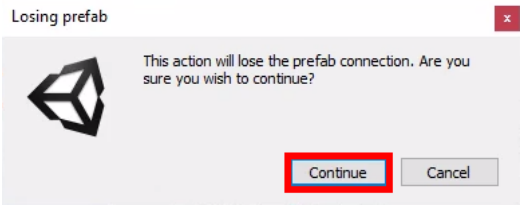
10

Atirando

Transcrição

Anteriormente, configuramos a bala. Agora, faremos a heroína atirá-la, por meio do *script*, que deve ser arrastado a "Jogador" para que isso aconteça, considerando que a arma possui difícil acesso em "Hierarchy" e todos os *scripts* estão nele. Como a arma está presa a "Jogador", o *script* que inserirmos nele, funcionará nela.

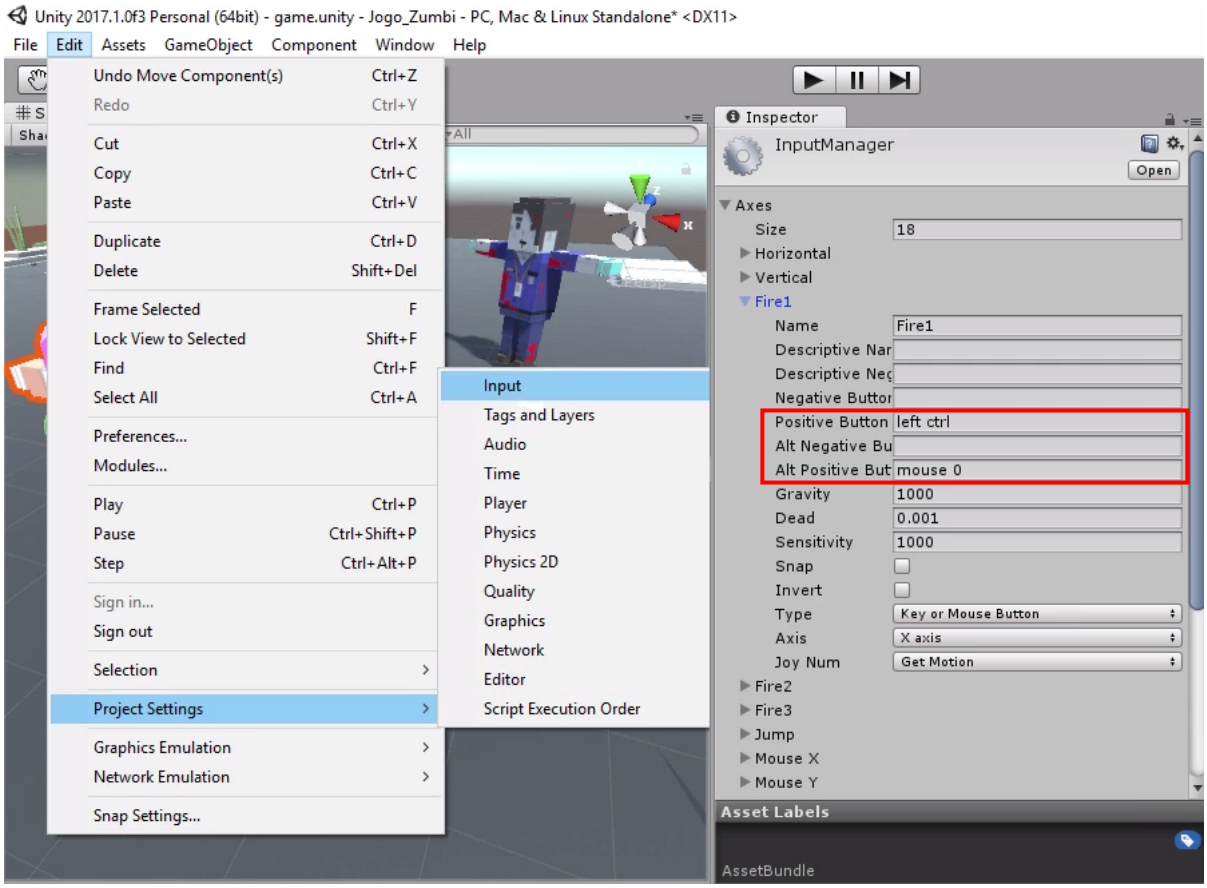
Em "Project > Assets > Script", clicaremos com o botão direito do mouse e selecionaremos "Create > C# Script" para adicionar um *script*, que chamaremos de "ControlaArma" e o arrastaremos para o "Inspector" de "Jogador". Abrirá um aviso de que perderemos o *Prefab*, no qual clicaremos em "Continue".



Após arrastarmos, em "Inspector", clicaremos em "Apply". Assim, não perderemos nenhum "Prefab". Abriremos "ControlaArma" e desenvolveremos um código para que a arma dispare a bala com o clicar botão do mouse.

Para calcular o clique do botão do mouse, dentro de `Update()` — método que está sempre rodando — adicionaremos `if`, pois **se** o usuário utilizar o `Input` correspondente ao botão do mouse (`Get.ButtonDown`), especificando o botão (`Fire1`) entre aspas (`"`) por se tratar de um texto e entre parênteses (`()`).

A especificação de `Fire1` vem de "Edit > Project Settings > Input". Em "Inspector", no mesmo local em que encontramos "Vertical", encontraremos "Fire1", que pode ser acessado por meio do botão do mouse `0` — clique normal — ou por "Ctrl".



