02

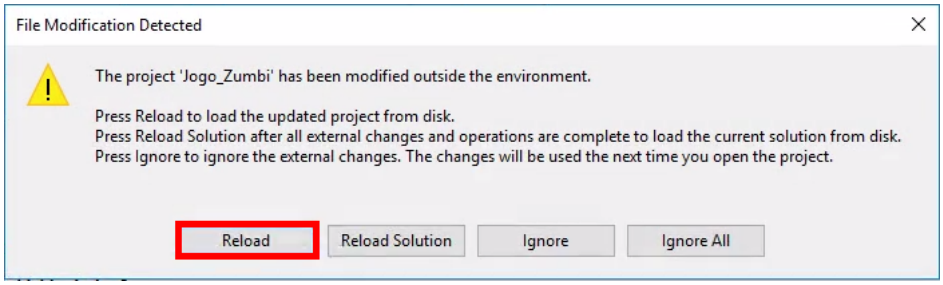
Zumbi seguir o jogador

Transcrição

Aplicamos animações aos personagens. Veremos como fazer os zumbis seguirem "Jogador", via *script*. Para isso, criaremos um novo clicando com o botão direito do mouse na pasta "Scripts", selecionando "Create > C# Script". Nomearemos como "ControlaInimigo", considerando que será utilizado para controlar os zumbis, que são os inimigos. Em seguida:

- selecionaremos "Zumbi", em "Hierarchy";
- arrastaremos "ControlaInimigo" para o "Inspector" dele;
- clicaremos em "Apply" para replicar a configuração nas cópias de "Zumbi".

Abriremos o *script*, clicando duas vezes nele. Abrirá uma mensagem de detecção de modificação no arquivo, avisando que o projeto "Jogo_Zumbi" foi modificado fora do ambiente de desenvolvimento. Clicaremos em "Reload".



Para que o inimigo persiga "Jogador", mesclaremos a movimentação dos personagens com a da câmera. Buscaremos o componente (`GetComponent<>`) de `Rigidbody` , no método `void FixedUpdate()` , como fizemos em "Jogador". Para isso, teremos que adicionar "Rigidbody" ao "Inspector" de "Zumbi".

Em "Constraints", marcaremos as caixas dos três eixos (X, Y e Z) de "Freeze Rotation" e a posição "Y" de "Freeze Position". Dessa forma, eles não rotacionarão aleatoriamente, nem cairão pelo cenário. Clicaremos em "Apply", para que a alteração seja aplicada em todos os zumbis.

De volta ao código:

- acrescentaremos `MovePosition` para mexer o inimigo, da mesma forma que fizemos com "Jogador";
- entre parênteses (`()`), colocaremos a posição em que eles estarão (`GetComponent<Rigidbody>().position`) perante à física e somaremos (`+`) a direção do movimento, estabelecida da mesma forma que fizemos com a câmera, criando uma variável pública (`Jogador`) equivalente ao objeto, no início do código.

No entanto, não atribuímos valor à variável `Jogador` em "Inspector", ainda. Se salvarmos as alterações da seguinte forma:

```
public class ControlaInimigo : MonoBehaviour {

    public GameObject Jogador;

    // Use this for inicialization
    void Start () {
```

```
    }

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

    }

    void FixedUpdate()
    {
        GetComponent<Rigidbody>().MovePosition
            (GetComponent<Rigidbody>().position + )
    }
}
```

Na Unity, a variável não aparece em "Controla Inimigo (Script)", porque não concluímos uma linha no código.

Considerando que o número de *scripts* está aumentando, veremos uma forma de identificar e localizar os erros, clicando em "Window > Console". Abrirá uma janela com os erros da Unity, apontados com um ponto de exclamação (`!`) dentro de um octógono vermelho. O que aparece em um triângulo amarelo é somente um aviso, que podemos ignorar. Inclusive, podemos selecioná-lo e clicar em "Clear" para apagá-lo.

