Raycast, Instantiate, Destroy e Prefab

Flávio Roberto Dias Silva

Encontro 4
Desenvolvimento de Jogos Eletrônicos com Unity 3D.

Universidade Estadual do Oeste do Paraná

O último encontro terminou com o desafio de criar um pulo para nosso personagem.

Tentamos utilizar a propriedade

```
CharacterController.isGrounded
```

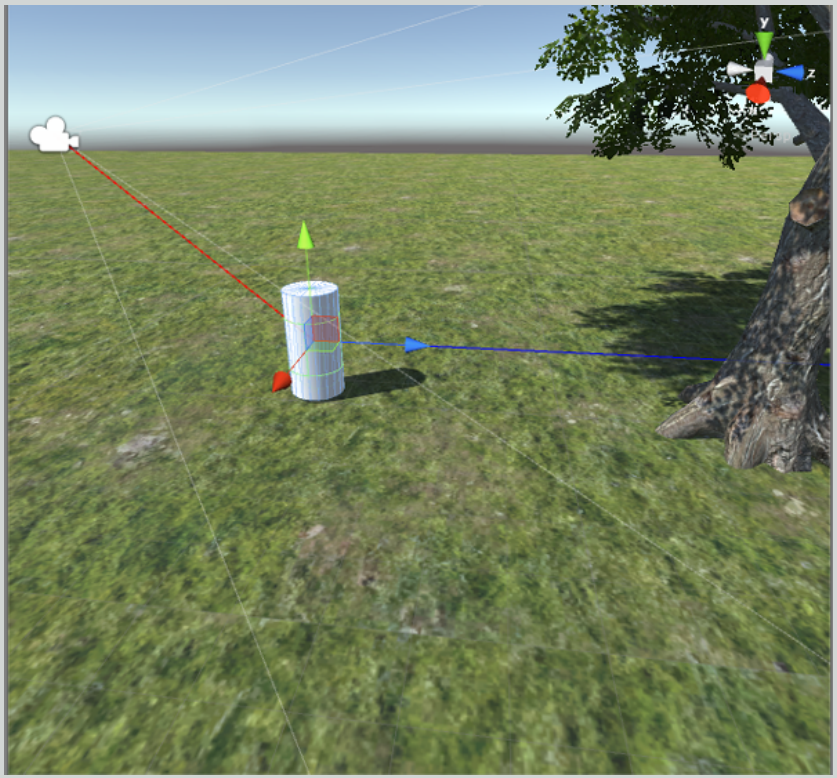
mas essa propriedade se mostrou ineficaz para nosso intuito de verificar se o personagem estava no chão.

Para resolver o problema utilizamos a bibliotecas de físicas `Physics` e a função `Raycast`.

Vamos estudar mais a função `Raycast`

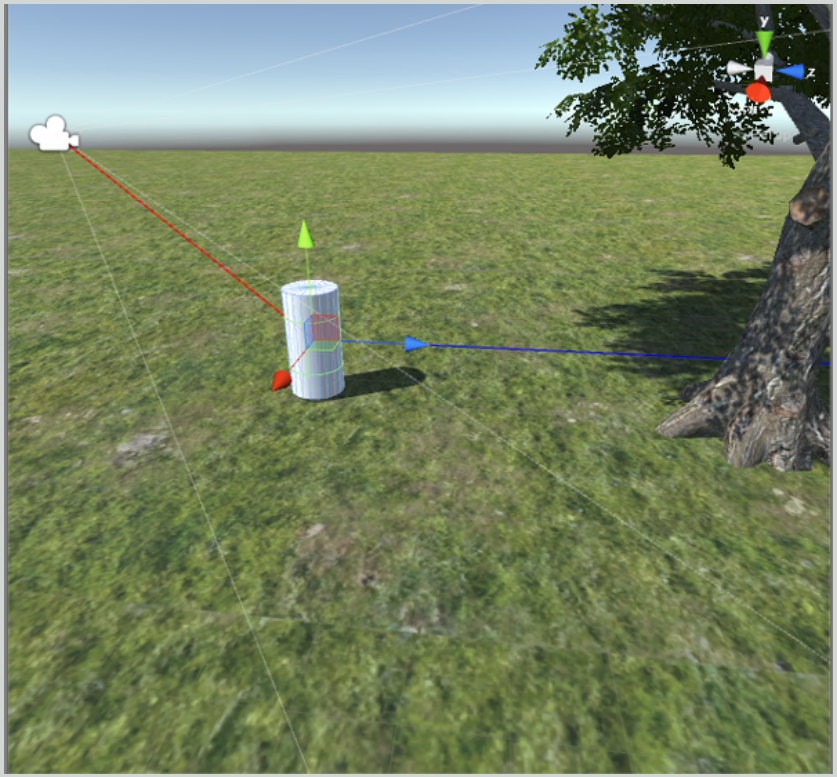
A função `Physics.Raycast(...)` lança um raio invisível no mundo 3D retornando verdadeiro se esse raio tocar em algo que tenha um `Collider` e falso caso contrario.

