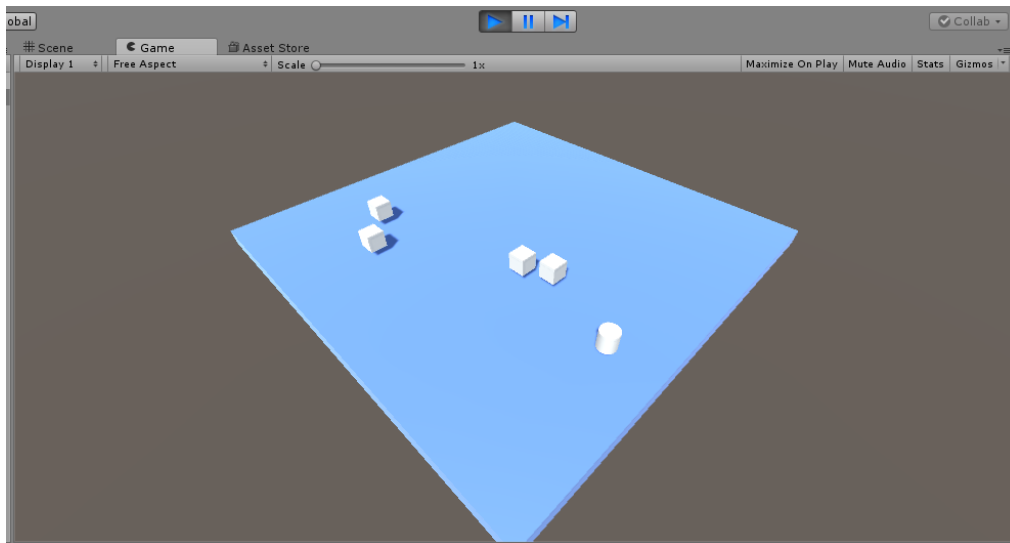
Figura 20 - Cena após mudanças de posicionamento, escala e rotação de seus GameObjects.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Execute agora o jogo e se tudo foi feito corretamente você deverá ver cinco instâncias dos Prefabs criados sobre o piso em posições aleatórias. Essas instâncias podem ser tanto Cubos como Cilindros, pois isso também é escolhido aleatoriamente. Veja o resultado na **Figura 21**.

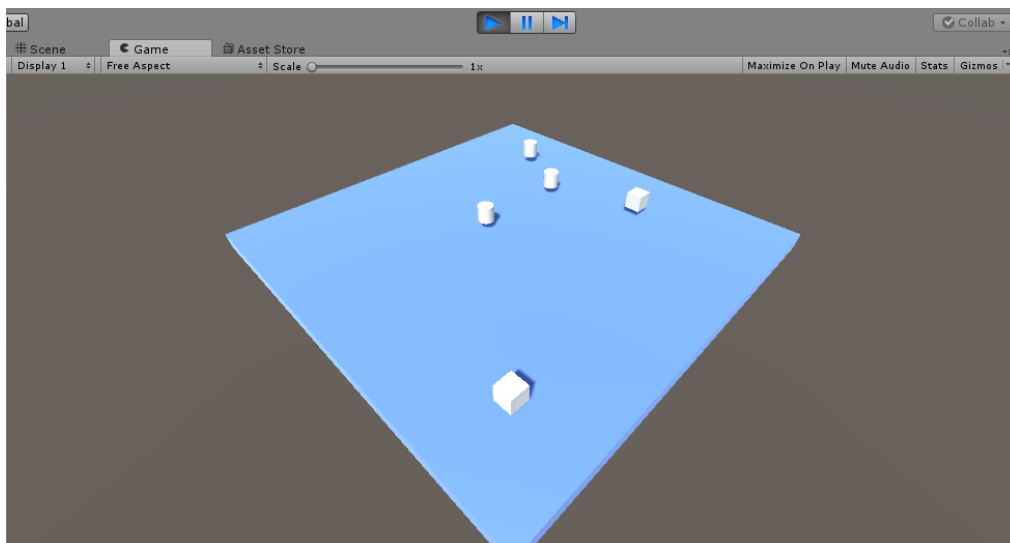
Figura 21 - Instâncias dos prefabs posicionados aleatoriamente sobre o piso usando o nosso script InstanciarPrefabs.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Pare o jogo e execute novamente para observar que o resultado será diferente. Ver **Figura 22**.