Podemos arrastar a janela de "Console" à direita de "Project" para facilitar o acesso. Os erros indicam que faltou preenchermos e concluir com ponto e vírgula (`;`) uma linha do código. De volta ao código, somaremos a posição do "Jogador" (`Jogador.transform.position`) e colocaremos ponto e vírgula (`;`) no final da linha.

```
    void FixedUpdate()
    {
        GetComponent<Rigidbody>().MovePosition
            (GetComponent<Rigidbody>().position +
            Jogador.transform.position);
    }
```

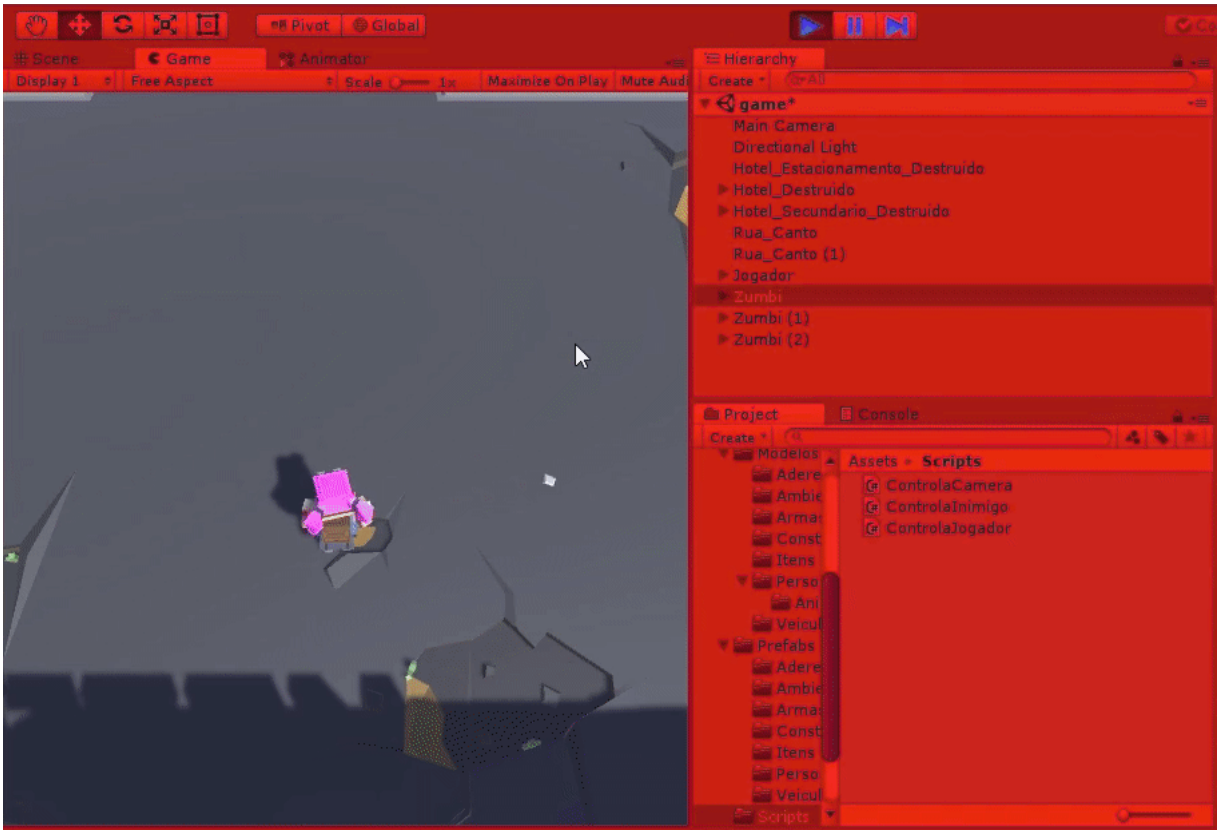
Salvaremos e voltaremos à Unity, na qual veremos que a variável de "Jogador" apareceu em "Controla Inimigo (Script)". Só falta preenchê-la. Se abrirmos "Console", há somente um aviso, os erros sumiram.

Preencheremos a variável, arrastando "Jogador" de "Hierarchy" para "Jogador" de "Controla Inimigo (Script)". Mas, notem que mesmo clicando em "Apply", esse ajuste não será aplicado às cópias, pois "Prefab" não salva esse tipo de configuração. Para aplicar em todos, podemos:

- selecioná-los clicando em um;
- pressionar a tecla "Shift";

- clicar no último;
- arrastar os três para "Jogador", em "Controla Inimigo (Script)".

Clicaremos em "Play" e veremos que os zumbis não aparecem. Pausaremos e clicaremos em "Play" para procurá-los e os encontraremos atrás dos hotéis, porque determinamos que a perseguição seria feita de acordo com a posição do "Jogador" e isso foi executado muito rápido.



