05

# Rotação do Zumbi

### Transcrição

Os zumbis estão perseguindo a heroína, mas eles a empurram quando na verdade deveriam parar e atacá-la ao se aproximarem.

Consideraremos que os inimigos irão parar quando chegarem perto do "Jogador", então precisaremos estabelecer uma distância e, a partir dela, fazer os cálculos para determinar se estão perto ou longe da heroína.

Abriremos "ControlaInimigo" e, em `void FixedUpdate()`, calcularemos a distância entre "Jogador" e "Zumbi". A Unity facilitará esse cálculo por meio de `Vector3.Distance`, que adicionaremos após declarar a variável `distancia`, do tipo `float`. O `Vector3` passará a distância, que será um número (`float`), entre as duas posições.

Após `Distance`, entre parênteses (`()`), colocaremos as duas posições (`transform.position` e `Jogador.transforms.position`, separadas por vírgula) que consideraremos para calcular a distância e a Unity fará o cálculo automaticamente. Assim, calcularemos a distância entre a posição do "Jogador" e os zumbis.

Na sequência, especificaremos que quando os zumbis chegarem em determinada distância do "Jogador", ele deverá parar. Enquanto não chegarem nessa distância mínima, devem continuar andando. No código, faremos isso por meio de `if`. Se a distância for maior (`>`) que `2`, os zumbis se movimentarão. Então, passaremos o primeiro trecho de `void FixedUpdate()`, que faz os zumbis andarem, para as chaves (`{}`) de `if`.

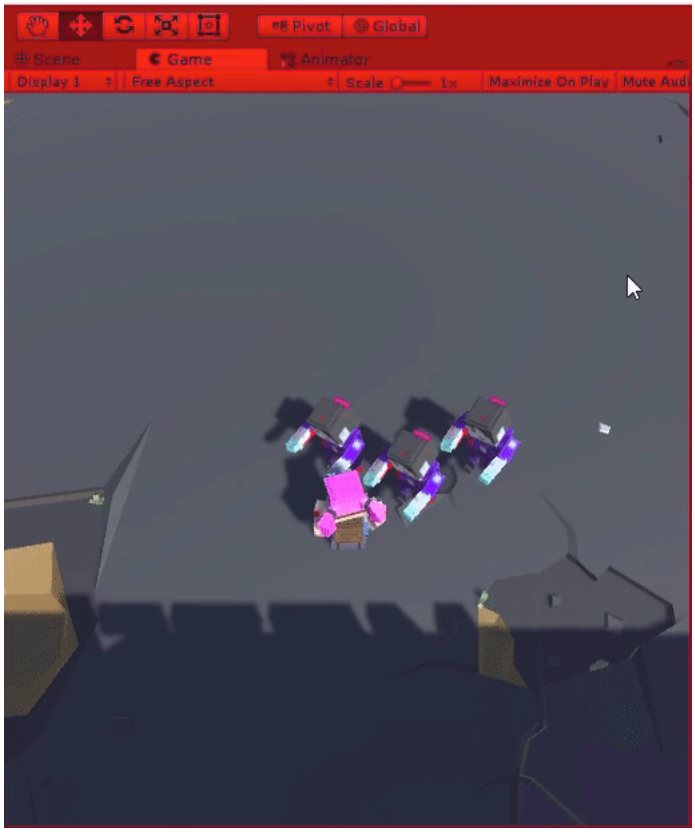
Em `if`, o valor `2` que estabelecemos como referência foi calculado considerando que o raio ("Radius"), do centro à borda, dos colisores da heroína e dos zumbis é igual a `1`. Ou seja, quando os colisores se baterem, se somarão (`1 + 1 = 2`). Poderíamos automatizar essa conta, mas como não alteraremos os valores da colisão, deixaremos como `2`.

