01

Rotação do jogador

Transcrição

Continuaremos o desenvolvimento do jogo. Notem que, ao ativar "Play", "Jogador" atira com o clicar do mouse e se movimenta com o pressionar das teclas; mas falta dinâmica. Os zumbis estão à direita da heroína, que só atira para frente.

Levando isso em consideração, faremos a personagem rotacionar e atirar para todos os lados, de acordo com o ponteiro do mouse. Para isso, abriremos o *script* `ControlaJogador.cs`, afinal, todas as dinâmicas da personagem principal foram aplicadas nele.

Adicionaremos o trecho de rotação em `FixedUpdate()`, pois utilizaremos `Rigidbody` — como fizemos com o inimigo — para rotacionar a heroína. Quando utilizamos a física (`Rigidbody`), temos que inseri-la `FixedUpdate`.

Precisamos estabelecer uma referência de rotação para a heroína, da mesma forma que fizemos com o inimigo, quando criamos uma rotação que o direcionou para a heroína. Para aplicar essa rotação em "Jogador", considerando a direção do mouse:

- criaremos uma variável do tipo raio (`Ray`);
- que sairá (`=`) da câmera principal (`Camera.main`, com "C" maiúsculo);
- pegará um ponto da tela e o transformará em um raio (`ScreenPointToRay`);
- entre parênteses (`()`), colocaremos para qual ponto da tela queremos apontar o raio, que será direcionado pelo mouse (`Input.mousePosition`).

Para vermos o raio na Unity e saber para onde ele está apontando, sem ficar abstrato, utilizaremos `Debug`, que procura erros (*bugs*). Na sequência, adicionaremos ponto (`.`) e `DrawRay`, que desenha o raio e permite a visualização de como ele ficou após a criação.

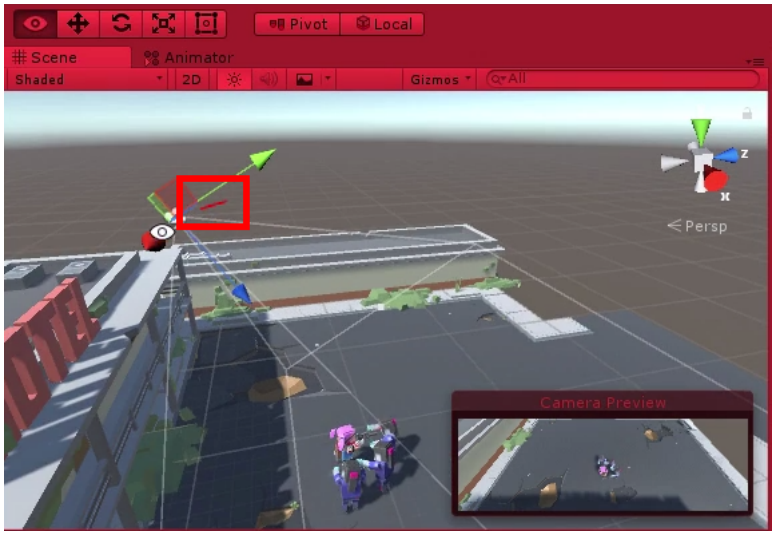
Quando desenhamos o raio, entre parênteses (`()`), precisamos especificar onde ele começa e a direção em que ele está. Esses dados estão armazenados na variável `raio`, então colocaremos `raio.origin`, vírgula (`,`) e a direção para qual o raio aponta (`raio.direction`). Em seguida, acrescentaremos outra vírgula para colorir o raio e torná-lo mais visível por meio de `Color.red`.