De volta ao código, se o usuário clicar o botão do mouse, especificaremos que a personagem atirará. Considerando que ela irá atirar diversas vezes, teremos que criar balas. Faremos isso por meio de `Instantiate()`, função da Unity que **cria novos objetos**. Nos parênteses dessa função, indicaremos:

- o objeto que queremos criar;
- a posição na qual será criado;
- a rotação do objeto.

Ainda não temos um objeto no código. Precisamos criá-lo por meio de uma variável pública (`public GameObject`), considerando que ela será visível em "Inspector", e nomearemos como `Bala` .

Declarada a variável, iremos colocá-la entre o parênteses de `Instantiate` , seguida de:

- vírgula (`,`);
- posição do "Jogador" (`transform.position`);
- rotação do "Jogador" (`transform.rotation`).

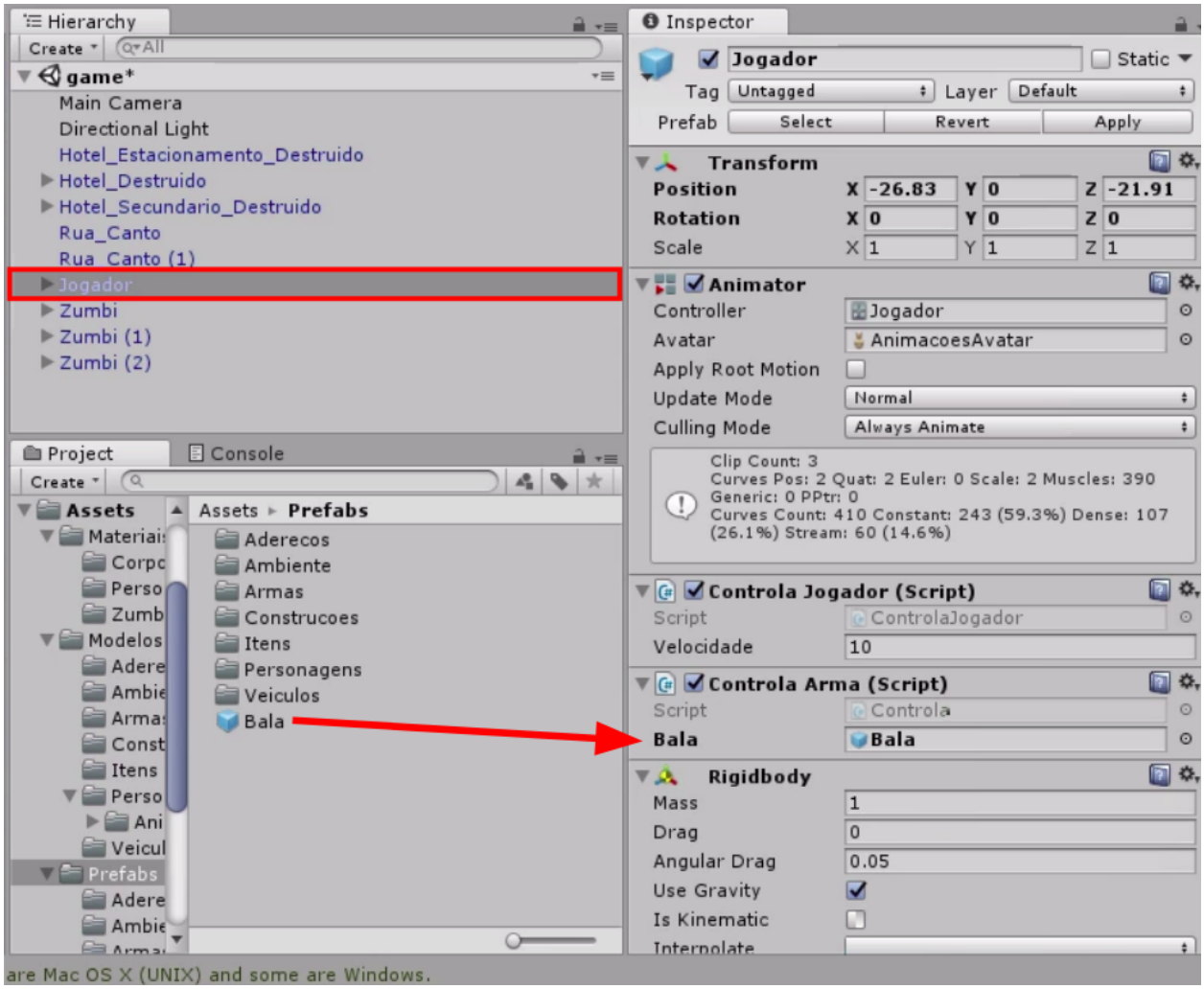
Salvaremos e minimizaremos o *script* `ControlaArma.cs` da seguinte forma:

```
public class ControlaArma : MonoBehaviour {

    public GameObject Bala;

    // Use this for inicialization
    void Start () {

    }
```

