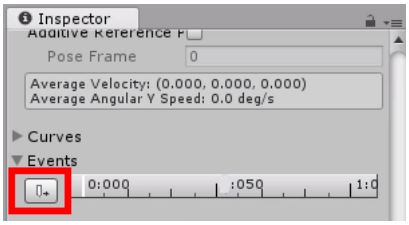
10

Reiniciar o jogo

Transcrição

Os zumbis já estão atacando a heroína. Agora, faremos "Jogador" perder (*Game Over*) quando os zumbis encostarem nela.

Para reconhecer que os zumbis encostaram na heroína, acessaremos "Project > Assets > Modelos3D > Personagens > Animacoes > Animacoes_Ataque". Em "Inspector", ativaremos o "Play" da demonstração da animação e posicionaremos no ponto (aproximadamente em 28 segundos) em que o zumbi está batendo. Clicaremos em "Events" e no botão com uma barra e sinal de soma (+) para criar um evento nesse ponto da animação.



Se movermos o ponto de referência na demonstração da animação, veremos que a barra azul de "Events" se move de acordo com ela.

Em "Function", nomearemos esse novo evento como "AtacaJogador". Selecionaremos e copiaremos o nome e, em seguida, clicaremos no botão "Apply".

Agora, precisamos configurar o código para que rode o evento "AtacaJogador" no momento em que os zumbis atacam "Jogador". Assim, *linkaremos* o evento no jogo ao código.

Em `ControlaInimigo.cs` , que no momento está da seguinte forma:

```
public class ControlaInimigo : MonoBehaviour {
```

```
    public GameObject Jogador;
    public float Velocidade = 5;

    // Use this for initialization
    void Start () {

    }

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

    }

    void FixedUpdate()
    {
        float distancia = Vector3.Distance(transform.position, Jogador.transform.position);

        Vector3 direcao = Jogador.transform.position - transform.position;

        Quaternion novaRotacao = Quaternion.LookRotation(direcao);
        GetComponent<Rigidbody>().MoveRotation(novaRotacao);

        if (distancia > 2.5)
        {
            GetComponent<Rigidbody>().MovePosition
                (GetComponent<Rigidbody>().position +
                direcao.normalized * Velocidade * Time.deltaTime);

            GetComponent<Animator>().SetBool("Atacando", false);
        }
        else
        {
            GetComponent<Animator>().SetBool("Atacando", true);
        }

    }
}
```

Acrescentaremos, abaixo de `FixedUpdate`, ainda dentro da classe, outro método. Já utilizamos:

- `FixedUpdate`;
- `Update`;
- `Start`.

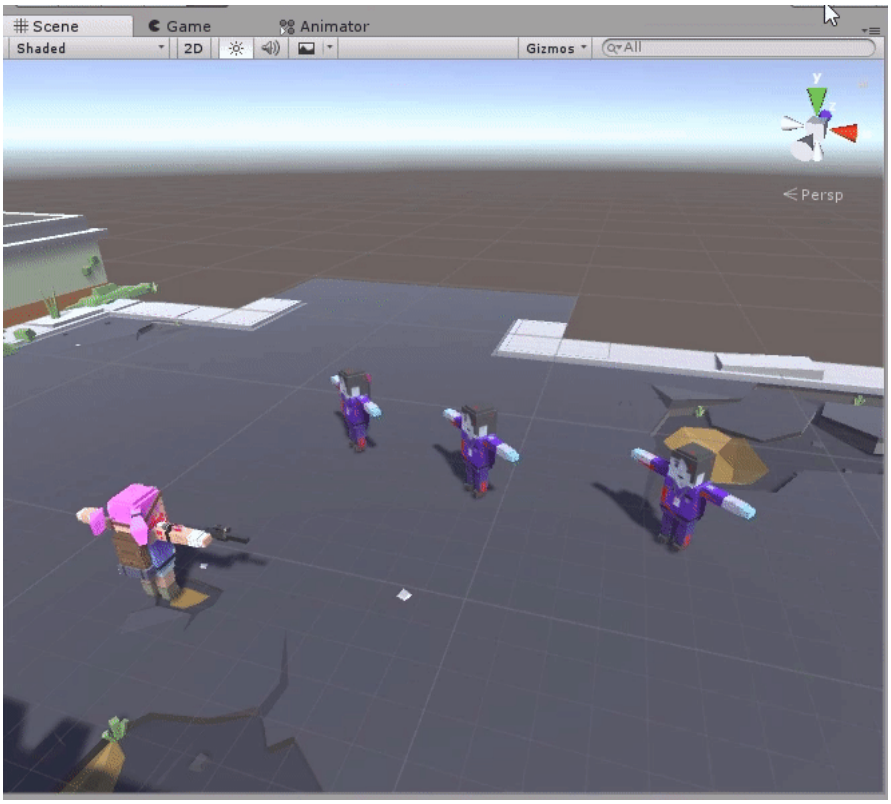
Esses três são fornecidos pela Unity. Iremos gerar um que a Unity não forneceu para atacar "Jogador". Começaremos digitando `void`. Na sequência utilizaremos o atalho "Ctrl + V" para colar o nome `AtacaJogador` que copiamos em "Events", pois o método deve ter o mesmo nome. Por isso copiamos, anteriormente; para garantir que o texto estará idêntico.

