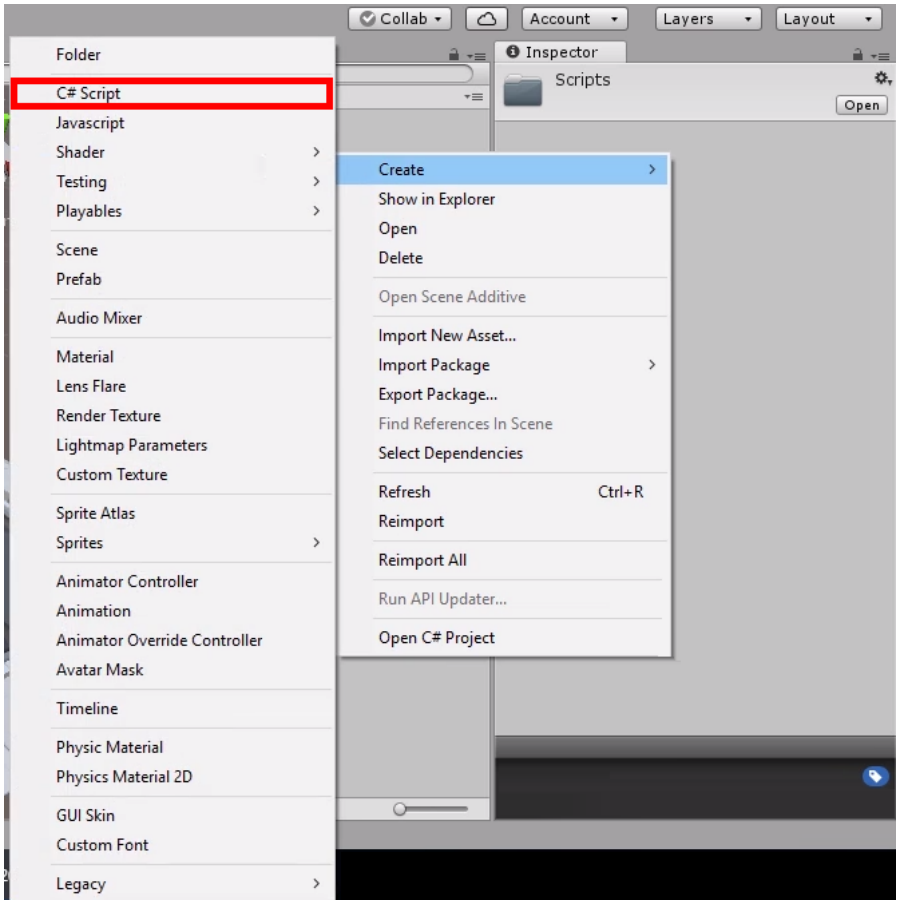
01

# Gerador de zumbis

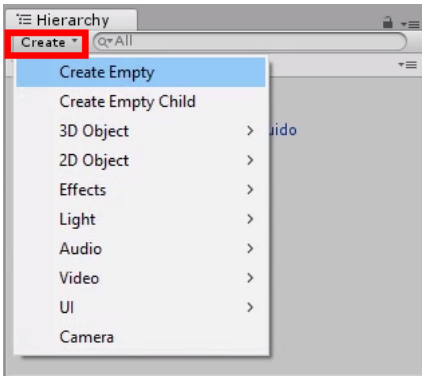
### Transcrição

O jogo está ficando bem bacana. Os zumbis perseguem a heroína e, ao atingi-la, o jogo é pausado e pode ser reiniciado (*load scene*) com o clique do mouse.

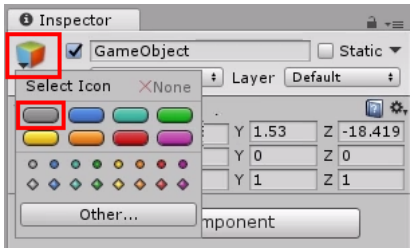
No entanto, só temos três zumbis. Se os matamos, o jogo acaba. Levando isso em consideração, faremos com que zumbis sejam gerados o tempo todo, para que o jogadores possam jogar até serem atingidos pelo inimigo. Para isso, criaremos um novo *script* em "Project > Assets > Scripts", clicando com o botão direito do mouse e selecionando "Create > C# Script".