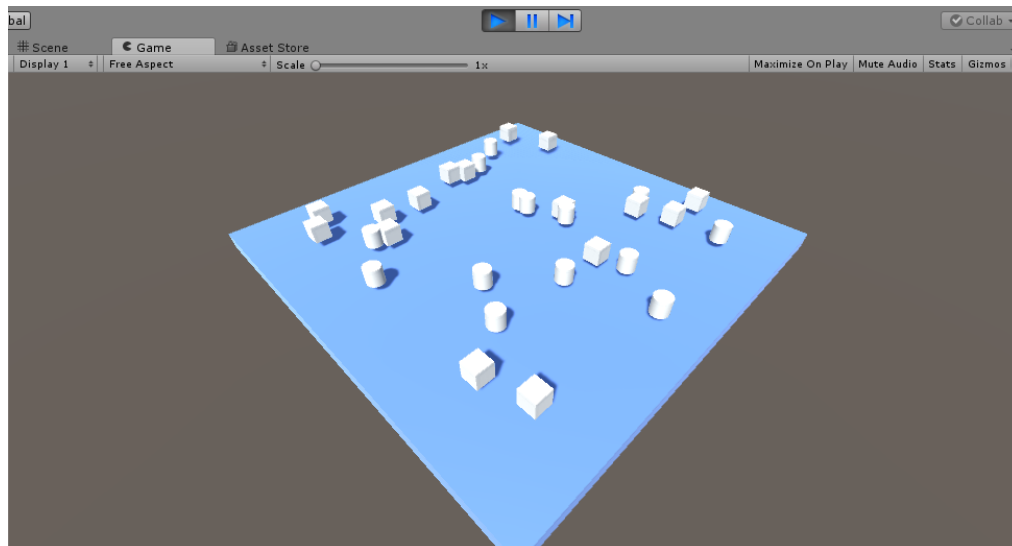
Figura 22 - Outra execução mostra um resultado diferente na posição das instâncias.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Para o jogo, selecione InstanciarPrefabs e altere as propriedades do script modificando Quantidade de 5 para 30. Rode novamente e observe que agora temos várias (30) instâncias criadas na cena. Veja um exemplo na **Figura 23**.

Figura 23 - 30 instâncias criadas dinamicamente através do script InstanciarPrefabs.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: <https://unity3d.com/pt/>. Acesso em: 14 de março de 2017.

Nesta aula, aprendemos a criar Prefabs, a partir de GameObjects da cena, e Instâncias do mesmo tanto no editor, arrastando-os para a cena através de scripts com o Instantiate. Para isso criamos um script que pode receber uma lista de Prefabs, uma distância máxima e uma quantidade, além de instâncias desses Prefabs de modo aleatório ao redor do GameObject ao qual ele está associado, respeitando a distância máxima e quantidade escolhidas. Esse script é bem genérico e serve para qualquer tipo de Prefabs que você tenha, então ele também pode ser utilizado em outros jogos.

Criamos nossos experimentos com o Instantiate em uma nova cena, mas se desejar pode usá-lo no labirinto do seu jogo normalmente, basta criar um GameObject vazio e adicionar esse script criado nele, configurando os Prefabs base, quantidade e distância, como vimos.

Estamos chegando ao fim da nossa aula 6. Parabéns pelos avanços conquistados no decorrer das aulas. Estamos conseguindo desenvolver bem todas as etapas e fases. Na próxima aula aprenderemos a criar novas fases para o nosso jogo para torná-lo mais interessante e menos curto, não é mesmo? Até lá!

Resumo

Hoje aprendemos a criar objetos reutilizáveis, chamados de Prefabs, com as definições dos tipos de polígonos existentes no jogo *Polygonal Rescue*. Aprendemos também a criar novas instâncias desses Prefabs nas cenas, tanto manualmente como a partir de scripts, com o uso do método `Instantiate`. Criamos também um sistema de instanciação aleatório de Prefabs o qual facilita a criação de zonas nas quais os Prefabs aparecem em locais distintos a cada execução do jogo.

Leitura Complementar

O uso de Prefabs não só é importante como também é considerado crucial para a criação de um jogo que utiliza as boas práticas de reusabilidade no Unity. Seguem alguns links para você estudar mais sobre o assunto:

- <https://docs.unity3d.com/Manual/Prefabs.html>
- <https://unity3d.com/pt/learn/tutorials/topics/interface-essentials/prefabs-concept-usage>
- <https://docs.unity3d.com/Manual/InstantiatingPrefabs.html>
- <https://docs.unity3d.com/ScriptReference/Object.Instantiate.html>

Autoavaliação

1. Como criar um Prefab a partir de um `GameObject` criado na cena?
2. `GameObjects` complexos com vários filhos podem ser transformados em Prefabs da mesma maneira que os mais simples?
3. Como você aplica a mudança de uma instância de um Prefab no Prefab em si?

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].

K. Aava Rani. 2014. Learning Unity Physics. Packt Publishing

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].