Abriremos parênteses (`()`) e chaves (`{}`), da mesma forma que fizemos em `FixedUpdate`, `OnTriggerEnter`, entre outros. Quando o evento rodar, no momento da animação em que o inimigo bater no "Jogador", esse método que estamos desenvolvendo será rodado automaticamente pela Unity.

Para pausar o jogo, utilizaremos `Time.timeScale`, que representa a escala de tempo da Unity. O padrão é `1`. Se colocarmos `2`, por exemplo, o jogo será executado duas vezes mais rápido. Se colocarmos `0` ele irá parar, ou seja, vai pausar o jogo. Atribuindo o valor `0` a `Time.timeScale` em `AtacaJogador`, estabelecemos que o jogo deve ser pausado para vermos algo acontecer quando um zumbi atacar "Jogador" e para quem está jogando entender que ele foi atingido pelo inimigo. Salvaremos e minimizaremos `ControlaInimigo.cs` com o trecho acrescentado, que ficou da seguinte forma:

```
void AtacaJogador ()
{
    Time.timeSacle = 0;
}
```

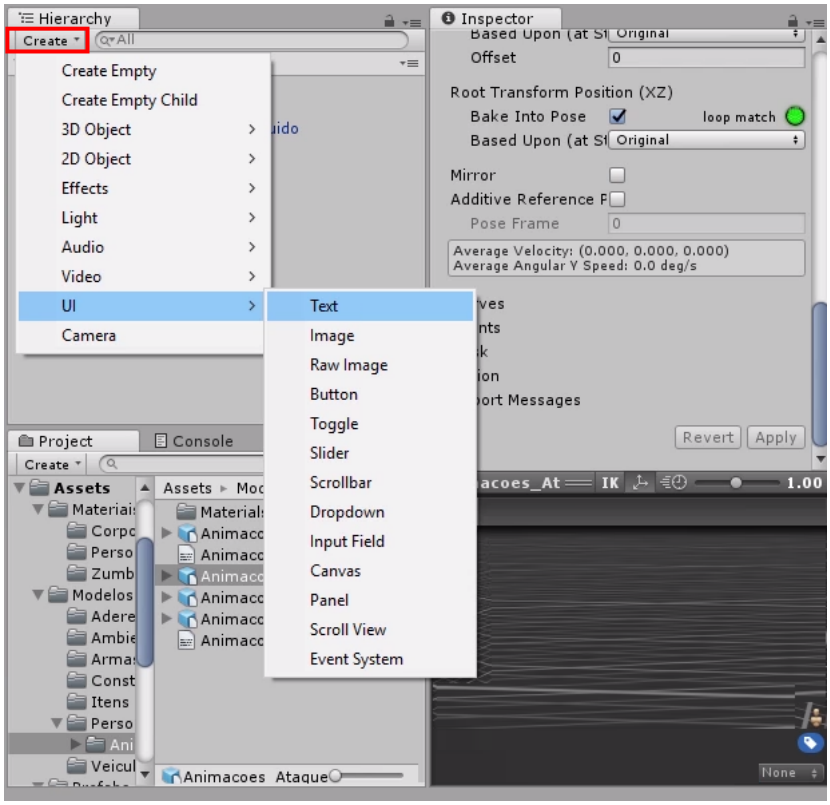
De volta à Unity, ativaremos "Play" para ver o funcionamento da alteração que fizemos.