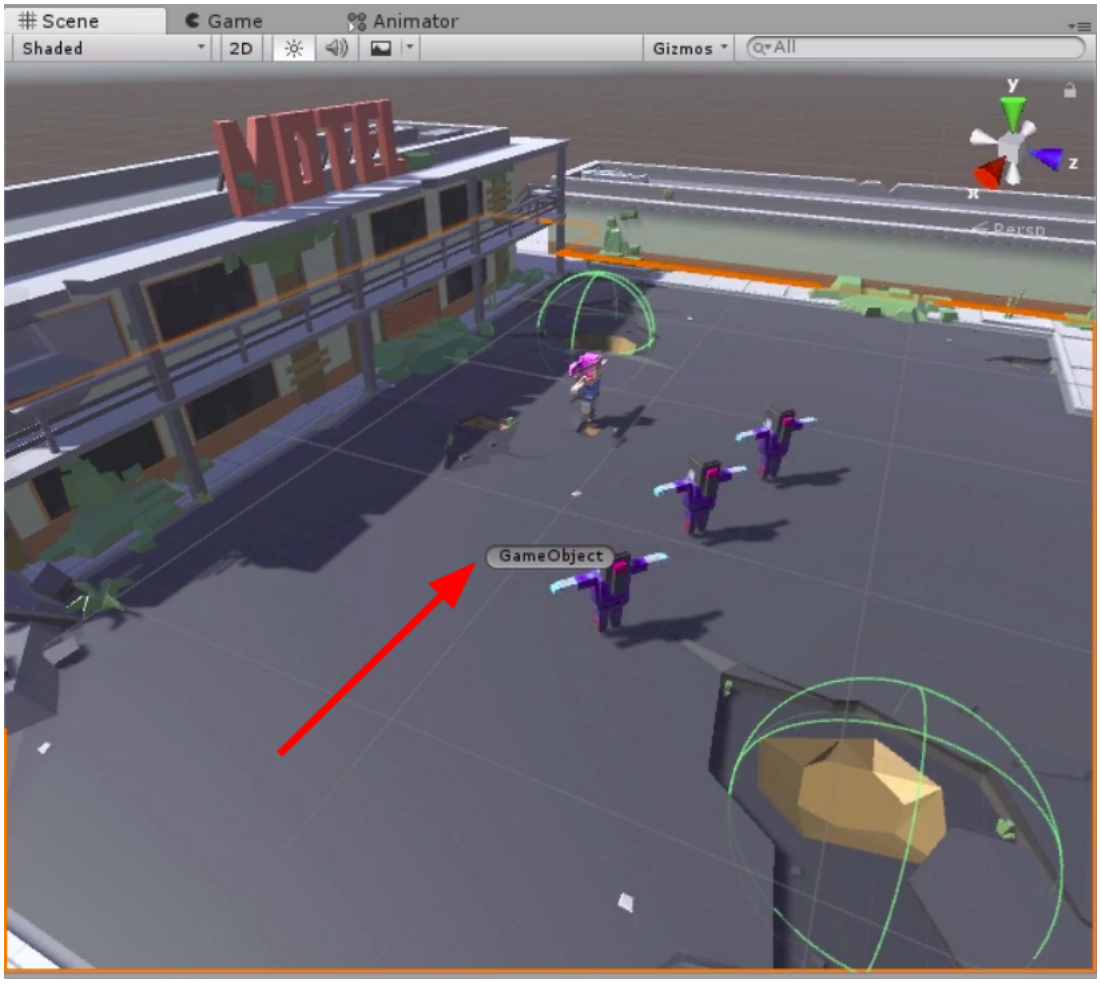

Nomearemos o novo *script* como "GeradorZumbis" e o colocaremos em um **objeto vazio**, da mesma forma que fizemos com o cano da arma, anteriormente (https://cursos.alura.com.br/course/criacao-de-jogos-com-unity/task/30399). Assim, marcaremos uma **posição** para criação de zumbis no espaço, com esse objeto referência. Não será necessário estar em determinado lugar do jogo para gerar inimigos. O objeto terá como única função, marcar uma posição. Em "Hierarchy", clicaremos em "Create > Create Empty".




Teclaremos "W" e o localizaremos, mas é difícil localizar um objeto vazio. Se perdermos a seleção, não conseguiremos mais clicar nele. Então, clicaremos em "Hierarchy > GameObject" e, em "Inspector", clicaremos no cubo colorido do canto superior esquerdo. Selecionaremos cinza.

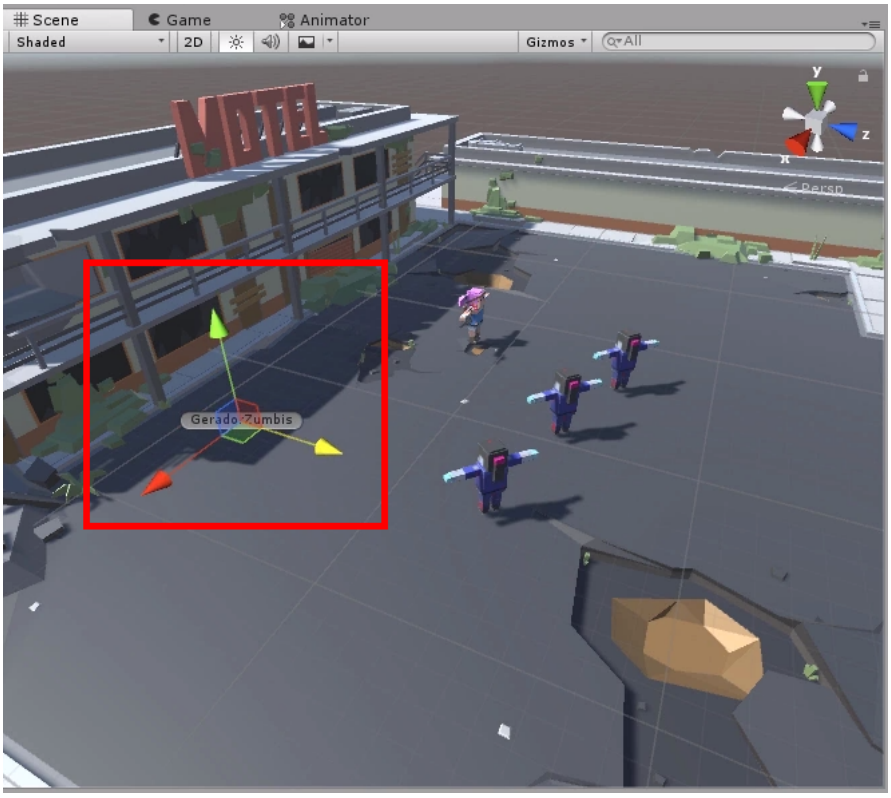



Notem que o nome do objeto aparecerá em um retângulo de cantos curvos. Se perdermos a seleção dele, basta clicarmos no ícone para selecioná-lo.