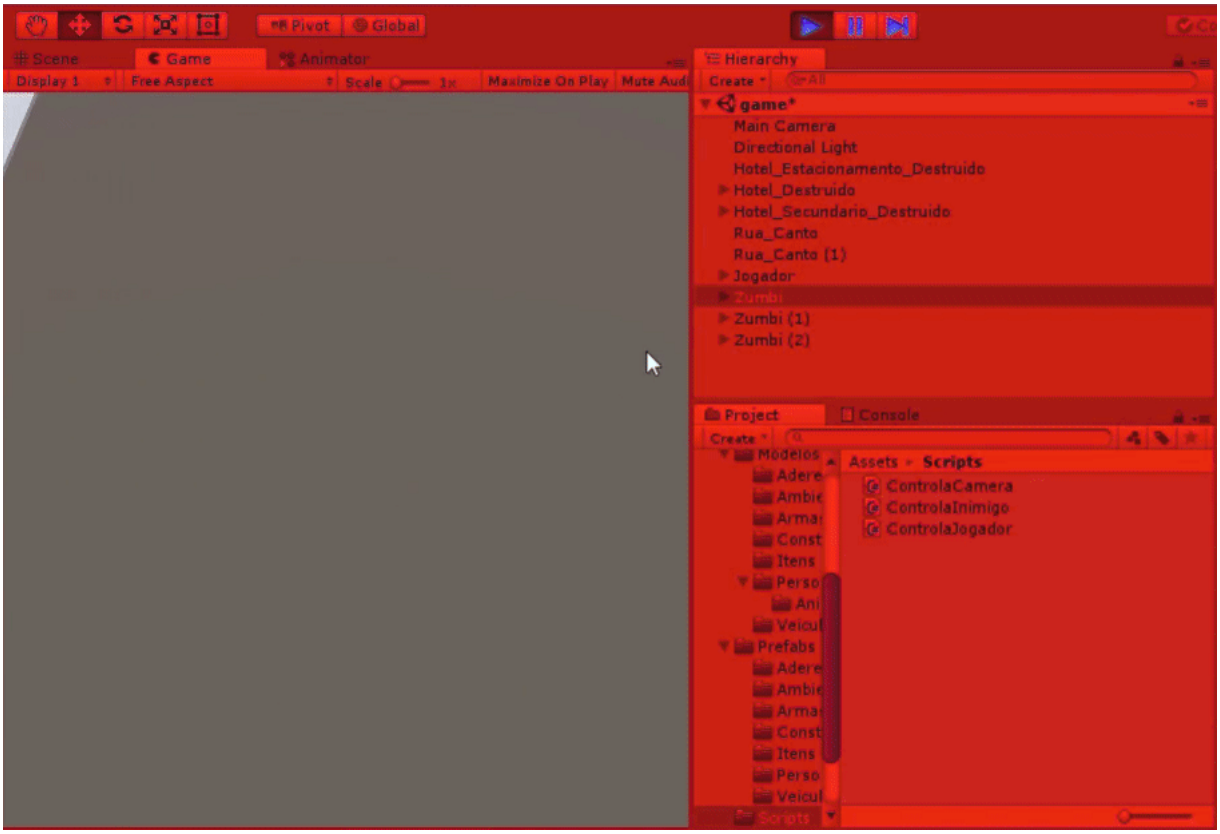
Dependendo do lugar que "Jogador" estiver, na hora que ligarmos o "Play", os zumbis irão para dentro do "Hotel_Destruido". Notem que eles não andam na direção do "Jogador", porque não estamos levando em consideração a posição inicial dele.

Queremos que os zumbis sigam a heroína, considerando a posição inicial deles. Quando os mandamos em direção ao "Jogador", consideramos que estavam no marco zero do mundo, ou seja, os três eixos de "Position", em "Transform", estavam zerados. Então, eles andam em direção ao zero do mundo, enquanto queremos que eles andem em direção ao "Jogador", considerando a posição inicial deles, como fizemos com a câmera, quando medimos a distância. Mas, nela fizemos o contrário, subtraímos a posição do "Jogador" da posição dela, porque queríamos estabelecer uma distância entre eles.

Em relação aos zumbis, queremos o contrário. Subtrairemos a posição em que eles estão, da posição do "Jogador". Semelhante ao que fizemos com a câmera, o código ficará da seguinte forma:

```
void FixedUpdate()
{
    Vector3 direcao = Jogador.transform.position - transform.position;
    GetComponent<Rigidbody>().MovePosition
        (GetComponent<Rigidbody>().position +
        direcao);
}
```

Salvaremos, minimizaremos o editor de texto e testaremos, clicando no "Play". Encontraremos os zumbis dentro de "Jogador", porque não ajustamos a **velocidade**.