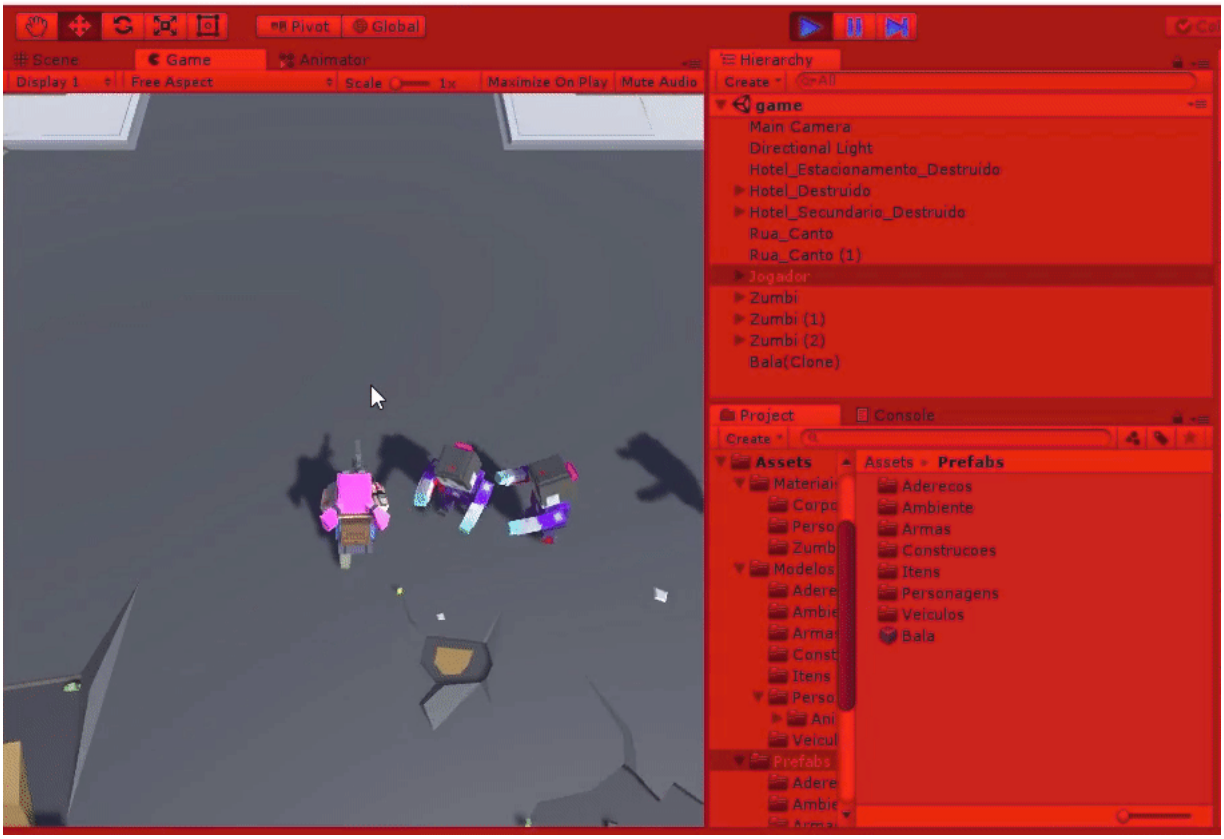
```
    // Update is called once per frame
    void Update () {
        if(Input.GetButtonDown("Fire1"))
        {
            Instantiate(Bala, transform.position, transform.rotation);
        }
    }
}
```

Voltaremos à Unity e, considerando que a bala deverá aparecer quando for atirada, apagaremos a que está em exibição, próximo à arma.

Para que as balas sejam criadas e atiradas, arrastaremos "Bala", de "Prefab", para "Controla Arma (Script)", no "Inspector" de "Jogador".

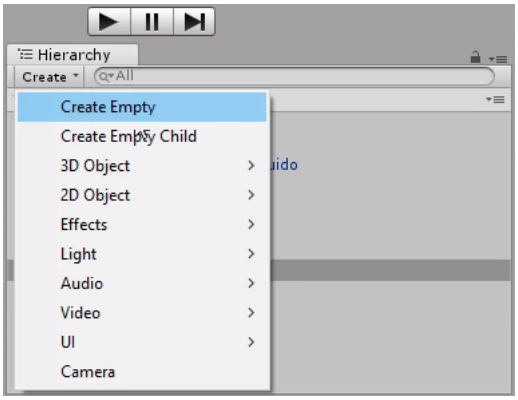


Assim, não precisamos do objeto no cenário para criá-lo. Pressionaremos "Play" e veremos que ao mover a heroína e clicar no mouse para atirar as balas, elas ficam no chão, formando um rastro.