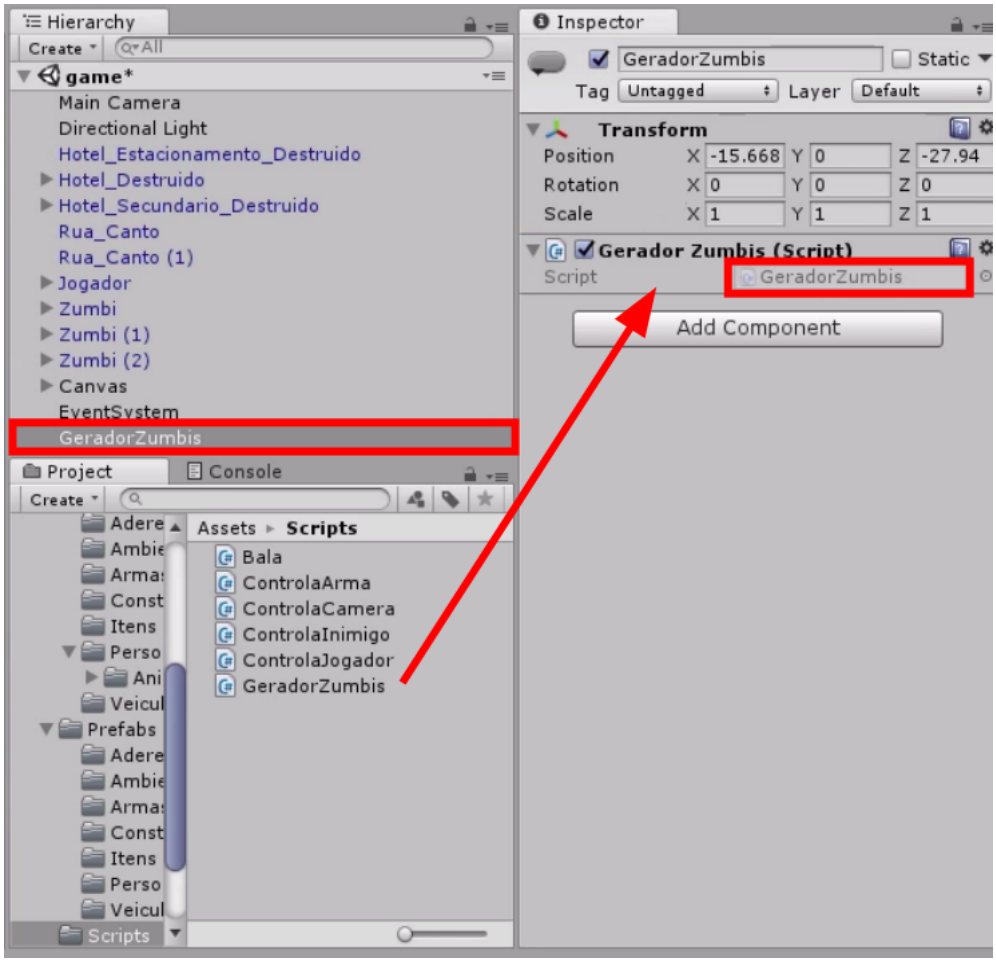



Assim fica mais fácil localizar e selecionar objetos vazios. Em "Inspector", substituiremos o nome "GameObject" por "GeradorZumbis". Iremos posicioná-lo próximo a "Hotel_Destruido" e atribuir `0` a "Position Y" de "Transform", em

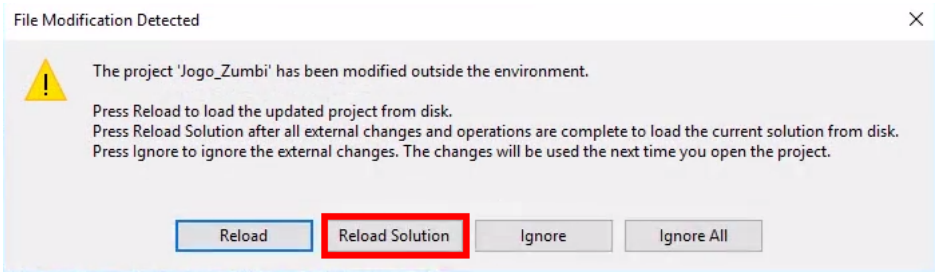
"Inspector".