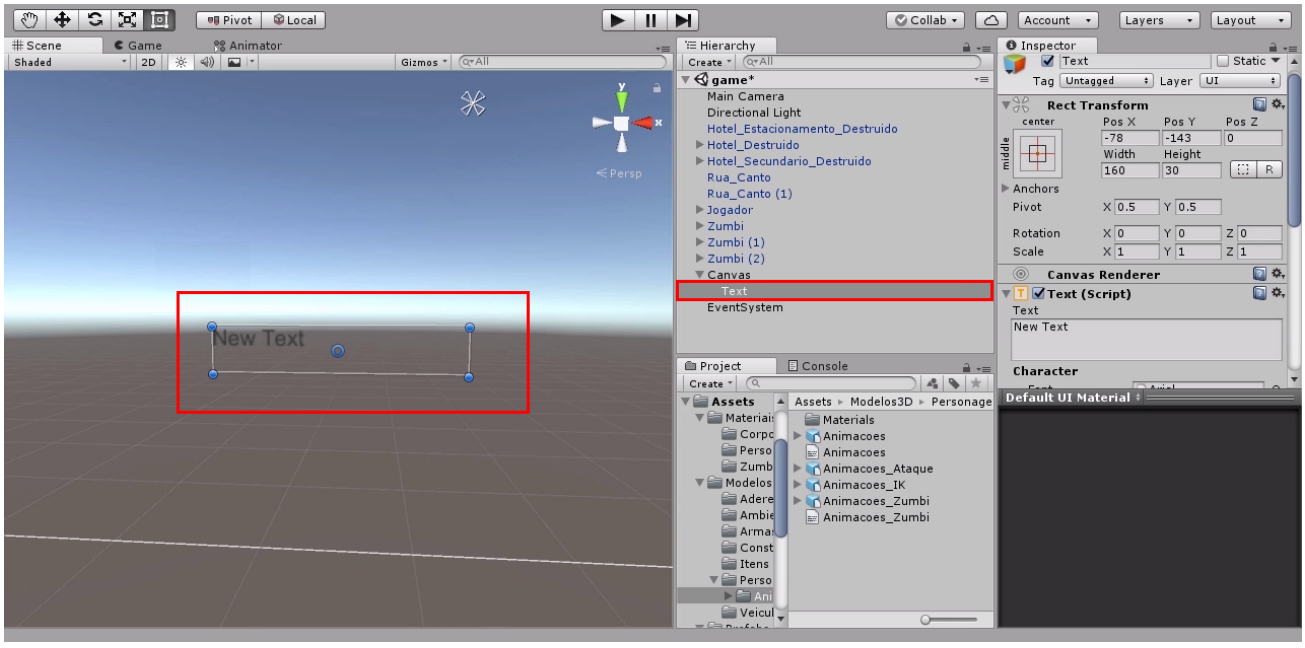
Notem que ao atingir a heroína, o jogo foi pausado. Assim, temos um jogo em que "Jogador" deve fugir dos zumbis e se não matá-los e for atingido por eles, o jogo é pausado e temos *Game Over* ("Fim de Jogo" em português).

Precisamos sinalizar que essa pausa é de *Game Over*. Para nós é fácil deduzir porque estamos criando o jogo, mas para quem vai jogar, sem participar da criação, é necessário um aviso na tela do que aconteceu. Para isso, utilizaremos as **interfaces** como meio de comunicação.

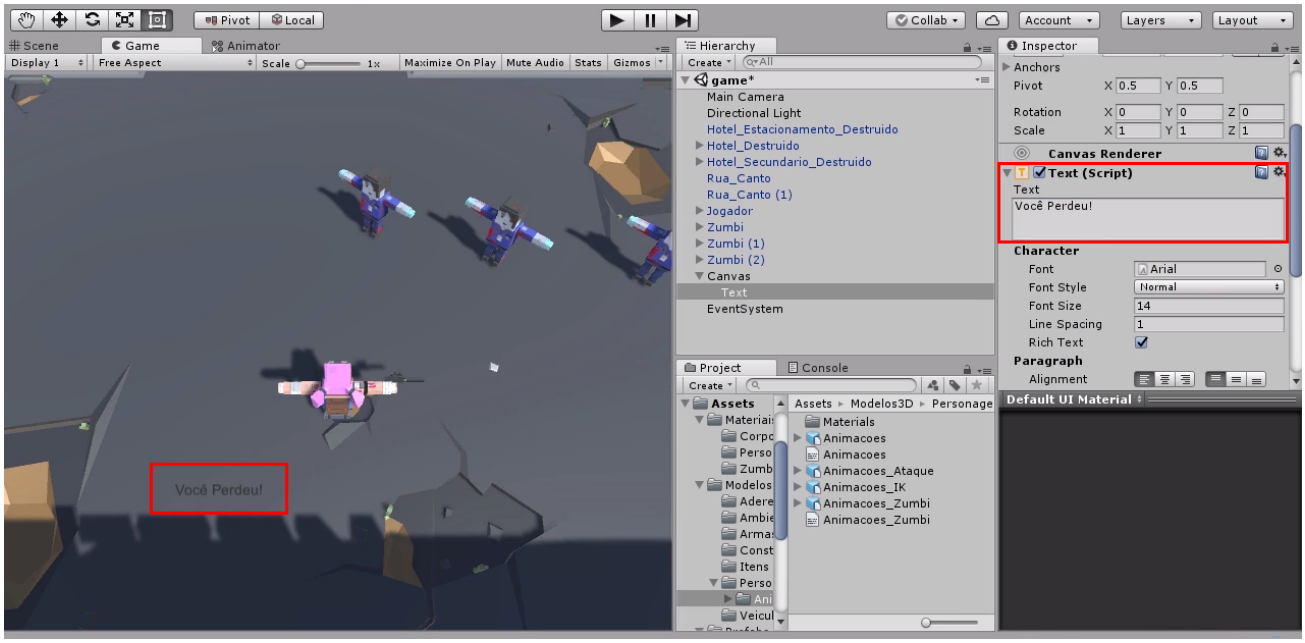
Em "Hierarchy", clicaremos em "Create > UI (*User Interface*) > Text" para criar um texto ao usuário.