Salvaremos e minimizaremos o `FixedUpdate` de `ControlaJogador.cs` da seguinte forma:

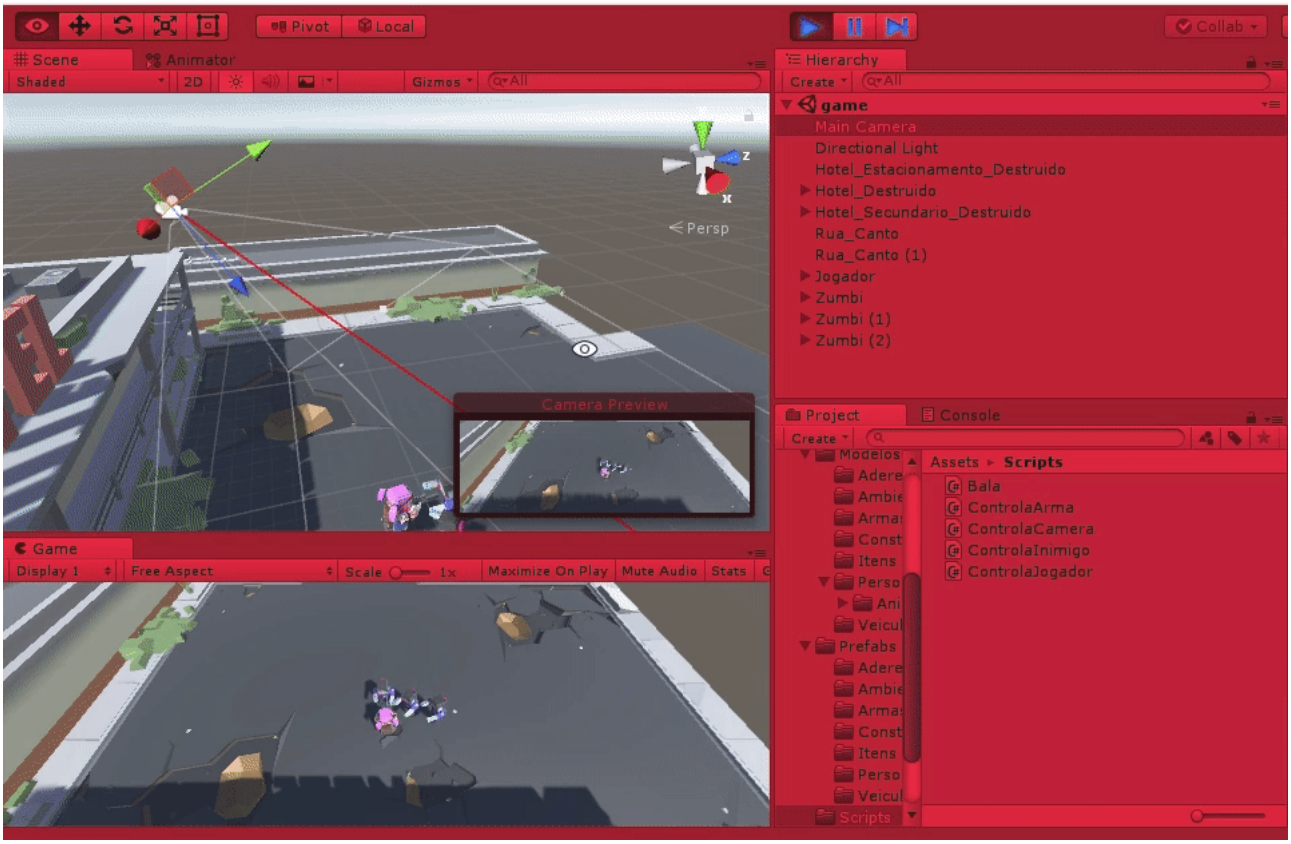
```
void FixedUpdate()
{
    GetComponent<Rigidbody>().MovePosition
        (GetComponent<Rigidbody>().position +
        (direcao * Velocidade * Time. deltaTime));

    Ray raio = Camera.main.ScreenPointToRay(Input.mousePosition);
    Debug.DrawRay(raio.origin, raio.direction, Color.red);
}
```

Ao desenharmos o raio na tela, veremos que ele é pequeno.



Para aumentá-lo, multiplicaremos (`*`) a direção (`raio.direction`) por `100` . Salvaremos e minimizaremos o *script*. Desativaremos "Play" — se estiver ativado — para ativá-lo novamente e visualizar a alteração no tamanho do raio. Arrastando a janela "Game" para baixo, veremos o raio mudar de direção, de acordo com o movimento do mouse.