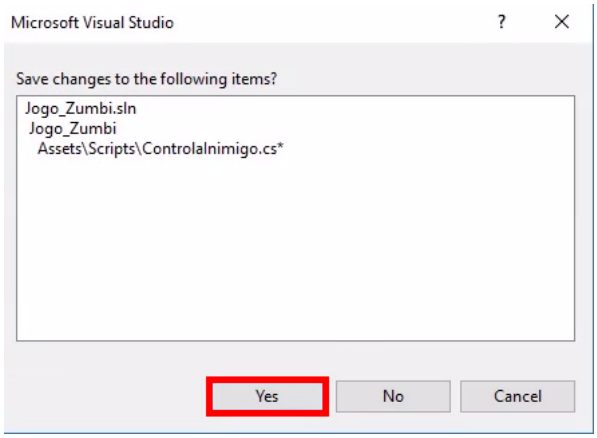

Arrastaremos o *script* `GeradorZumbis.cs` que criamos em "Project > Assets > **Scripts**" para o "Inspector" de "Hierarchy > Canvas > **GeradorZumbis**".



Abriremos o *script*, clicando duas vezes nele no "Inspector" de "GeradorZumbis". Abrirá uma mensagem de detecção de modificação no arquivo, na qual selecionaremos a opção "Reload Solution".



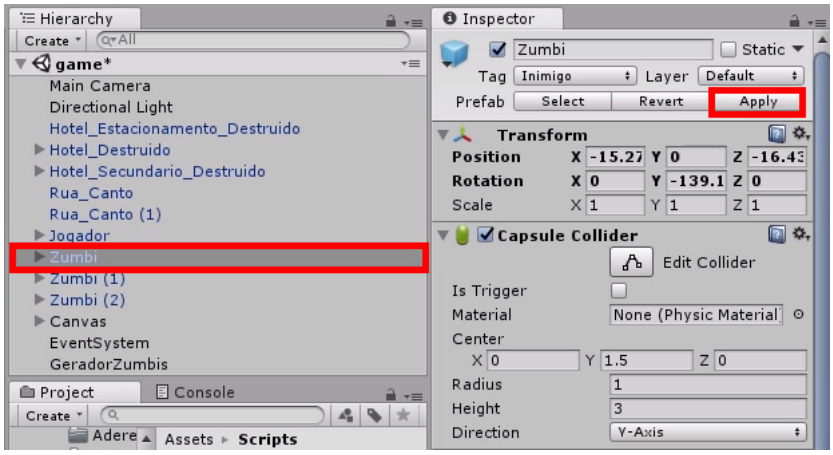

Na sequência, abrirá outra mensagem, perguntando se desejamos salvar os seguintes itens. Selecionaremos a opção "Yes".




Começaremos a desenvolver o código declarando uma variável para zumbi, acima de `Start`, por meio de:

```
public GameObject Zumbi;
```

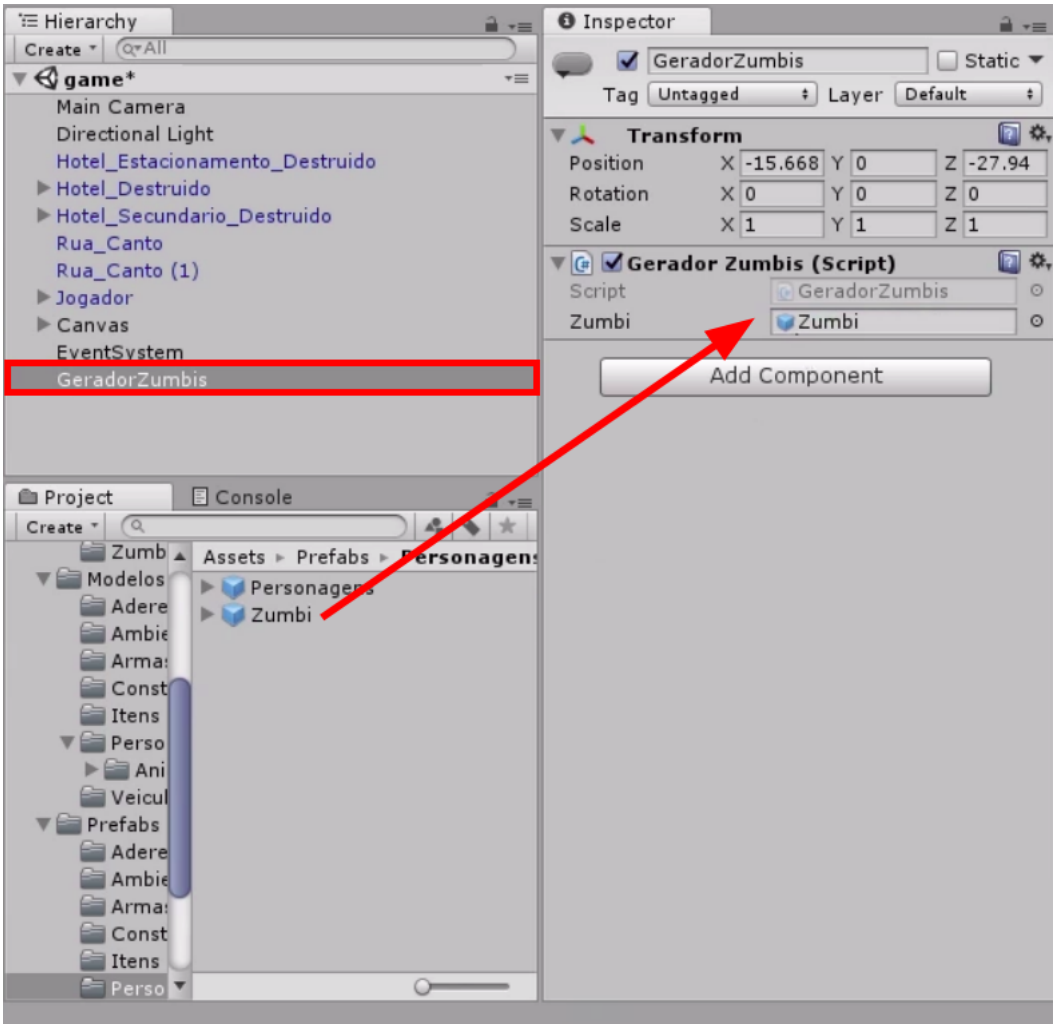
Salvaremos e minimizaremos o código. De volta à Unity, preencheremos a variável, que aparecerá abaixo do *script*, em "Inspector". Para preenchê-la, em vez de utilizar um dos três zumbis que criamos, selecionaremos "Hierarchy > Zumbi". Em "Inspector", clicaremos em "Apply".