Reparem que o texto ("Text") virá dentro de um "Canvas". Não se preocupem, pois a Unity fez isso automaticamente. O "Canvas" representa a tela do jogo, dentro da qual o texto deve ser inserido. Teclaremos "F" para localizar o texto e encontraremos uma caixa para editar.

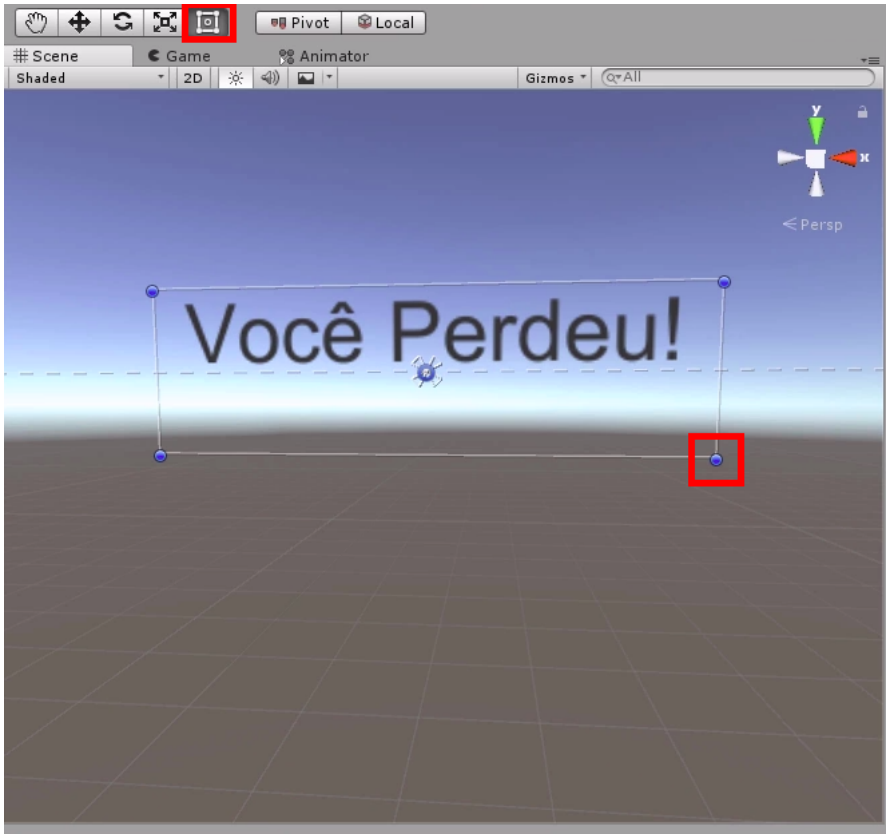


A edição pode ser feita em "Inspector". Em "Text (Script)" podemos modificar o texto. Digitaremos `Você Perdeu`. Se clicarmos na aba "Game", veremos o texto próximo ao canto inferior esquerdo.



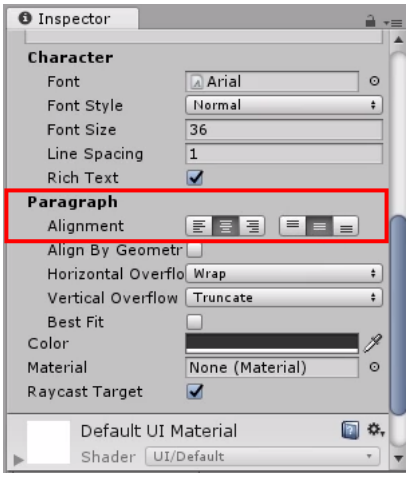
O texto está pequeno. Precisamos aumentar o tamanho da fonte. Faremos isso em "Font Size", clicando em cima e arrastando para a direita até chegar em `36`. Dessa forma a fonte ficará maior que a caixa de texto. Portanto, iremos aumentá-la também, selecionando a ferramenta de retângulo no menu do canto superior esquerdo, que lida com as interfaces utilizadas. Em "Scene", podemos ativá-la clicando ou pressionando a tecla "T".

Clicaremos em uma das bolinhas azuis dos quatro cantos do retângulo e arrastaremos para aumentar até conseguir visualizar o texto.