Da mesma forma que fizemos em "ControlaJogador", precisamos calcular uma velocidade. Assim,

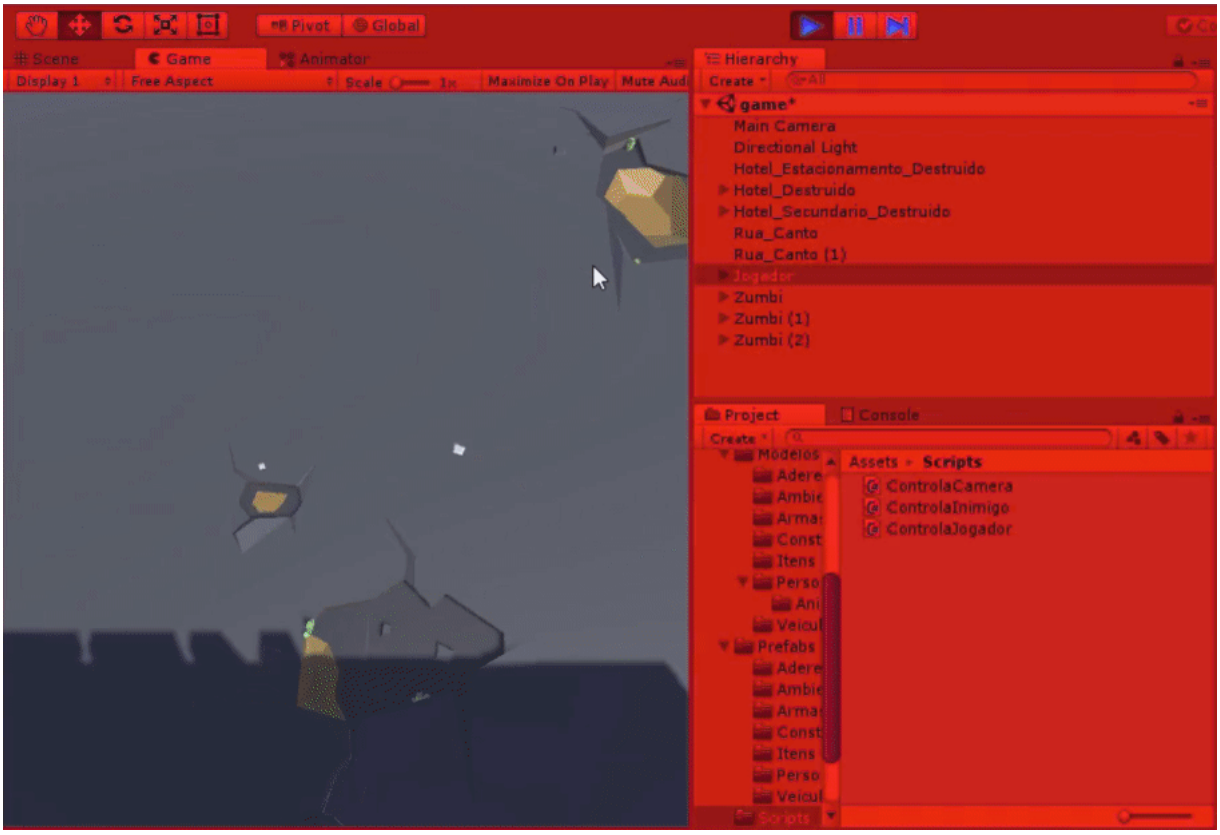
- criaremos a variável pública (`public`) `Velocidade` , do tipo `float` , no início do código;
- atribuiremos `5` como valor inicial, pois não faz sentido o zumbi ser mais rápido que "Jogador", que está com velocidade igual a `10` ;
- em `void FixedUpdate()` , multiplicaremos (`*`) `direcao` pela `Velocidade` e por `Time.deltaTime` .