Em "Hierarchy", selecionaremos os três zumbis:

- "Zumbi";
- "Zumbi (1)";
- "Zumbi (2)".

E deletaremos, pressionando a tecla "Delete". Na sequência, acessaremos "Project > Assets > Prefabs > Personagens" e arrastaremos "Zumbi" ao espaço da variável `Zumbi`, em "Inspector" de "GeradorZumbis".



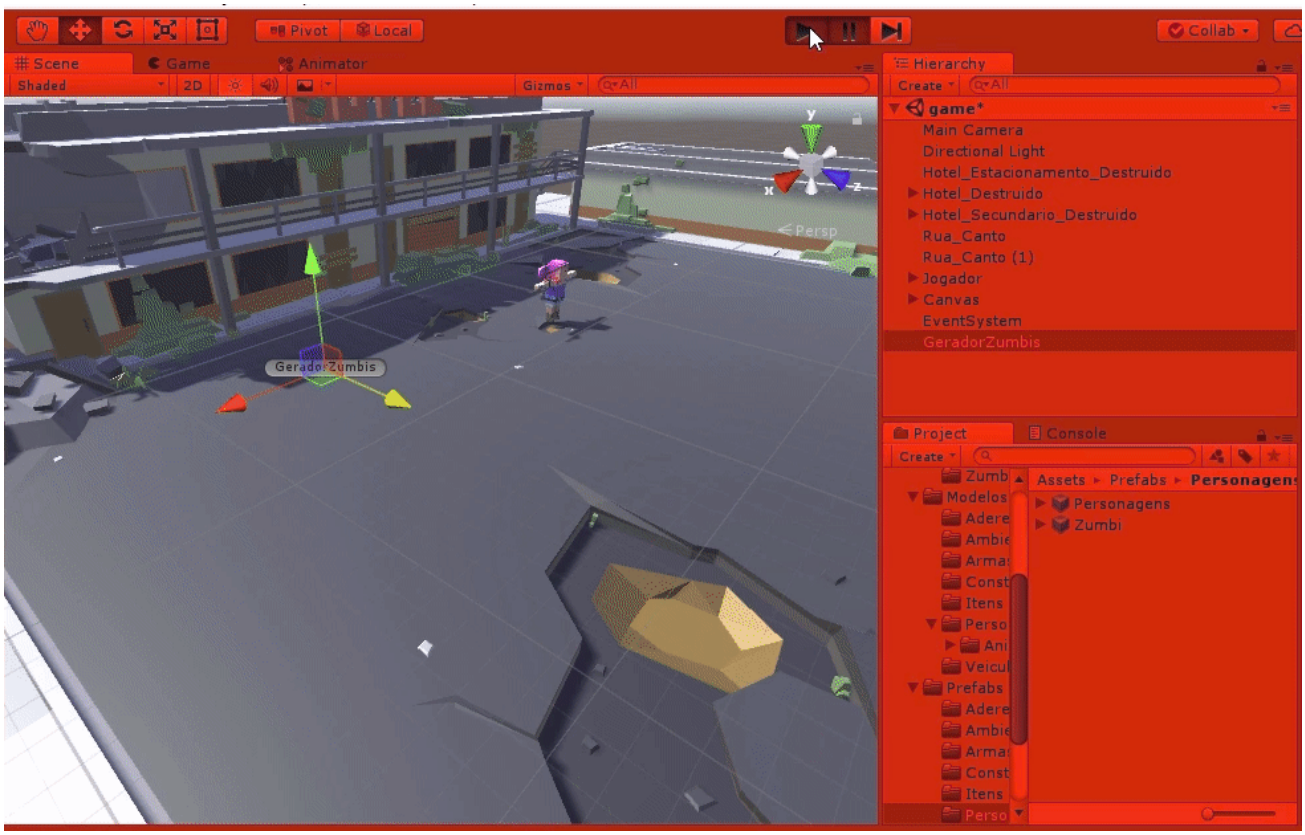

Se preenchêssemos com um dos zumbis que estavam no jogo, no momento em que o matássemos, a Unity não saberia qual zumbi teria que gerar, considerando que estaria gerando cópias daquele que foi extinguido. Como "Prefab" é fixo, se o utilizarmos como referência para gerar zumbis, ele estará lá, sem sofrer influência dos que são atingidos durante o jogo.