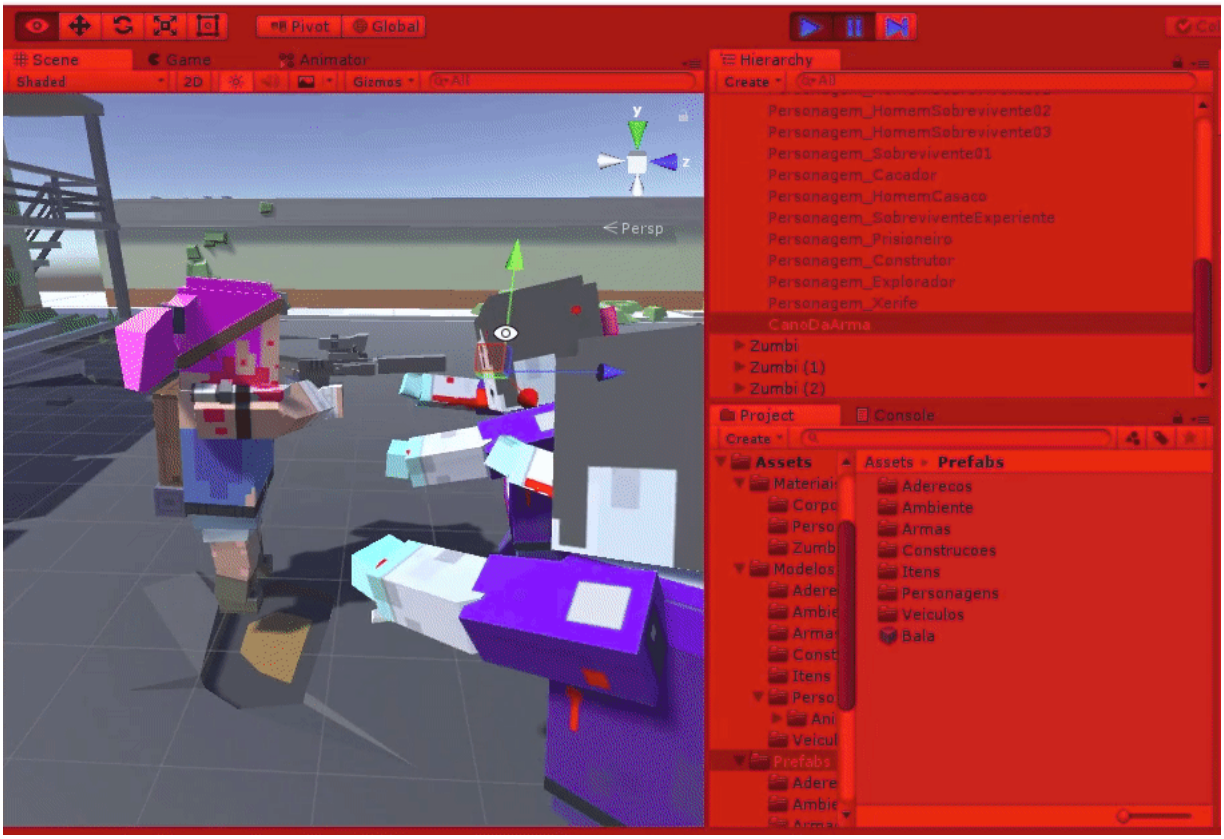
Isso acontece porque as posicionamos de acordo com a posição e rotação do "Jogador", e o *gizmo* de movimentação está nos pés dele. Por isso, as balas são criadas e ficam no chão. Para corrigir, precisamos posicioná-las no cano da arma.

Assim, criaremos um **objeto de referência** — com função única de marcar a posição do cano da arma — para que as balas sejam criadas nela, clicando em "Hierarchy > Create > Create Empty".

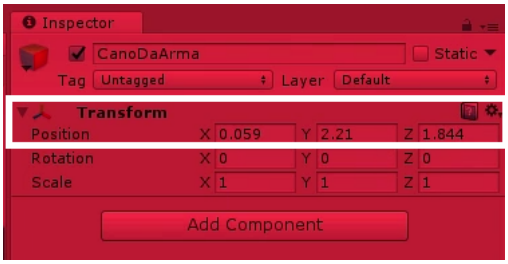


Ao criá-lo, em "Hierarchy" aparecerá como "GameObject" e em "Scene", aparecerá somente com a posição, sem arte. Utilizaremos esse objeto para marcar a posição da bala. Nomearemos como "CanoDaArma" e o arrastaremos para "Jogador", em "Hierarchy", pois quando um objeto está dentro do outro, segue os movimentos daquele em que foi inserido.

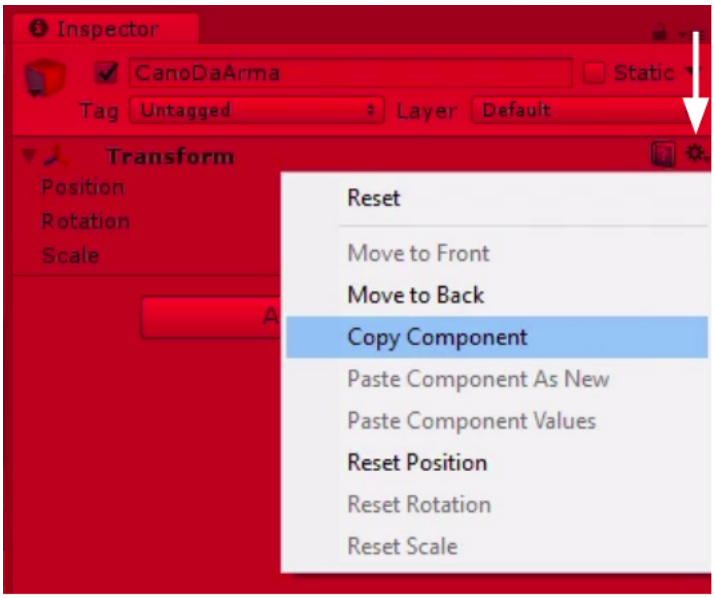
Para encontrar a posição exata em que queremos colocar as balas, ativaremos "Play" e "Pause". Ao aplicarmos Zoom, ajustaremos exatamente na altura do cano da arma. Posicionaremos a uma certa distância do cano para dar a ideia de que está saindo dele.



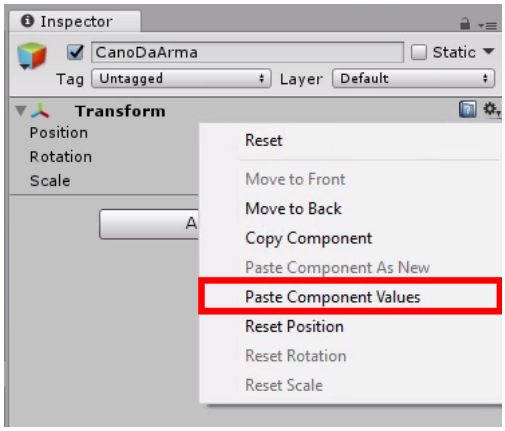
Mas, como ressaltamos anteriormente, ao desativarmos o "Play", todas as alterações que fizemos com ele ativado serão descartadas. Por isso, antes de desativar, anotaremos os valores dos três eixos de "Position", em "Transform".