Para que a personagem olhe em direção ao que o raio aponta, ativaremos o "Pause" por meio do atalho "Ctrl + Shift + P" ou "Command + Shift + P" no Mac. Assim, fixaremos o raio em uma posição e faremos a personagem olhar para o local onde ele toca o chão.

De volta ao *script*, faremos um teste, considerando que o raio em si não sabe onde ele encosta no chão. Para isso, utilizaremos `if` . Mas a variável `raio` não contém essa propriedade, portanto declararemos uma nova (`impacto`) do tipo `RaycastHit` , em `FixedUpdate` , que pegará o raio gerado e a posição em que ele bate ("hit" em inglês) em algo. A nomearemos como `impacto` , considerando que armazenará a posição em que o raio encosta no chão. A partir dela, saberemos a posição correta para qual rotacionaremos a personagem.

Na sequência, sabendo que `if()` é utilizado para testes, testaremos se o raio encosta no chão utilizando-o. Entre parênteses (`()`), colocaremos a parte de física (`Physics`) da Unity e geraremos um raio (`Raycast()`). Abriremos parênteses para `Raycast` , no qual colocaremos `raio` , combinado à variável `impacto` e a distância limite (`100`) para o teste de raio, que vai até o infinito, e não é o que queremos. `100` é um valor um tanto arbitrário, `50` talvez já desse conta, mas em vez de ficarmos testando diversos valores, estabelecemos `100` .

No `FixedUpdate` , que até o momento está da seguinte forma:

```
void FixedUpdate()
{
    GetComponent<Rigidbody>().MovePosition
        (GetComponent<Rigidbody>().position +
        (direcao * Velocidade * Time.deltaTime));

    Ray raio = Camera.main.ScreenPointToRay(Input.mousePosition);
    Debug.DrawRay(raio.origin, raio.direction, Color.red);

    RaycastHit impacto;
```

```
        if(Physics.Raycast(raio, impacto, 100))
        {

        }
    }
```

Será apontado um erro na parte de impacto, porque quando usamos `Raycast()`, a Unity solicita a utilização de `out` antes da variável (no caso, `impacto`). Notem que não atribuímos valor a ela, considerando que o raio não encostou em nada. Assim, `out` servirá para informar que `impacto` entrará sem valor, mas encontrará um dentro de `if`. No seguinte trecho:

```
if(Physics.Raycast(raio, impacto, 100))
```

Criamos o raio (`Raycast`) e juntamos a `impacto`, assim ele guardará a posição que o raio toca o chão. Então, entre as chaves (`{}`) de `if`, adicionaremos:

- `Vector3` para guardar a posição de impacto;
- `posicaoMiraJogador` nome da posição em que o jogador irá mirar;
- `=` para atribuir valor a `Vector 3`;
- `impacto.point` para pegar o ponto da variável `impacto`, pois o raio já está tocando o chão, gerando um ponto de impacto;
- `- transform.position` para subtrair (`-`) a posição de "Jogador", obtendo o ponto de impacto com base na posição dele, da mesma forma que fizemos anteriormente, movimentando o inimigo e a câmera;
- `posicaoMiraJogador.y = transform.position.y` para cancelar o eixo Y do raio, considerando que ele toca o chão e "Jogador" está em cima dele. Assim, igualamos a altura de ponto de impacto e "Jogador", fazendo com que ele não olhe para baixo ou para cima, dependendo de onde o raio toca.

Enfim, `FixedUpdate` de `ControlaJogador.cs` ficará da seguinte forma:

```
    void FixedUpdate()
    {
        GetComponent<Rigidbody>().MovePosition
            (GetComponent<Rigidbody>().position +
            (direcao * Velocidade * Time.deltaTime));

        Ray raio = Camera.main.ScreenPointToRay(Input.mousePosition);
        Debug.DrawRay(raio.origin, raio.direction, Color.red);

        RaycastHit impacto;

        if(Physics.Raycast(raio, out impacto, 100))
        {
            Vector3 posicaoMiraJogador = impacto.point - transform.position;

            posicaoMiraJogador.y = transform.position.y;

        }
    }
```