Com a variável `Zumbi` preenchida com o "Prefab", voltaremos a `GeradorZumbis.cs`. Em `Update`, utilizaremos `Instantiate()` para gerar ou instanciar zumbis, portanto colocaremos a variável `Zumbi` entre os parênteses (`()`), seguida pela posição (`transform.position`) que marcamos com o objeto referência e pela rotação (`transform.rotation`), separadas por vírgulas (`,`). Salvaremos e minimizaremos o código da seguinte forma:

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

public class GeradorZumbis : MonoBehaviour {

        public GameObject Zumbi;

        // Use this for initialization
        void Start () {

        }
```

```
        // Update is called once per frame
        void Update () {
            Instatiate(Zumbi, transform.position, transform.rotation);
        }

}
```

De volta à Unity, ativaremos "Play" para ver a alteração que fizemos.




Notem que milhares de zumbis foram gerados, porque `Update` roda o tempo todo, então a cada *frame*, um zumbi é gerado. Em vez de gerar **um por *frame***, faremos com que seja gerado **um por segundo**. Para isso, primeiro precisamos de um **relógio** para contar o tempo e saber se passou um segundo.

Abaixo da declaração de `Zumbi`, declararemos uma variável do tipo `float`, que não será pública. A nomearemos como `contadorTempo` e atribuiremos ( `=` ) o valor `0`. Esse será o nosso relógio.

Em `Update`, acima de `Instatiate()`, colocaremos `contadorTempo` e atribuiremos ( `=` ) a ele `contadorTempo` (ele mesma) somada ( `+` ) a variável que transforma o tempo de *frames* em segundos. Assim, `contadorTempo` é iniciado em `0`.

Com o rodar do jogo (método `Update`), será somado a ele o tempo que a Unity demorou para gerar aquele *frame*. Por exemplo, se `contadorTempo` é igual a `0` e `Time.deltaTime` é igual a `1`, a soma será de `0 + 1` e o resultado ( `1` ) será atribuído a `contadorTempo`. Valendo `1`, a próxima soma será `1 + 1`, supondo que demorou um segundo. Então, na próxima soma, será somado `Time.deltaTime` a `contadorTempo` valendo `2`. Em programação podemos simplificar, deletando `contadorTempo` da conta e adicionando o sinal de soma ( `+` ) antes do sinal de igual ( `=` ). Dessa forma, continuamos a somar o valor que `contadorTempo` já tem a `Time.deltaTime` e atribuímos o resultado de volta a `contadorTempo`. Assim, contaremos o tempo em segundos.

Adicionaremos outra variável pública `TempoGerarZumbi`, abaixo de `contadorTempo`, também do tipo `float`, que administre o tempo para gerar zumbis, que será `1`. Ou seja, a cada um segundo serão gerados zumbis.

Em `Update`, para saber se passou um segundo, adicionaremos `if`, abaixo de `contadorTempo`. Então, se (`if`) o tempo que estamos contando (`contadorTempo`) for maior ou igual (`>=`) a um segundo (`TempoGerarZumbi`), novos zumbis serão gerados.

Dentro de `if`, adicionaremos chaves (`{}`) e, entre elas, colocaremos o trecho de `Instatiate()`. Abaixo dele, iremos zerar o tempo para controlar a geração de zumbis. Faremos isso atribuindo (`=`) o valor `0` a `contadorTempo`. O código ficará da seguinte forma:

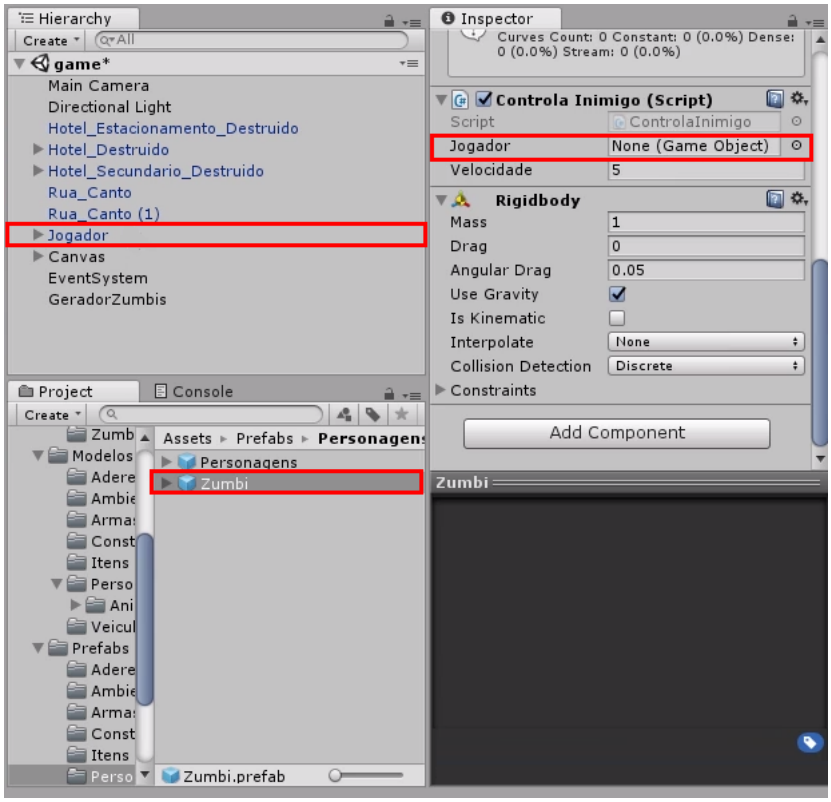
```
public class GeradorZumbis : MonoBehaviour {

        public GameObject Zumbi;
        float contadorTempo = 0;
        public float TempoGerarZumbi = 1;