O código ficará da seguinte forma:

```
void FixedUpdate()
{
    GetComponent<Rigidbody>().MovePosition
        (GetComponent<Rigidbody>().position +
        Jogador.transform.position);
}

public class ControlaInimigo : MonoBehaviour {

public GameObject Jogador;
    public float Velocidade = 5;

// Use this for inicialization
void Start () {

}

//Update is called once per frame
```

```
void Update () {

}

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

Após salvar e minimizar o arquivo de texto, ligaremos o "Play" e veremos que os zumbis vão em direção ao "Jogador", mas estão arrastando ele tão rápido que a câmera não consegue acompanhar.

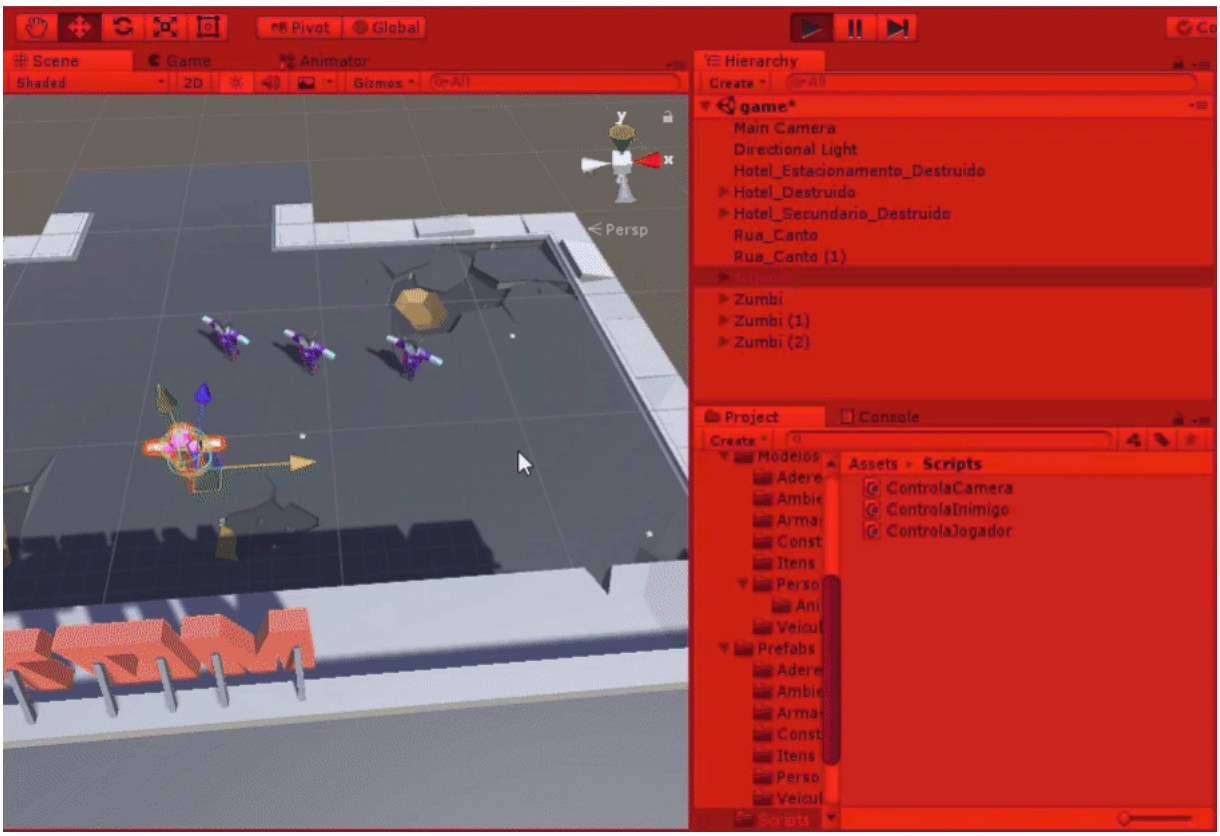


A heroína se movimenta de `1` em `1` quadradinho, enquanto os inimigos utilizam a distância até a personagem como velocidade. Para ajustar, precisamos aplicar o processo de **normalização**, que consiste em pegar a distância entre os personagens e igualar a do jogador, que se movimenta de `1` em `1`.

A Unity calcula isso por meio de `normalized`. Dessa forma, normalizamos uma grande distância, diminuindo-a para `1`. Aplicaremos no código, após `direcao`:

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

Salvaremos a edição e minimizaremos. Arrastaremos a heroína para o centro do cenário e clicaremos no "Play". Veremos que os zumbis a seguem e a empurram de forma lenta, além de se manterem virados para um lado.



O primeiro ajuste que faremos será para que os zumbis não empurrem a heroína. O objetivo é que ao encostarem nela, eles parem para atacá-la. Veremos como fazer isso adiante.