Assim, estabelecemos que se a distância for maior que `2`, os inimigos andarão. Se não for maior que `2` eles não andarão. Salvaremos o trecho de código ficará da seguinte forma:

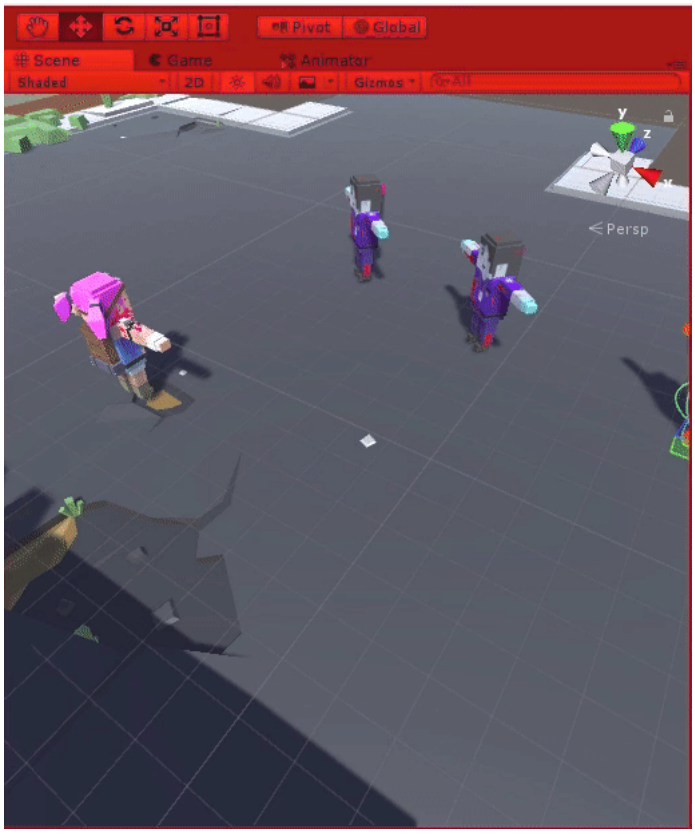
```
void FixedUpdate()
{
    float distancia = Vector3.Distance(transform.position, Jogador.transform.position);

    if(distancia > 2)
    {
        Vector3 direcao = Jogador.transform.position - transform.position;
        GetComponent<Rigidbody>().MovePosition
            (GetComponent<Rigidbody>().position +
            direcao.normalized * Velocidade * Time.deltaTime);
    }
}
```

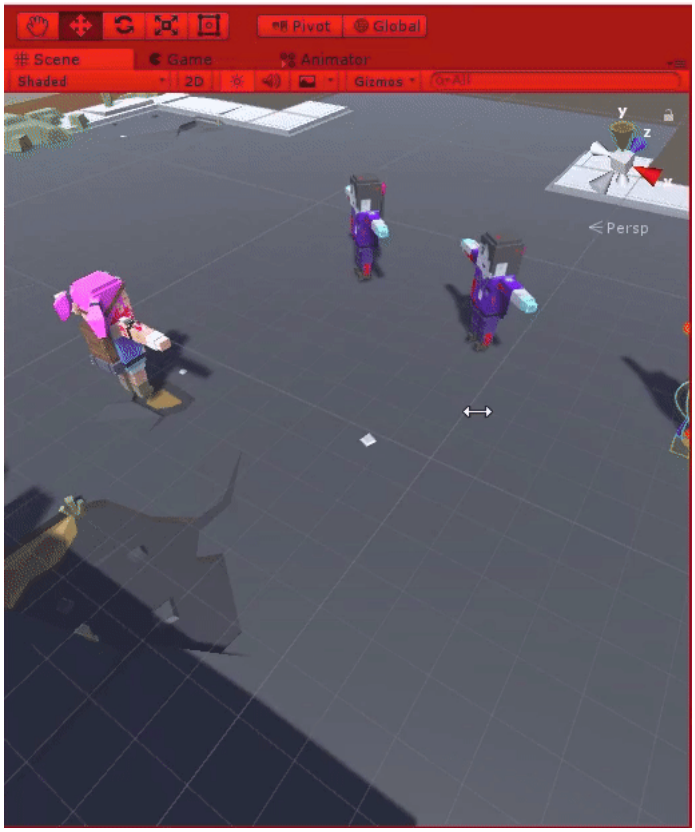
Ao clicarmos no "Play", veremos que os zumbis ainda estão empurrando a heroína, porque um empurra o outro na tentativa de levá-la.




Para corrigir, podemos aumentar o `2` que determinamos em `if`. Aumentaremos para `2.5`, salvaremos e ativaremos o "Play" na Unity. Notem que, agora, eles chegam nela e param de andar, mas continuam rodando a animação de caminhar. Veremos como resolver isso adiante.