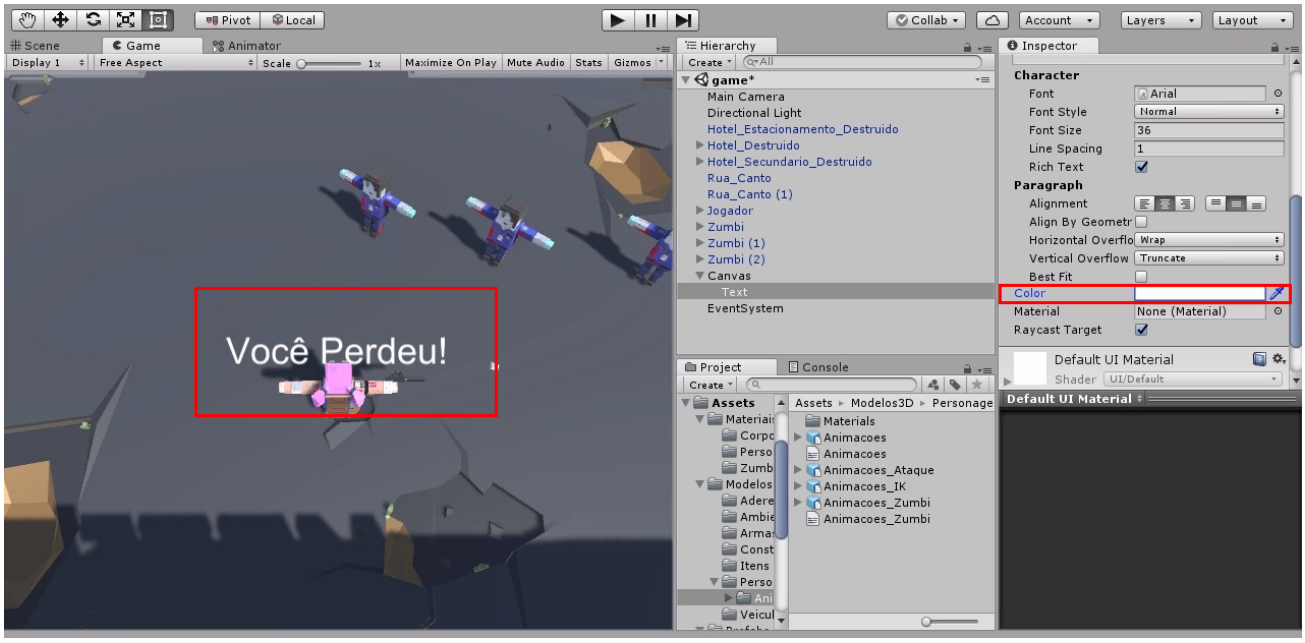


Se abrirmos "Game" novamente, veremos que já está em um tamanho melhor.

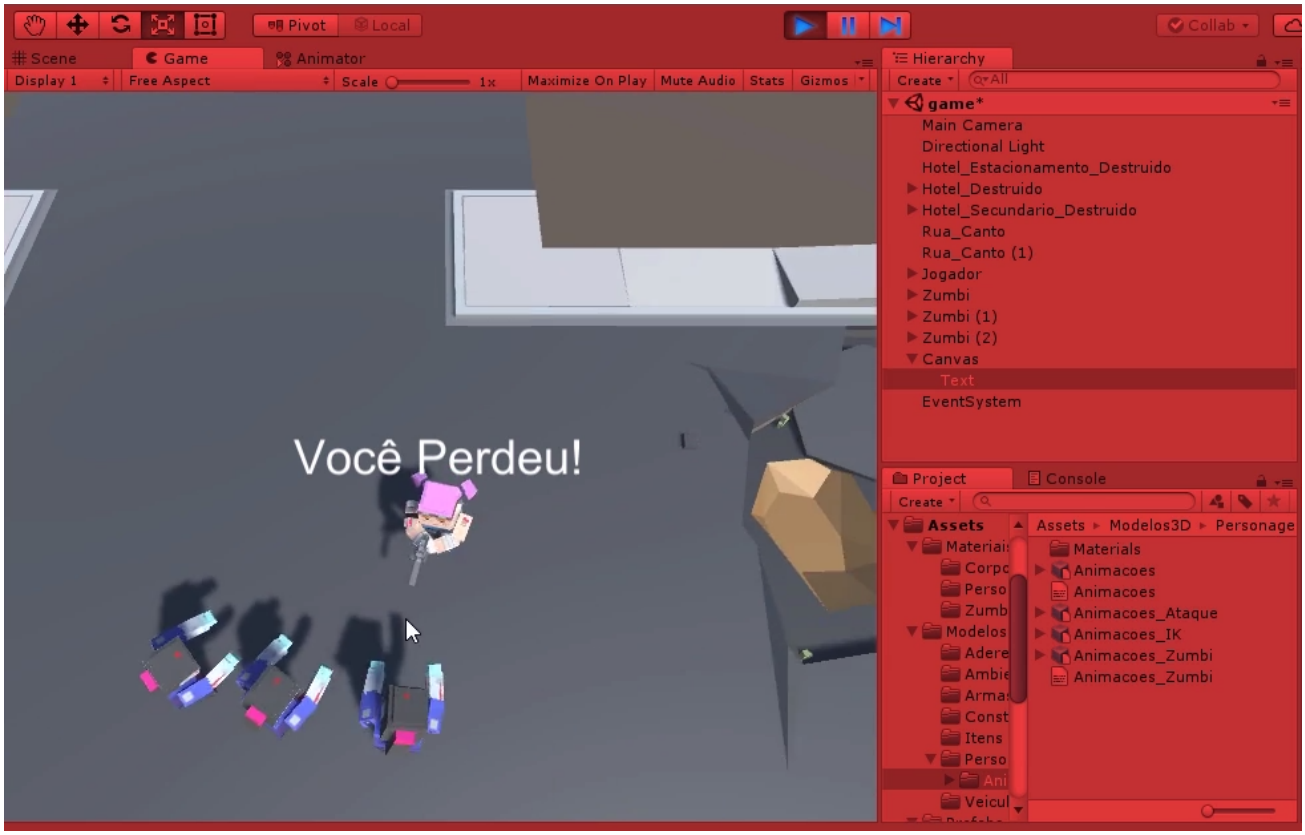
Em "Inspector", aplicaremos `0` a todos os eixos (X, Y e Z) de "React Transform" para centralizar a caixa de texto na tela. Alinharemos o texto horizontalmente e verticalmente na caixa, selecionando as respectivas opções em "Alignment".