Assim, teremos a posição exata que queremos. Mas não copiaremos anotando os valores no papel. Para copiar, clicaremos na engrenagem no canto superior direito e selecionaremos a opção "Copy Component".

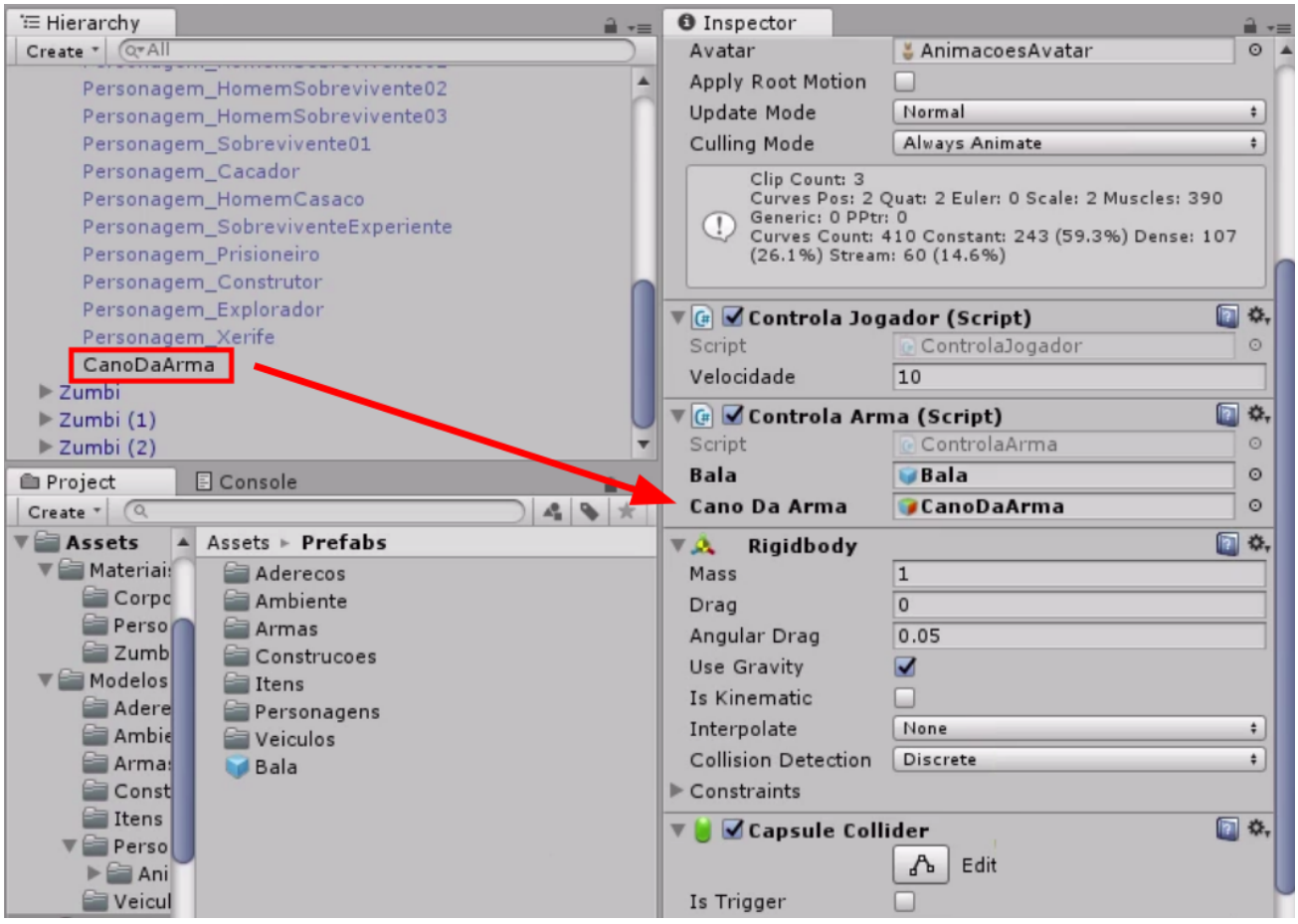


Ao desativarmos "Play", as alterações que fizemos serão descartadas. Clicaremos novamente na engrenagem e selecionaremos a opção "Paste Component Values".



Em "Scene", veremos que o objeto está na posição que definimos com "Play" ativado. Lembrem-se dessa dica, pois ela é muito útil.

Criamos a posição em que as balas serão criadas, com base no cano da arma. Porém, não criamos esse objeto no *script*. Então, declararemos uma variável pública (`public`) de `CanoDaArma` . Salvaremos e minimizaremos o editor de texto. De volta à Unity, arrastaremos "CanoDaArma" de "Hierarchy" para "Controla Arma (Script)" em "Inspector".