        // Use this for initialization
        void Start () {

        }

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

            contadorTempo += Time.deltaTime;

            if(contadorTempo >= TempoGerarZumbi)
            {
                Instatiate(Zumbi, transform.position, transform.rotation);
                contadorTempo = 0;
            }

        }

}
```

Assim, contaremos o tempo. Se ele for maior ou igual a um, zumbis serão gerados e o relógio que cronometra esse tempo será zerado em seguida para reiniciar a contagem. Minimizaremos e salvaremos o *script*. De volta à Unity, ativaremos "Play" para ver como está funcionando.

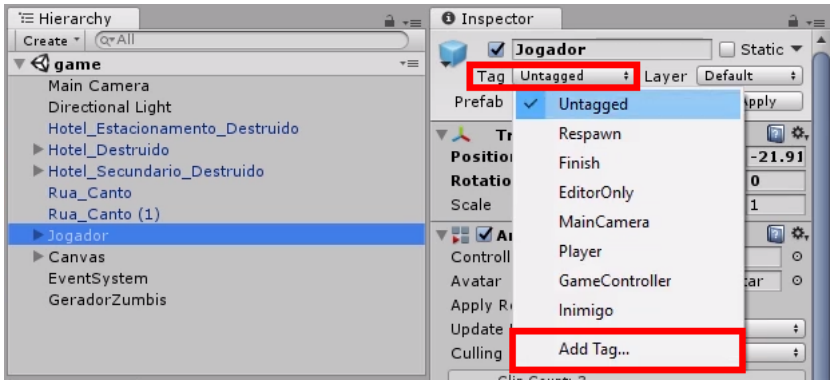

No entanto, os zumbis são criados e, como não estão perseguindo a heroína, ficam todos parados na posição de "GameObject". Se observarem a janela de "Console", que posicionamos à direita de "Project", veremos que há uma lista de erros. Um deles indica que, no *script* `ControlaInimigo.cs` , a variável `Jogador` não possui valor.

Se abrirmos "Inspector" de um "Zumbi(Clone)", observaremos que na parte "Controla Inimigo (Script)", o campo de `Jogador` está vazio (*None (Game Object)*).

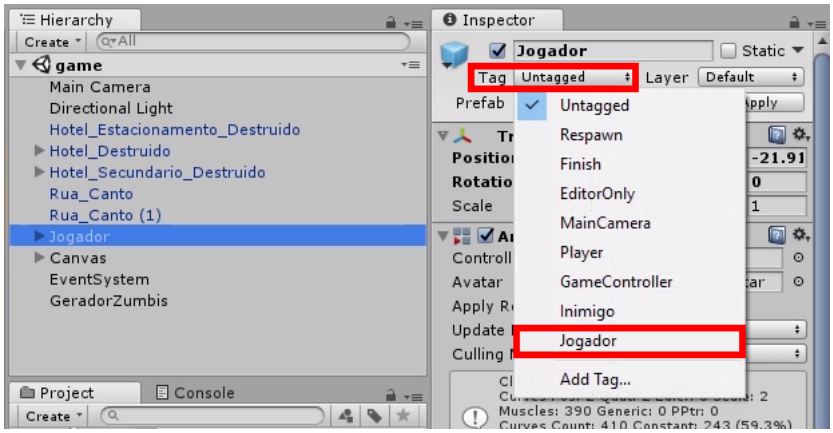

Isso aconteceu porque "Prefab" de "Zumbi", que está em "Project", não recebe nada que está em "Hierarchy", no caso, "Jogador".




Da mesma forma que fizemos a bala atingir os inimigos, para que os zumbis encontrem "Jogador", utilizaremos a etiqueta (*tag*). Com "Jogador" selecionado em "Hierarchy", em "Inspector, à direita de "Tag, clicaremos em "Untagged > Add Tag...".




Na sequência, clicaremos no botão de soma ( + ) para adicionar a *tag* "Jogador". De volta a "Inspector", acessaremos "Untagged" novamente para selecionar "Jogador".




Agora que "Jogador" está etiquetado, abriremos `ControlaInimigo` e, em `Start`, atribuiremos ( = ) `GameObject.FindWithTag` a `Jogador`. Assim, estabelecemos a busca do objeto com a etiqueta `"Jogador"`. Como

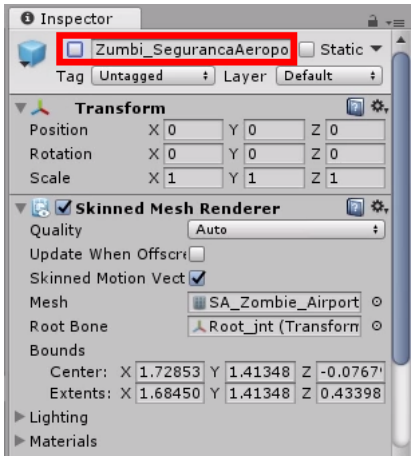
inserimos o trecho em `Start`, buscaremos "Jogador" somente uma vez, no início do jogo. Não será necessário recorrer a `Update` o tempo todo para buscá-lo quando o zumbi for criado. Lembrem-se que ao colocar o **nome** da etiqueta, entre parênteses ( `()` ) e aspas ( `"` ), devemos digitar **exatamente igual** ao nome que definimos na Unity. O método `Start` ficará da seguinte forma:

```
// Use this for initialization
void Start () {
    Jogador = GameObject.FindWithTag("Jogador");
}
```

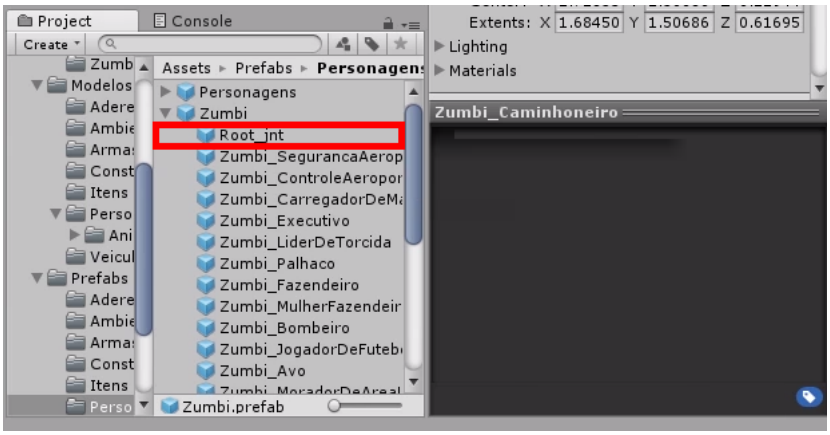
Dessa forma, ao gerar zumbis, a Unity buscará o objeto com a *tag* "Jogador" e lançará para a variável `Jogador`, ou seja, os inimigos perseguirão a heroína. Salvaremos e minimizaremos `ControlaInimigo.cs`. Na Unity, ativaremos "Play" para testar.


Funcionou. A cada um segundo, um zumbi é gerado, e todos perseguem a heroína. Mas, o jogo fica um pouco sem graça com todos os **zumbis iguais**. Desenvolveremos uma forma de diferenciá-los. Em "Project > Assets > Prefabs > Personagens", se clicarmos na seta à esquerda de "Zumbi", veremos uma lista com 27 tipos diferentes deles. O objetivo será utilizar um desses 27 tipos ao criá-los.

Começaremos selecionando o primeiro da lista ("Zumbi_SegurancaAeroporto") e desativando a caixa de seleção à esquerda do título dele, em "Inspector". A lógica é procuraremos dentro de "Project > Assets > Prefabs > Personagens > Zumbi" um objeto entre 27, ao gerar zumbis.




Considerando que a enumeração dos zumbis é iniciada em `0` ("Root_int", esqueleto deles) e vai até `27`, abriremos `ControlaInimigo.cs` e, em `Start`, declararemos uma variável abaixo de `Jogador`.




A variável será do tipo `int` (inteiro, ou seja, não abrange números decimais). A nomearemos como `geraTipoZumbi` e atribuiremos ( `=` ) como valor `Random.Range(1, 28)`, assim selecionaremos um número ( `Range` ) entre `1` e `28` de forma aleatória (*random*). Estabelecemos `28` como limite, porque ao selecionar aleatoriamente, o último número é descartado, ou seja, **`28` será excluído** e o número será sorteado de `1` a `27`. Dessa forma, ao iniciarmos o jogo, um número, entre `1` e `27`, será sorteado aleatoriamente. Suponhamos que seja `10`. Então, o décimo objeto da lista de zumbis será ativado.

Para acessar os diferentes tipos em "Zumbi", abaixo de `geraTipoZumbi`, utilizaremos `transform`. Na sequência, adicionaremos ponto ( `.` ) e `GetChild` para pegar um objeto que está dentro de algo e, entre parênteses ( `()` ), especificaremos o número dele na lista, que será obtido de `geraTipoZumbi`. Em seguida, sairemos de `transform` digitando `.gameobject` e adicionaremos `.SetActive(true)` para ativar. Com a alteração, `Start` ficará da seguinte forma:

```
// Use this for initialization
void Start () {
    Jogador = GameObject.FindWithTag("Jogador");
    int geraTipoZumbi = Random.Rage(1, 28);
}
```

Assim, geraremos um número de `1` a `27`, acessaremos "Zumbi" para encontrar um objeto com número correspondente ao que foi gerado e o ativaremos. Salvaremos e minimizaremos `ControlaInimigo.cs`. De volta à Unity, ativaremos "Play" e vejam que bacana:


Estamos gerando zumbis diferentes a cada segundo. Considerando que são 27 tipos, em algum momento haverá repetição, mas já ficou mais divertido com a variedade de zumbis.