Para fazer o pulo do personagem utilizamos um raycast para baixo para verificar se o personagem está no chão.



Na imagem ao lado utilizamos `Debug.DrawLine(...)` para exibir as linhas vermelha e azul que podem ser entendidas como `Raycast`.

A linha vermelha simboliza um raio emitido da camera em direção do personagem e a linha azul um raio emitido do personagem na sua direção `forward`.



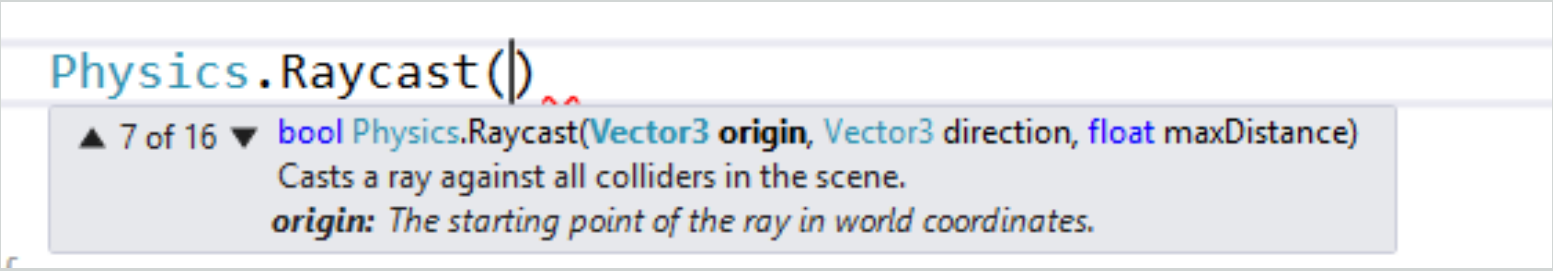
No caso vermelho a função `Raycast` retornaria verdadeiro pis o raio toca o personagem.

No caso azul teríamos um verdadeiro pois o raio toca uma arvore.

A função `Raycast` se utiliza de um recurso de programação orientada a objetos chamado sobrecarga, isso é, existem várias definições para a função de nome `Raycast` mas com parâmetros diferentes.

No total `Physics.Raycast` conta com 16 formas de inserção de parâmetros.

Destacarei alguns deles, para os demais, existe uma referencia no *Asana* que discute todas as 12 dessas 16 definições que eram as que existiam na época do vídeo.



A primeira definição que vamos mencionar é a que pede **origem, direção e distancia máxima**.

Essa é a definição usada no script em anexo no *Asana* que tem uma função para o pulo do personagem.

Origem é o ponto de onde parte o raio

Direção é o vetor que aponta para onde o raio se deslocará, esse vetor será normalizado pela função.