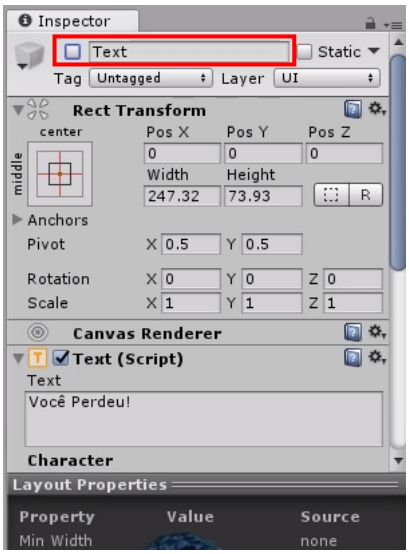
Com o texto devidamente centralizado, mudaremos a cor dele para visualizá-lo melhor. Em "Inspector", clicaremos em "Color" e selecionaremos a cor branca.



Agora, o texto ganhou mais destaque. Mas, se ativarmos "Play", veremos que o texto está fixo e é exibido antes mesmo de "Jogador" perder.



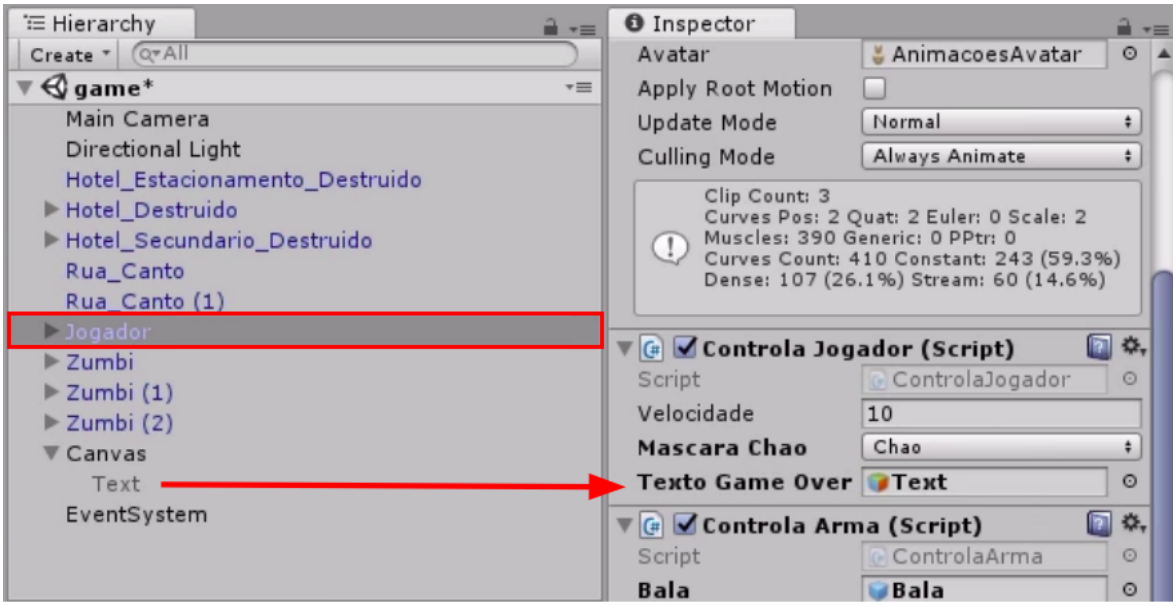
Para que apareça somente quando "Jogador" perder, desmarcaremos a caixa de seleção de "Text" no topo de "Inspector".



Na sequência, em `ControlaJogador.cs`, no início do código, abaixo de `LayerMask`, criaremos uma variável (`TextoGameOver`) à qual atribuiremos o texto:

```
public GameObject TextoGameOver;
```

Salvaremos e minimizaremos `ControlaJogador.cs`. De volta à Unity, arrastaremos "Texto" de "Hierarchy" para "Texto Game Over" no "Inspector" de "Jogador".

