11

Novo gerador de Inimigos

Você pode fazer download do projeto Unity como ele está ao final desta aula clicando aqui (https://github.com/alura-cursos/unity-shooter-2/raw/master/Final_Aula04.zip)

Nós agora geramos uma posição e verificamos aquela posição para zumbis. Se aquela posição tiver algum zumbi, nós geramos uma nova posição, mas essa nova posição não é verificada.

Temos então que ficar verificando até encontrarmos uma posição satisfatória para gerar um zumbi.

Para isso vamos utilizar a estrutura `while` da programação (o `while` é o enquanto). Ou seja, enquanto o que estiver sendo testado não acontecer ele continua testando até conseguir.

Vamos então trocar nosso `if` para um `while` ?

```
void GerarNovoZumbi ()
{
    Vector3 posicaoDeCriacao = AleatorizarPosicao();
    Collider[] colisores = Physics.OverlapSphere(posicaoDeCriacao, 1, LayerZumbi);

    while (colisores.Length > 0)
    {
        posicaoDeCriacao = AleatorizarPosicao();
        colisores = Physics.OverlapSphere(posicaoDeCriacao, 1, LayerZumbi);
    }

    Instantiate(Zumbi, posicaoDeCriacao, transform.rotation);
}
```

Agora, enquanto ele não encontrar zumbis na área, ele vai ficar testando aquela área até achar uma boa posição para criar zumbis.

Essa é a melhor forma de gerar Zumbis.

Fazendo desta forma não teríamos problemas, mas ao colocar um `while` no Update da Unity ou utilizar uma estrutura complexa(que não é o nosso caso), podemos ter problemas. Para evitar qualquer problema independente do uso, vamos usar um método tipo de retorno de método especial, o `IEnumerator` .

Vamos então trocar nosso retorno método para o IEnumerator. Ele nos permite parar a execução, verificar o que temos até aquele momento e depois, se necessário, voltar de onde paramos. Para garantir isso usamos um tipo de retorno diferente conhecido como `yield` .

```
IEnumerator GerarNovoZumbi ()
{
    Vector3 posicaoDeCriacao = AleatorizarPosicao();
    Collider[] colisores = Physics.OverlapSphere(posicaoDeCriacao, 1, LayerZumbi);

    while (colisores.Length > 0)
    {
```

```
            posicaoDeCriacao = AleatorizarPosicao();
            colisores = Physics.OverlapSphere(posicaoDeCriacao, 1, LayerZumbi);
            yield return null;
        }

        Instantiate(Zumbi, posicaoDeCriacao, transform.rotation);
    }
```

O `yield return null;` faz com que o calculo seja feito, e se a posição estiver ocupada, nós esperamos até o próximo frame para continuar procurando a melhor posição. Assim, não travamos a Unity.

Na Unity, esse tipo de implementação é conhecido como **Coroutine**.

Para chamar este método, agora temos que usar a linha

```
StartCoroutine(GerarNovoZumbi());
```

Agora vamos executar algumas modificações no nosso script para ele ficar ainda melhor. A primeira delas é limitar os zumbis de serem criados enquanto o jogador está vendo, porque fica estranho os zumbis aparecerem na tela do jogador.

Então vamos limitar a criação para somente quando nosso Gerador estiver longe do personagem. Para isso, temos que ter uma variável que guarda uma referência ao nosso jogador.

```
    private GameObject jogador;

    void Start()
    {
        jogador = GameObject.FindWithTag("Jogador");
    }
```

Vamos criar uma variável para servir de referência de distância entre as duas, e comparar as duas posições no nosso Update.

```
private float DistanciaDoJogadorParaGeracao = 20;


    void Update () {

        if(Vector3.Distance(transform.position, jogador.transform.position) > DistanciaDoJogadorPara
        {
            contadorTempo += Time.deltaTime;

            if (contadorTempo >= TempoGerarZumbi)
            {
                StartCoroutine(GerarNovoZumbi());
                contadorTempo = 0;
            }
        }
    }
```

Agora, nossos zumbis estão sendo criados somente se a posição entre o Gerador e o Jogador for maior do que os `20` da variável `DistanciaDoJogadorParaGeracao` .

Para melhorar ainda mais, a nossa segunda modificação é visual, para podermos distribuir melhor os nossos Geradores pelo mapa.

Vamos extrair o valor do raio de geração para uma variável:

```
private float distanciaDeGeracao = 3;
```

E vamos substituir o valor no método de AleatorizarPosicao.

```
Vector3 posicao = Random.insideUnitSphere * distanciaDeGeracao;
```

Agora, vamos utilizar um método da Unity que cria um "ícone" para o nosso Gerador. Vamos criá-lo com a cor amarela e a forma de uma esfera cujo raio é de acordo com a nossa variável

```
void OnDrawGizmos()
{
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireSphere(transform.position, distanciaDeGeracao);
}
```