Distancia máxima é a distancia que o raycast se deslocará procurando uma colisão.

```
RaycastHit hit = new RaycastHit();
Physics.Raycast()
```

▲ 8 of 16 ▼ bool Physics.Raycast(Vector3 **origin**, Vector3 direction, out RaycastHit hitInfo)

Como pode ser observado essa definição leva em consideração também origem e direção, mas não tem distancia máxima. Isso quer dizer que o raio vai se prolongar "infinitamente" pelo espaço 3D até encontrar um ponto de colisão.

Essa segunda definição pede uma outra variável de um tipo pertencente a biblioteca `UnityEngine` esse tipo é o `RaycastHit`.

A variável do tipo `RaycastHit` guarda informações sobre o ponto de colisão, podendo nos dizer quem é o gameObject que foi tocado pelo raio, a direção normal da colisão, o ponto no espaço 3D o qual aconteceu a colisão, entre muitas outras.

Para acessar essas informações devemos fazer

```
RaycastHit hit = new RaycastHit ();

if(Physics.Raycast(vetorOrigem,vetorDirecao,out hit))
{
  hit.point;//ponto de colisão
  hit.normal;//vetor normal a colisão
  hit.gameObject;//objeto tocado pela colisão
}
```

Exercício

Exercício: Crie uma nova cena chamada exercícios de Raycast.

Nessa cena espalhe alguns objetos num terreno, por exemplo: Cubo, esfera, capsula, cilindro.

Crie um personagem com uma das movimentações das aulas anteriores [Obs: Você não precisa criar um script de movimentação, pode usar os scripts das aulas anteriores].

Utilize a função raycast para lançar um raio nos objetos em cena dizendo o nome de cada objeto.

Na maioria dos jogos que imaginarmos existem elementos que são inseridos na fase em tempo de jogo, sejam eles inimigos e armadilhas ou simplesmente efeitos de luz, partículas, explosões e etc...

Para inserir elementos em tempo de jogo na Unity utilizamos uma função estática da classe `MonoBehaviour` chamada `Instantiate`.

Parâmetros

A função Instantiate da versão 5.3.5f1 da Unity contam com três formas de ser chamada, falaremos aqui apenas da mais utilizada delas que conta com três parâmetros.

```
Instantiate(gameObject,posicaoNoMundo,rotacaoDoObjeto);
```

O primeiro parâmetro é do tipo GameObject. Ele se refere ao objeto que você quer instanciar, devemos preencher uma variável do tipo `GameObject` com o objeto que vai aparecer no nosso jogo.

O segundo parametro é do tipo `Vector3`. Ele se refere a posição onde nosso objeto irá aparecer no mundo.

O terceiro parametro é do tipo `Quaternion`. Ele se refere a rotação em que nosso objeto aparecerá no mundo.

A função `Instantiate` retorna uma referencia do nosso `GameObject` no tipo primitivo de **Object**. Esse Objeto pode ser convertido para `GameObject` utilizando a função padrão de conversão do C#.

```
GameObject G = (GameObject)Instantiate( umGameObject,
                                        umaPosicao,
                                        umaRotacao);
```

Depois de referenciado como `GameObject` podemos manipular a instancia do nosso objeto acessando as propriedades do gameObject. Por exemplo:

```
G.transform.position;
G.GetComponent<UmComponente>();
G.name;
G.tag;
```

Exercício

Exercício: Utilize a cena do Raycast para instanciar novos objetos. Para isso, crie referencias dos objetos em cena. Por exemplo:

```
GameObject[] G = new GameObject[4];
G[0]= GameObject.Find(“Cube”);
G[1]= GameObject.Find(“Sphere”);
G[2]= GameObject.Find(“Capsule”);
G[3]= GameObject.Find(“Cylinder”);
```

Tente utilizar algum botão para instanciar os gameObjects.

Destroy

Da mesma forma que precisamos inserir um objeto em cena precisamos destrui-lo, retira-lo da cena de jogo, exemplos de aplicações desse recurso são ilimitados:

- Um inimigo derrotado teria de desaparecer da cena.
- Um projetil deveria ser destruído depois de acertar seu alvo.
- Um efeito de partícula deveria ser destruído depois de feito o seu papel visual.