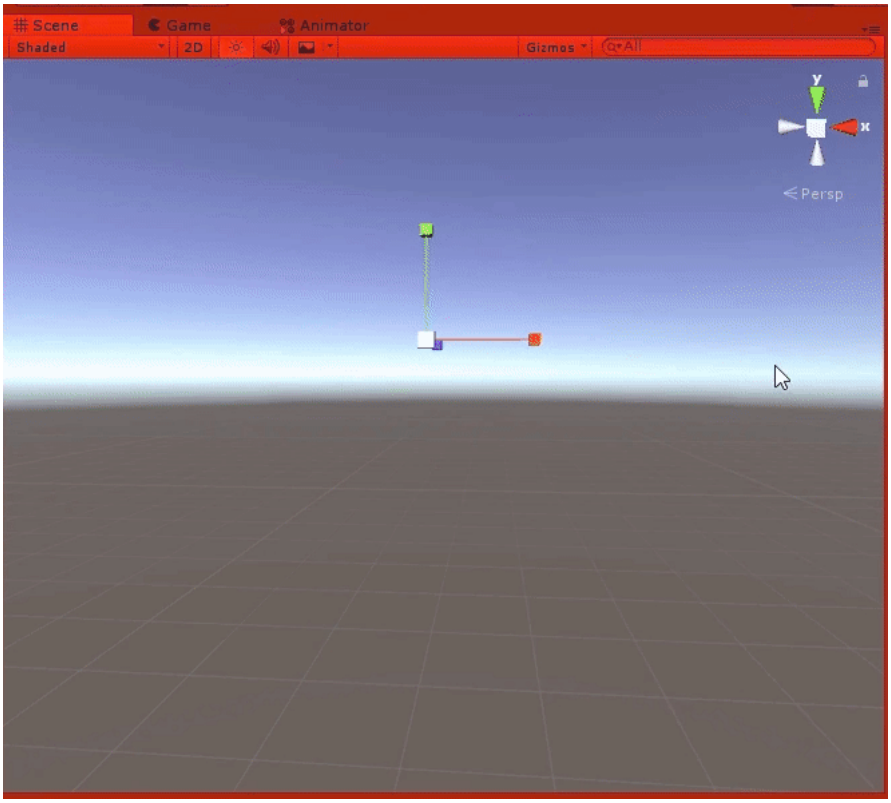


Inserimos em "Jogador" porque é um só. Se fôssemos inserir em zumbis, teríamos que adicionar a cada um deles e teríamos mais trabalho, consequentemente. Levando isso em consideração, é mais prático adicionar a "Jogador".

Adicionamos "Text" desativado e o ativaremos quando necessário. Para que no momento em que pausarmos o jogo, o texto seja ativado, utilizaremos o código de `ControlaJogador.cs`. Como declaramos `TextoGameOver` de forma pública (`public`), poderemos acessá-la por outros *scripts*. Em `ControlaInimigo.cs`, por exemplo, acrescentaremos um comando para acessar em `Jogador` o componente (`GetComponent`) de `ControlaJogador`, *script* no qual está a variável `TextoGameOver`. Para ativá-la ou desativá-la, utilizaremos `SetActive`. Assim, passaremos o valor de ativo. Entre parênteses (`()`) especificaremos se é verdadeiro ou falso, se está ativado ou não. Considerando que ao pausarmos estará desativado, entre os parênteses, colocaremos `true`. Salvaremos e minimizaremos `ControlaJogador.cs`, ao qual acrescentamos o seguinte trecho em `AtacaJogador`:

```
Jogador.GetComponent<ControlaJogador>().TextoGameOver.SetActive(true);
```

Ao pressionarmos "Play", veremos que o jogo roda normalmente e, quando os zumbis encostam na heroína, aparece a mensagem "Você Perdeu!".



Legal! O jogo está definido. Escapamos do zumbi e quando ele nos alcança, o perdemos e a mensagem é exibida. Agora, precisamos reiniciar o jogo, pois para nós é fácil clicar no "Play" para jogar. Mas para quem joga e vê a tela fora da Unity, é necessário um meio de reiniciar o jogo.

Para isso, ativaremos "Text" marcando a caixa de seleção em "Inspector" e, na parte de edição, vamos inserir "Enter" após a mensagem de perda e acrescentar "Clique para recomeçar!".

Na sequência, em "Scene", aumentaremos a caixa para visualizar o texto e atribuiremos `0` aos três eixos (X, Y e Z) em "Rect Transform". Se abrirmos "Game", veremos a nova mensagem centralizada.

Abriremos `ControlaJogador.cs` e criaremos uma variável (`Vivo`), do tipo Bool, equivalente à vida de "Jogador", logo no início do código, abaixo da declaração de `TextoGameOver` . Atribuiremos a ela o valor `true` , indicando que é verdade que "Jogador" está vivo.

```
public bool Vivo = true;
```

No método `AtacaJogador` que criamos, adicionaremos:

```
Jogador.GetComponent<controlaJogador>().Vivo = false;
```

Assim, indicamos que "Jogador" morreu. Em `FixedUpdate` , abaixo de `else` , adicionaremos um `if` para se "Jogador" `Vivo` for (`==`) falso (`false`), ou seja, se não estiver vivo.