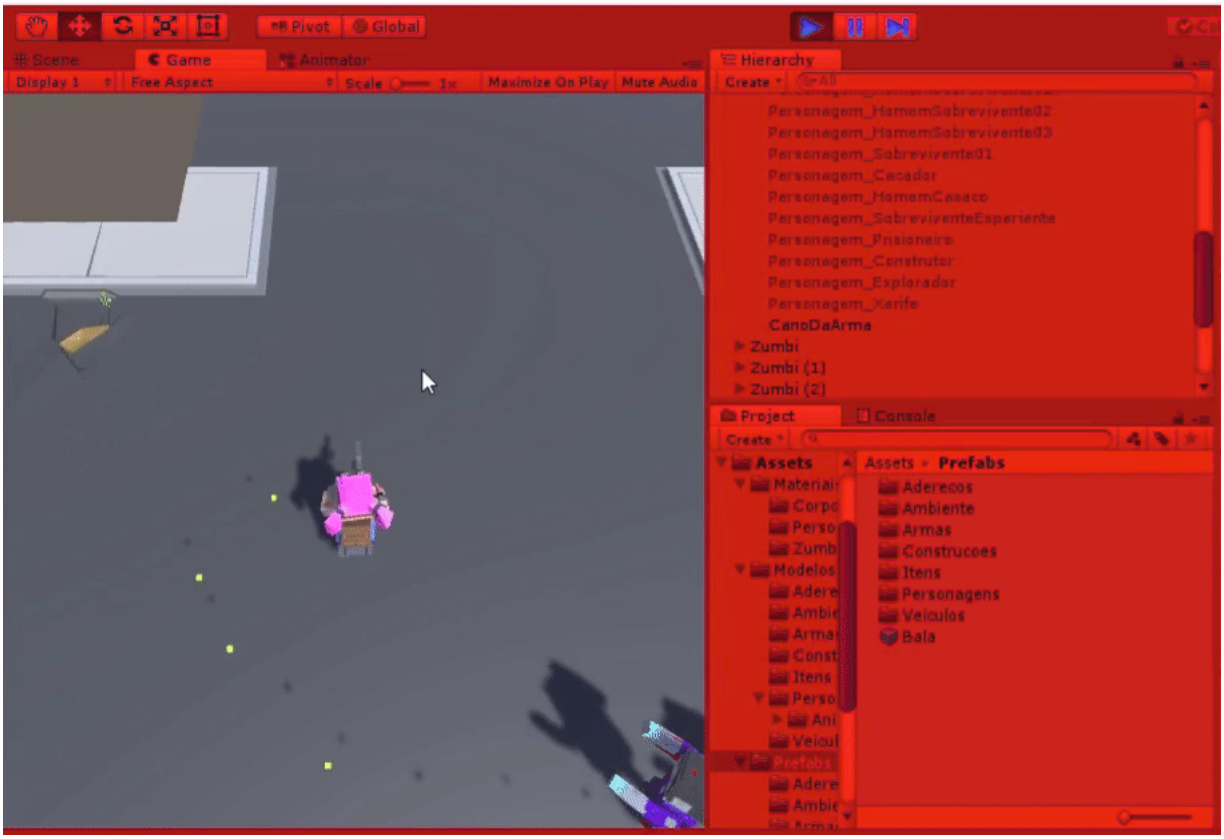


E, no código, adicionaremos `CanoDaArma` antes de `transform.position` e `transform.rotation` . O código ficará da seguinte forma:

```
public class ControlaArma : MonoBehaviour {

    public GameObject Bala;
```

```
    public GameObject CanoDaArma;

    // Use this for inicialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if(Input.GetButtonDown("Fire1"))
        {
            Instantiate(Bala, CanoDaArma.transform.position, CanoDaArma.transform.rotation);
        }
    }
}
```

Salvaremos e minimizaremos o editor de texto. Na Unity, ao ativarmos "Play", veremos que as balas são criadas na altura certa. Falta fazermos elas voarem, pois elas saem da arma e ficam paradas no ar.



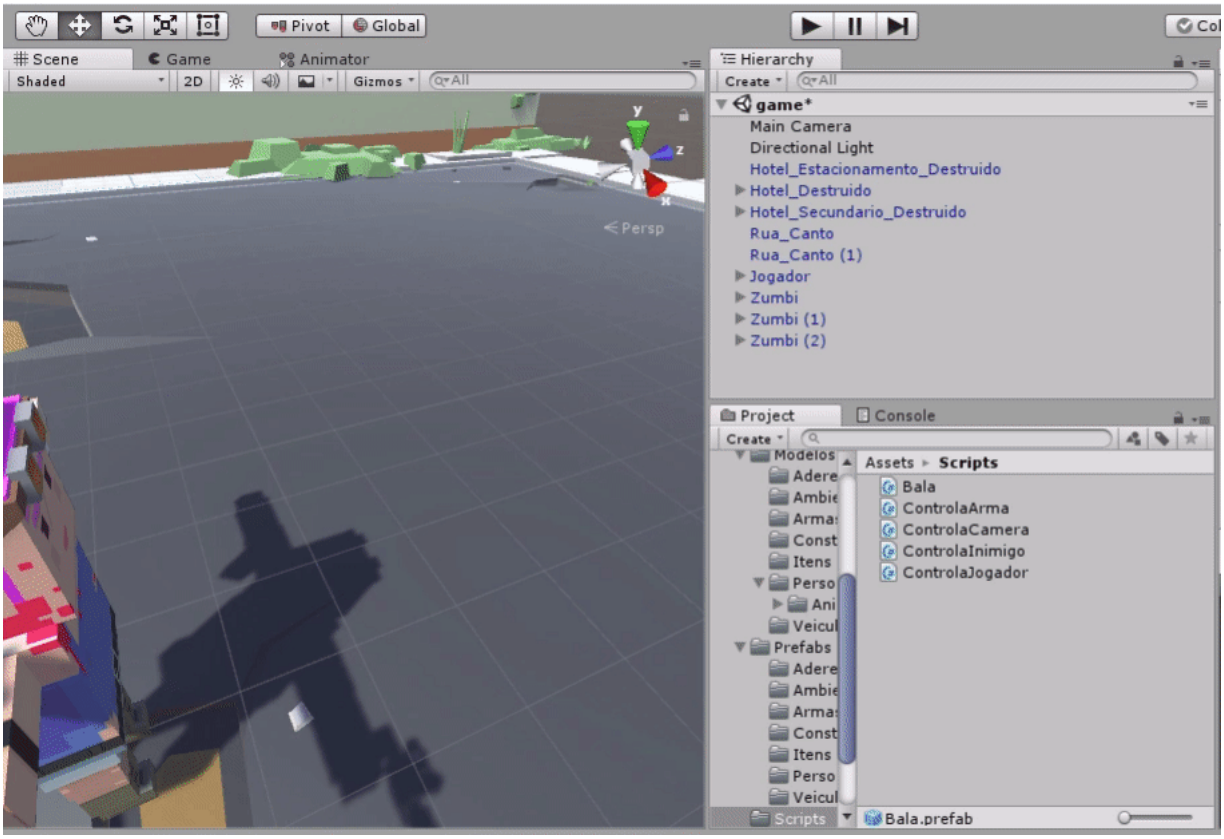
Criaremos um *script* para que a bala se movimente. Clicaremos com o botão direito do mouse em "Project > Assets > Script" e selecionaremos "Create > C# Script". Nomearemos como "Bala" e para acrescentá-lo à "Bala" em "Prefabs", considerando que não poderemos arrastá-lo:

- abriremos o "Inspector" de "Bala";
- clicaremos em "Add Component";
- selecionaremos "Scripts > Bala".

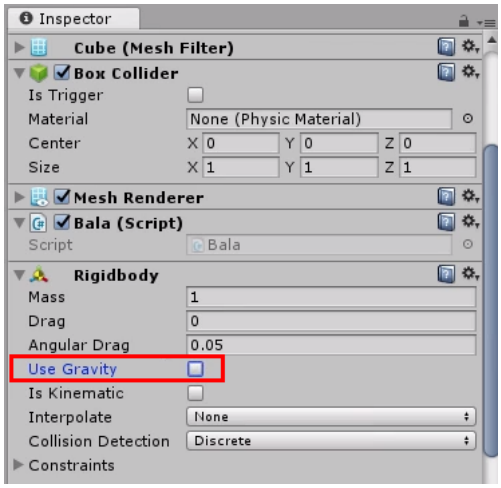
Dessa forma, adicionamos o *script* à "Bala". Abriremos para editá-lo e começaremos deletando o trecho de `Start`, que não utilizaremos. Na sequência, adicionaremos `Fixed` à `Update`, porque utilizaremos `Rigidbody`. Então salvaremos e minimizaremos o código que, por enquanto está da seguinte forma:

```
public class Bala : MonoBehaviour {

    // Update is called once per frame
    void FixedUpdate () {

    }
}
```

Voltaremos à Unity, clicaremos em "Add Component" e selecionaremos "Rigidbody". Em "Inspector", veremos que a caixa de "Use Gravity" está selecionada. Notem que, em função disso, após atirar, as balas caem.



Sendo assim, desabilitaremos "Use Gravity".



No código de `Bala.cs`, faremos com que ela vá para frente, da mesma forma que fizemos com "Jogador", anteriormente. Em `FixedUpdate`, entre as chaves (`{}`), adicionaremos:

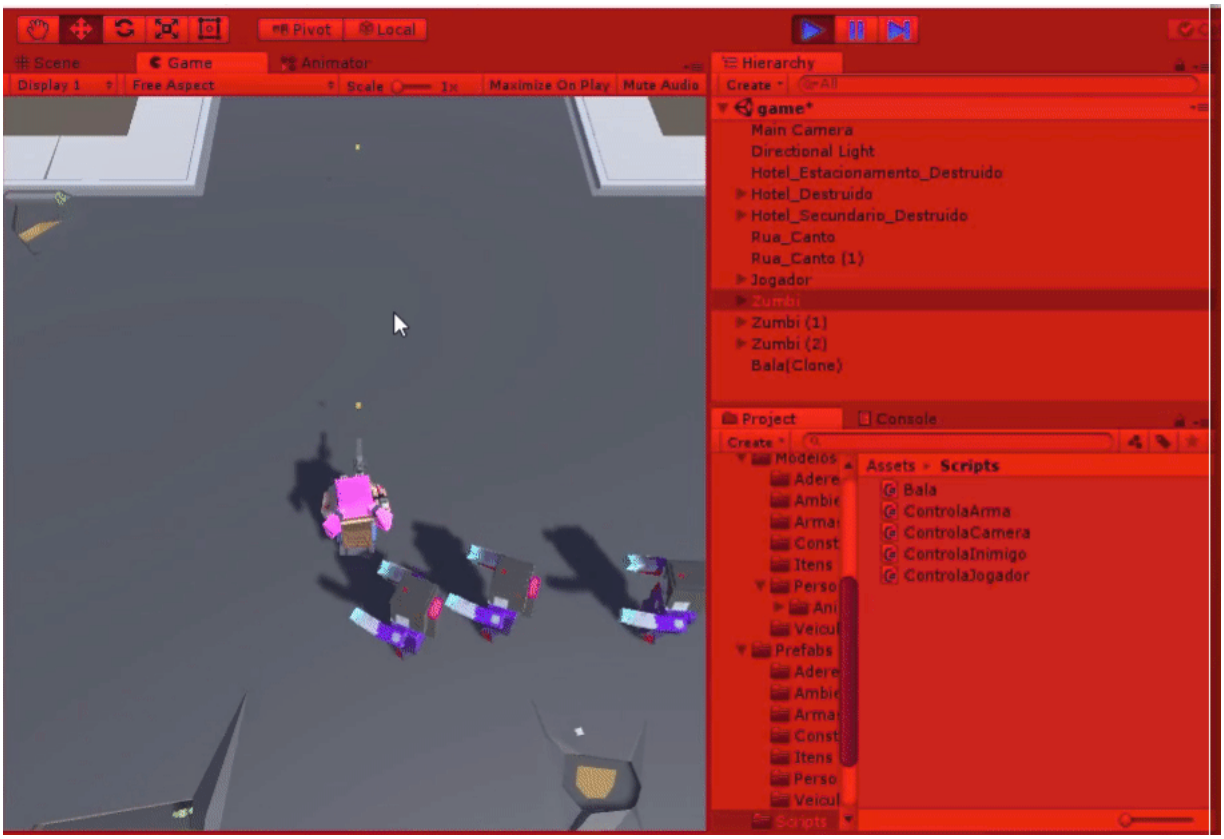
```
GetComponent<Rigidbody>().MovePosition
    (GetComponent<Rigidbody>().position +
    transform.forward * Velocidade);
```