Para fazer esse papel dentro do Unity utilizamos a função estática da classe `MonoBehaviour` chamada `Destroy`.

A função `Destroy` da versão 5.3.5f1 tem quatro formas de ser chamada (ela tem sobrecarga).

A primeira tem um parâmetro que é do tipo `GameObject`

```
Destroy(varDoTipoGameObject);
```

A segunda forma utiliza dois parâmetros, o primeiro continua sendo um `GameObject` e o segundo é do tipo **float**.

```
Destroy(varDoTipoGameObject, tempoParaSerDestruido);
```

A segunda forma agenda um tempo em segundos para o gameObject ser destruído. Muito útil, entre outras coisas, para destruir partículas, já agendado a destruição no momento da instancia.

A terceira e a quarta forma são variantes das duas primeiras mas em vez de usar um parâmetro do tipo `GameObject` utilizamos um parâmetro do tipo `Component`. São elas:

```
Destroy(variavelDoTipoComponente);
```

```
Destroy(outraVarDoTipoComponente, tempo);
```

Essa função tem grande utilidade para remover scripts e outros componentes que não precisam mais ser utilizados ou não podem mais se manter ativos para o prosseguimento do jogo.

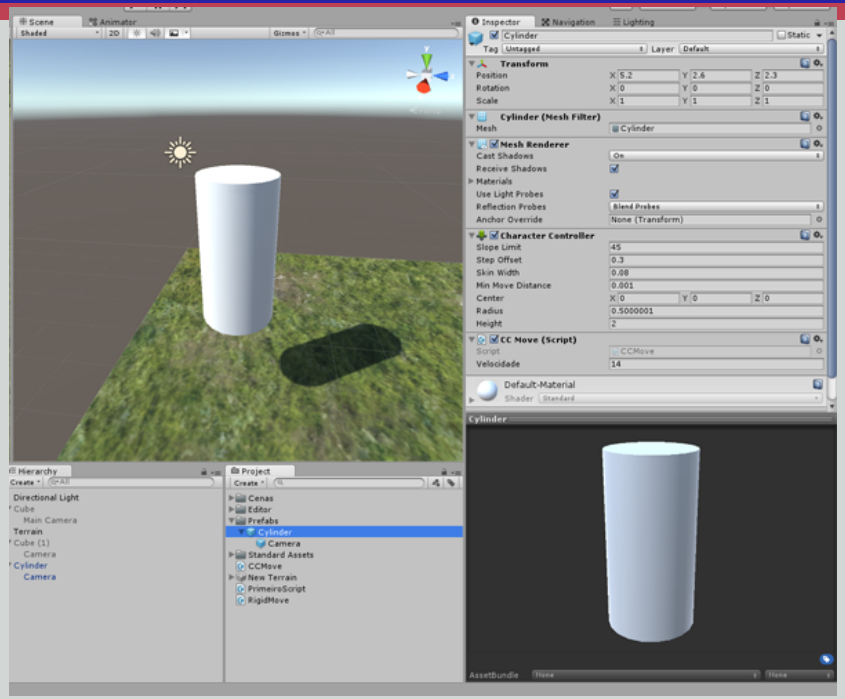
Exercício:

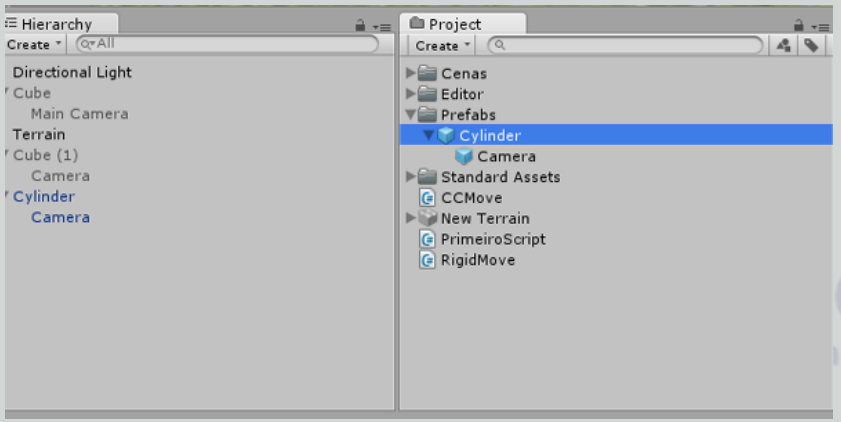
Exercício: Agende um tempo para a destruição dos objetos instanciados no exercício anterior.

Exercício: Destrua um componente de um gameObject.

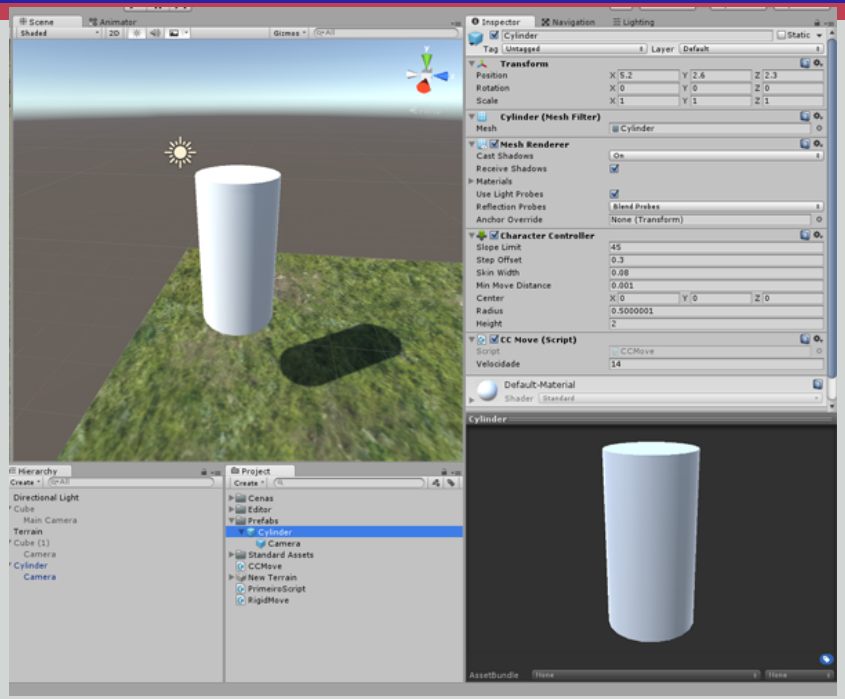
Prefabs: Em Unity Prefab indica um objeto de jogo pré fabricado, isso é, pré configurado para entrar em cena com as propriedades, componentes e scripts desejados.

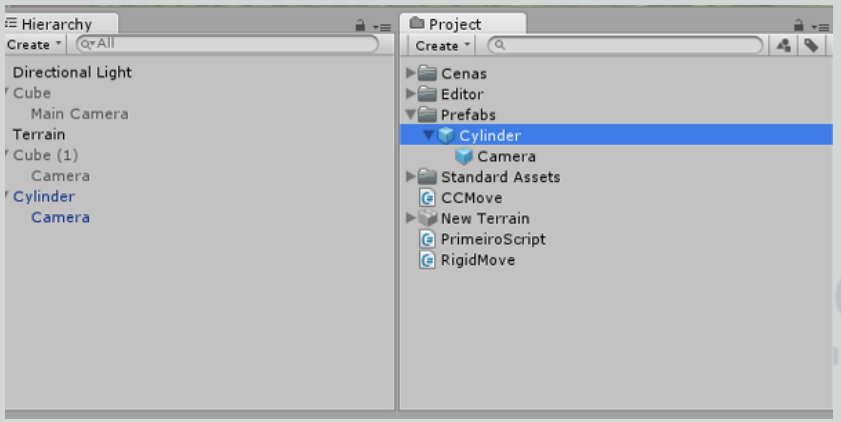
Para criar um prefab, é muito simples, você pode criar na aba cena um GameObject com as propriedades desejadas e logo após arrasta-lo para a aba Project.