Lembrando que um sinal de igual (`=`) serve para atribuir valor, ou seja, se tivéssemos utilizado **um**, estaríamos **afirmando** que `Vivo` é falso. Utilizando dois (`==`), estamos **perguntando** ao código se `Vivo` é falso ou não.

```
if(Vivo == false)
{
    if(Input.GetButtonDown("Fire1"))
    {

    }
}
```

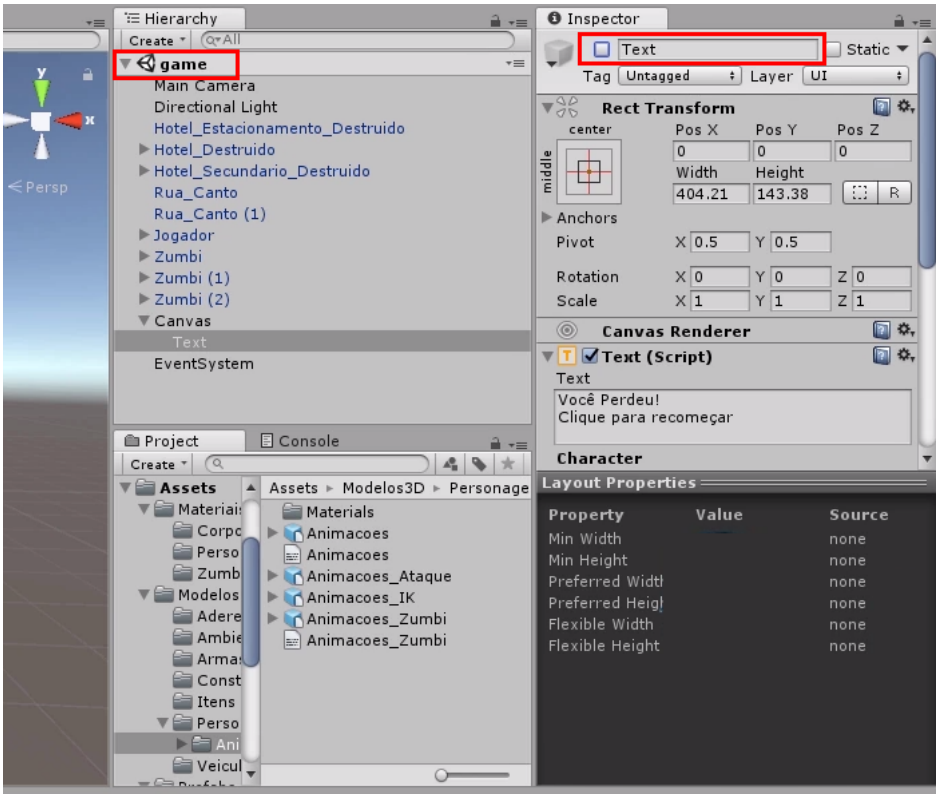
Entre as chaves (`{}`), colocaremos um `if` para que, se "Jogador" não estiver vivo, quando o usuário pressionar a tecla de clique — como se ele fosse atirar (`Input.GetButtonDown("Fire1")`) — o jogo deverá ser reiniciado. Para isso, no topo do código, abaixo de `UnityEngine` , adicionaremos:

```
using UnityEngine.SceneManagement;
```

Assim, estabelecemos que usaremos (`using`) a parte da Unity (`UnityEngine`) que lida com as cenas, (`SceneManagement`). Considerando que queremos reiniciar a cena que estamos utilizando, de volta às chaves de `if` , acrescentaremos:

```
SceneManager.LoadScene("game");
```

Dessa forma, acessamos `SceneManager` e damos `load` a uma cena. O nome da cena está em "Hierarchy", abaixo do botão "Create". No caso, usaremos "game" do jeito que aparece na Unity. Aproveitaremos para desabilitar "Text".



Salvaremos e minimizaremos `ControlaJogador.cs`. De volta à Unity, ativaremos "Play".

Notem que o jogo é reiniciado e permanece pausado, pois o jogo termina pausado. Podemos utilizar tanto o *script* de `ControlaJogador.cs` quanto `ControlaInimigo.cs`. Utilizaremos `ControlaJogador.cs` que é o ideal, pois é um só, então aplicamos somente uma vez. Nele, abaixo da declaração da variável `Vivo`, recriaremos o método `Start`:

```
private void Start()
{
    Time.timeScale = 1;
}
```

Por meio de `Time.timeScale`, voltamos a rodar o jogo em velocidade padrão `1` após reiniciá-lo em `Start`. Salvaremos e minimizaremos `ControlaJogador.cs`. De volta à Unity, ativaremos "Play" e veremos que está funcionando corretamente. A heroína se move e atira nos zumbis, que a perseguem e ao encostarem nela, o jogo é pausado e a mensagem de *Game Over* é exibida. Se clicamos na tela, a partida é reiniciada.

Ao reiniciar, a cena fica escura, mas não se preocupem. Isso acontece porque a Unity ainda está computando as luzes. Na hora que tirarmos o jogo da Unity, esse problema não se repetirá.