Dessa forma, somamos à posição em que "Jogador" está, a posição para qual queremos que a bala vá, que é para frente, no eixo Z. Então, utilizamos `transform.forward`, para que a bala siga em frente no eixo Z dela. Se usássemos `Vector3.transform`, como fizemos em "Jogador", ela iria em frente, no eixo Z da Unity.

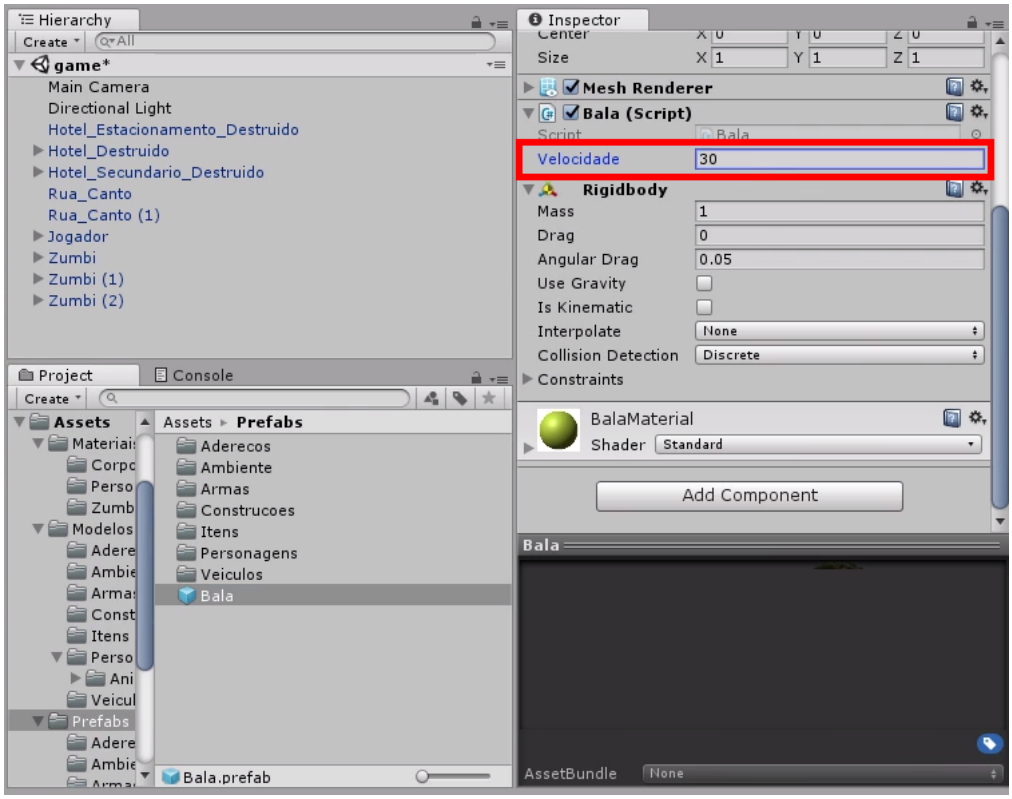
E, por fim, declaramos a variável `Velocidade`, acima de `Update` e multiplicamos (`*`) o trecho anterior por ela e por `Time.deltaTime`. Não nos preocuparemos com a normalização da velocidade porque `transform.forward` movimentará a bala somente um quadradinho no eixo Z, multiplicado pela `Velocidade` e pelo tempo (`Time.deltatTime`). Ou seja, não teremos o problema de velocidade discrepante.

Salvaremos e minimizaremos o código. Lembrem-se sempre de salvar, pois caso não o façam as alterações aplicadas no código não serão exibidas em "Game".

Na sequência, ativaremos o "Play" e notem que, agora, ao disparar as balas, elas vão embora.



Caso seja necessário ajustar a velocidade de disparo, podemos acessá-la em "Prefabs" e alterar a velocidade no *script*, localizado em "Inspector".



Assim, toda vez que clicarmos com o mouse, a Unity criará novas balas e as mandará para frente, considerando o eixo Z dela e a rotação do cano da arma. Pronto! As balas são criadas no cano e disparadas para frente.