Antes, faremos os inimigos olharem para a personagem. É estranho eles seguirem, sem olhar para o alvo. Quando ela está atrás deles, por exemplo, parece que eles a seguem andando para trás.




De volta ao código, faremos com que os zumbis virem de acordo com a posição da heroína. Eles só farão isso na perseguição, ou seja, quando estiverem se movimentando. Então, acrescentaremos dados ao trecho de `if`. A Unity facilita a vida do desenvolvedor de jogos, e disponibiliza `MoveRotation()` para rotacionar o objeto, conforme as coordenadas especificadas entre parênteses ( `()` ).

Criaremos uma variável do tipo `Quaternion`, estrutura utilizada para rotação, utilizando os eixos X, Y, Z e um imaginário que podemos criar para o cálculo das rotações. Mas, não nos aprofundaremos nisso. O importante é entendermos que **para rotacionar um objeto**, no código, utilizaremos uma **variável** do tipo `Quaternion`. Inclusive, se posicionarmos o mouse em cima de `MoveRotation()`, abre um aviso especificando que entre os parênteses deve ter um `Quaternion` de rotação.

A variável se chamará `novaRotacao` e atribuiremos a ela uma rotação baseada em `direcao`, por meio de `LookRotation`, que calcula a rotação de acordo com a posição ( `direcao` ) que passamos. Por fim, colocaremos `novaRotacao` entre os parênteses de `MoveRotation`. O trecho do código ficará da seguinte forma:

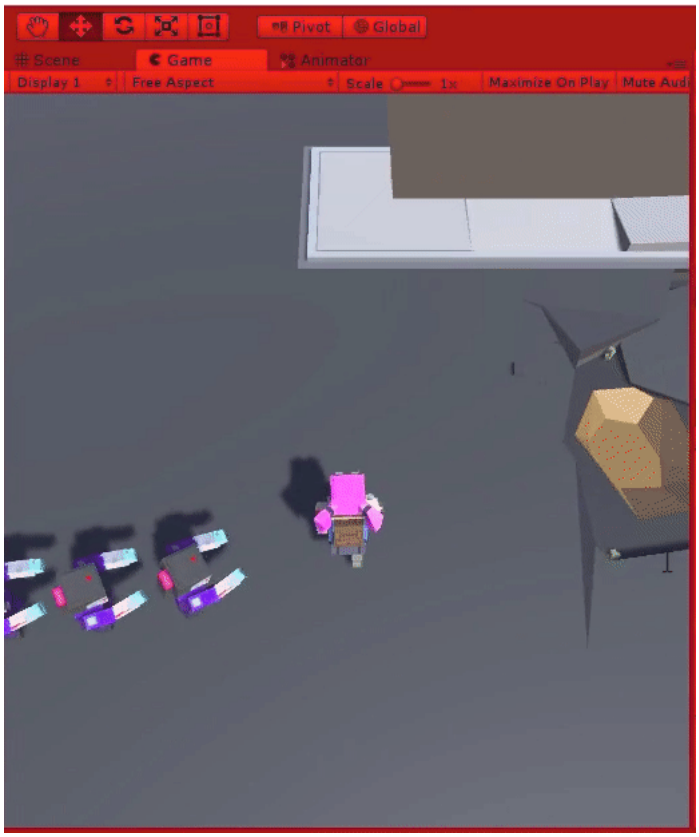
```
void FixedUpdate()
{
    float distancia = Vector3.Distance(transform.position, Jogador.transform.position);

    if(distancia > 2)
    {
        Vector3 direcao = Jogador.transform.position - transform.position;
        GetComponent<Rigidbody>().MovePosition
            (GetComponent<Rigidbody>().position +
            direcao.normalized * Velocidade * Time.deltaTime);

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

    }
}
```

Assim, calcularemos a rotação com base na posição do "Jogador". Salvaremos as alterações, minimizaremos o editor de texto e ativaremos o "Play". Agora, os inimigos andam voltados para heroína. Ao encostarem nela, pode acontecer de a empurrarem, mas na verdade estão se empurrando.




Dessa forma, fizemos os inimigos seguirem "Jogador".