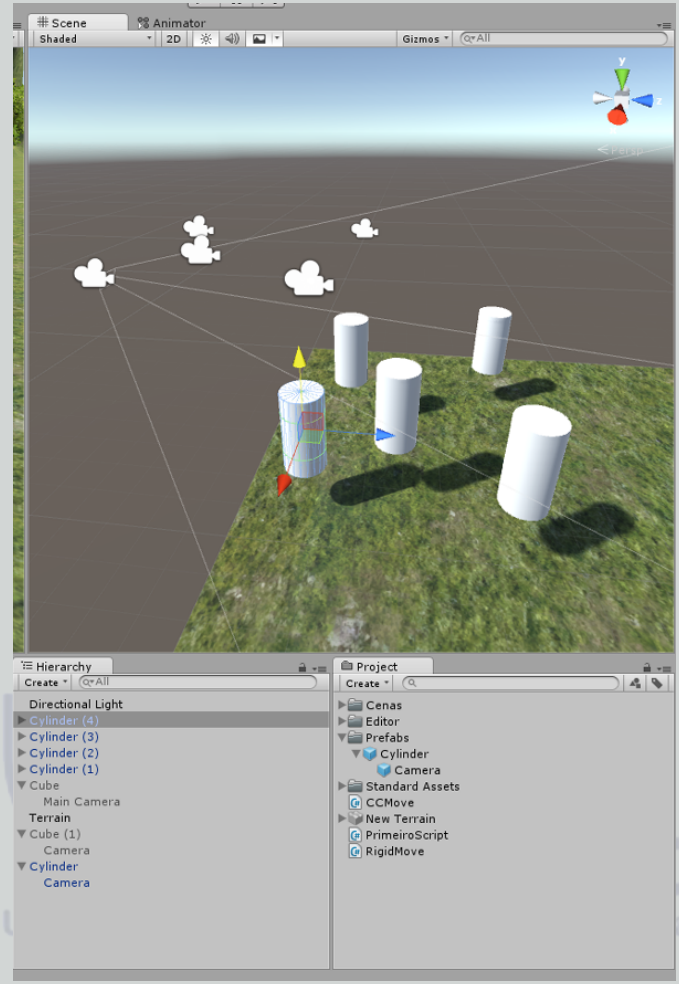
Observe que o “Cylinder” na aba Hierarchy que se refere ao prefab da aba Project é exibido em fonte azul.





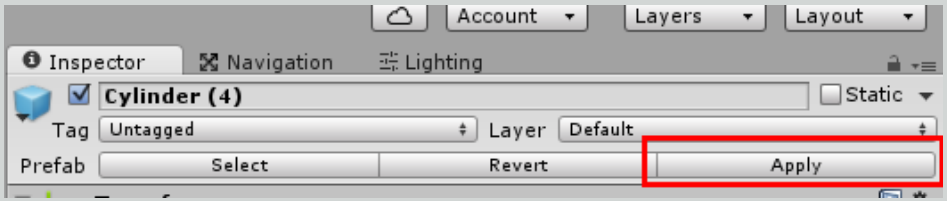
Interessante e importante:

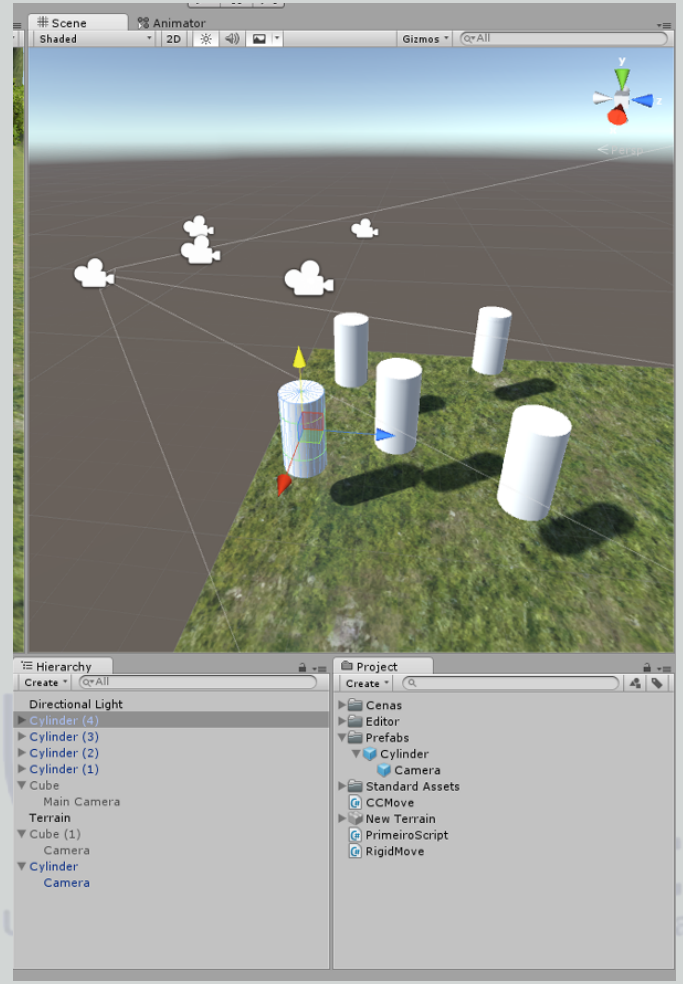
1: Podemos arrastar o Prefab da aba Project para a aba Hierarchy ou a aba Scene, criando assim uma outra cópia/referencia para o prefab.



Interessante e importante:

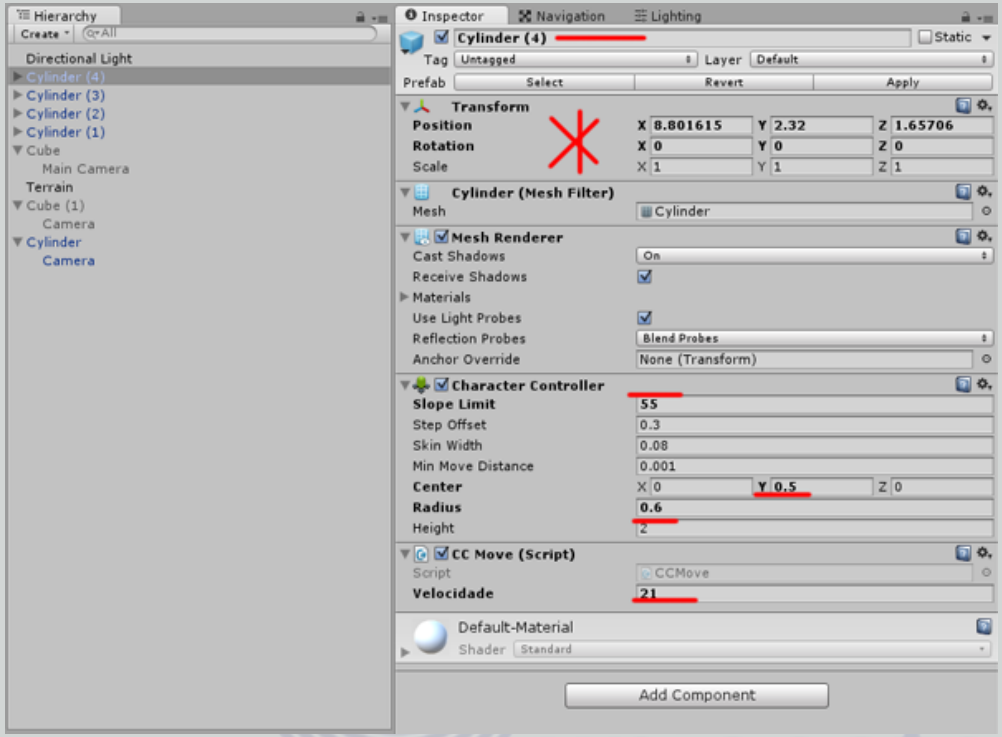
2: Modificando as propriedades no Prefab e clicando em aplicar da aba inspector do Prefab as modificações são aplicadas automaticamente nas cópias.





