03

Animações e Ataque

Nosso Chefe está perseguindo o personagem, mas ele ainda não está animado e nem está atacando.

Vamos então tratar isso?

No nosso inimigo, vamos criar as variáveis para referenciar o *script* das animações e status:

```
private Status statusChefe;
private AnimacaoPersonagem animacaoChefe;
```

E no `Start` vamos passar valor para elas:

```
statusChefe = GetComponent<Status>();
animacaoChefe = GetComponent<AnimacaoPersonagem>();
```

Agora para fazer com que o nosso personagem ande, vamos no `Update` utilizar a parte de animação, passando a velocidade que o nosso NavMeshAgent está utilizando.

```
animacaoChefe.Movimentar(agente.velocity.magnitude);
```

Você pode também no final do `Start` passar para o seu `agente` a velocidade que definimos no seu `Status` :

```
agente.speed = statusChefe.Velocidade;
```

Agora para fazer o Chefe atacar, temos que primeiramente fazer alguns testes. Vamos verificar se um caminho até o nosso personagem já foi calculado. Se não fizermos isso, o Unity pode pensar que já concluímos o nosso caminho e atacar o jogador de longe.

```
if(agente.hasPath == false)
{

}
```

Depois disso, temos que fazer a verificação em algumas variáveis. A primeira é uma que o `NavMeshAgent` já tem, e que guarda a distância que falta para o nosso jogador terminar o caminho, a `remainingDistance` .

A segunda é que nosso `NavMeshAgent` também tem uma distância de parada, que podemos customizar no Inspetor mudando o **Stopping Distance**. Coloque algo próximo de `5` , e isso vai garantir que quando estivermos a 5 unidades de distância do final do caminho, ele já irá parar e dar o caminho como concluído.

Vamos então verificar se o nosso Chefe já está perto do nosso jogador por estas duas variáveis vendo se a `remainingDistance` é menor ou igual a `stoppingDistance` e guardar esse valor em uma variável.

```
if(agente.hasPath == true)
{
```

```
    bool estouPertoDoJogador = agente.remainingDistance <= agente.stoppingDistance;

}


if(agente.hasPath == true)
{
    bool estouPertoDoJogador = agente.remainingDistance <= agente.stoppingDistance;
    if(estouPertoDoJogador == true)
    {

    }
}
```

Agora basta fazermos o `if` e `else` testando esta variável, e chamar os métodos de atacar o personagem.

```
if(agente.hasPath == true)
{
    bool estouPertoDoJogador = agente.remainingDistance <= agente.stoppingDistance;
    if(estouPertoDoJogador == true)
    {
        animacaoInimigo.Atacar(true);
    }
    else
    {
        animacaoInimigo.Atacar(false);
    }
}
```

É sempre bom rotacionarmos o Chefe quando ele está atacando, para não ficar tão estranho.

Como já temos os métodos para rotacionar o inimigo, vamos só criar e chamar os Componentes que já temos:

```
private MovimentoPersonagem movimentaChefe;


void Start
{
    movimentaChefe = GetComponent<MovimentoPersonagem>();
}
```

Agora basta chamarmos o método de rotação, gerando uma direção como já fizemos anteriormente.

```
if(agente.hasPath == true)
{
    bool estouPertoDoJogador = agente.remainingDistance <= agente.stoppingDistance;
    if(estouPertoDoJogador == true)
    {
        animacaoInimigo.Atacar(true);
        Vector3 direcao = jogador.transform.position - transform.position;
        movimentaInimigo.Rotacionar(direcao);
    }
    else
```

```
        {
            animacaoInimigo.Atacar(false);
        }
    }
```

Lembre-se de marcar o seu **Rigidbody** como **Is Kinematic**, para não termos problemas na movimentação.