Interessante e importante:

3: As propriedades modificadas numa cópia do Prefab que não constam no Prefab aparecem em negrito no Inspector.



Interessante e importante:

4: Prefabs podem ser utilizados pela função Instantiate via referencias de variáveis públicas, isso é, podemos instanciar um objeto sem a necessidade que exista uma cópia dele já em cena.

Para isso devemos criar uma variável

```
public GameObject G;
```

e preencher seu valor via Inspector.

Assim utilizar a variável `G` com a função `Instantiate`.

Exercício

Exercício: Utilize alguma forma 3D da Unity para criar um projetil.

- No modelo do projetil coloque um script que faça esse se movimentar;
- Em seu personagem faça uma função para instanciar o projetil com o pressionar de um botão.
- Faça com que o projetil seja destruído após alguns segundos[use `Time.deltaTime`].
- Tente verificar se o projetil acerta algum objeto [Apesar de não recomendado tente utilizar `Physics.Raycast(...)`].

[1] Barnes, D. J., Kölling, M.(2009), Programação Orientada a Objetos com Java. Uma introdução prática usando BLUEJ, 4ªed., Pearson Prentice Hall.

[2] Battaiola, A. L. (2000). Jogos por Computador ? Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação In: XIX Jornada de Atualização em Informática. Curitiba: SBC, Julho/2000, v. 2. pp. 83 - 122

[3] Battaiola, A. L.; Elias, N. C.; Domingues, R.G. et al (2002). Desenvolvimento de um Software Educacional com Base em Conceitos de Jogos de Computador In: XIII Simpósio Brasileiro de Informática na Educação. São Leopoldo: SBC, 2002, pp. 282-290.

[4] Crua, E. W. G.; Bittencourte, J. R.(2005) Desenvolvimento de Jogos 3D: Concepção, Design e Programação. Anais da XXIV Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, pp.1313-1356, São Leopoldo, Brasil, Julho de 2005. Deitel, H. M., Deitel, P. J.(2010), Java: Como programar, 8ªed., Pearson Prentice Hall. Rio de Janeiro IMPA.

[5] Freeman, E., Freeman, E.(2007), Use a Cabeça Padrões de Projetos, 2ªed., Rio de Janeiro Altabooks.

[6] Sintes, A.(2002), Aprenda Programação Orientada a Objetos em 21 dias, São Paulo Makron Books.

[7] Stellman, A.; Greene, J.(2011), Use a Cabeça! *C#*, Rio de Janeiro, AltaBooks.

[8] Unity Tecnologies (2016)(A). Unity 3D User Manual [online]. Disponivel em: [http://docs.unity3d.com/Manual/index.html][Acesso em 18/04/2016]

[9] Unity Tecnologies (2016)(B). Unity 3D Community Forum [online]. Disponivel em: [http://forum.unity3d.com/][Acesso em 18/04/2016]

[10] Unity Tecnologies (2016)(C). Unity 3D Online Tutorials [online]. Disponivel em: [https://unity3d.com/pt/learn/tutorials][Acesso em 18/04/2016]

[11] Unity Tecnologies (2016)(D). Unity 3D Community Wiki [online]. Disponivel em:

[http://wiki.unity3d.com/index.php/Main_Page][Acesso em 18/04/2016]