# Desenvolvimento com Motores de Jogos I

## Aula 13 - Interface com o Usuário – HUD

# Apresentação

Olá, desenvolvedores! Chegamos à aula 13 de nossa disciplina! Estão animados para continuar o desenvolvimento de nossa interface? Hoje, faremos mais alguns avanços nessa parte, e esse assunto ainda será explorado durante mais aulas.

Na aula de hoje, criaremos um HUD (sigla de Heads-Up Display) para o nosso jogo! Esse componente é primordial a todos os jogos e serve para indicar ao usuário informações importantes, disponibilizando-as diretamente na própria tela, a fim de que ele possa visualizá-las sem precisar mover-se.

Nosso HUD conterá dois elementos principais – um contador de tentativas restantes e uma barra de energia! Acerca do contador de tentativas restantes, aprenderemos mais detalhes sobre a utilização de imagens e texto, o que já começamos a conhecer na aula passada. Além disso, veremos como criar um script capaz de alterar os valores contidos no campo de texto em tempo de execução, conforme novas vidas sejam ganhas ou perdidas.

Em seguida, conheceremos o componente de UI Slider. Ele pode ser utilizado em interfaces diversas, como elemento de seleção entre valores dentro de uma faixa. Isso é útil para menus de configuração, por exemplo. Em nosso caso, no entanto, utilizaremos esse componente de uma maneira um pouco diferente: removeremos toda a interatividade e o aproveitaremos apenas como uma barra, capaz de mostrar valores entre um limite mínimo e um máximo. Esses valores serão representados pela energia restante do personagem!

Ainda não veremos, na aula de hoje, como causar dano ao personagem ou como podemos importar inimigos (Cuidado! Mísseis!) para a nossa cena (assuntos da próxima aula!), mas já criaremos a parte da interface, a qual é importante para que esses valores sejam utilizados.

Veremos, também nas próximas aulas, mais sobre interface, até quando formos modificar o valor da energia restante. Entretanto, por agora, focaremos nesses novos elementos! Prontos? Então vamos à aula! \o\ \o/ /o/

## Objetivos

Ao final desta aula, você deverá ser capaz de:

- Construir interfaces com hierarquia entre seus componentes;
- Utilizar containers para criar grupos de elementos de UI;
- Alterar elementos de UI através de scripts;
- Utilizar o componente Slider de maneira interativa ou não.

# 1. Contando as Tentativas Restantes

A primeira nova interface a ser desenvolvida em nossa aula é a contagem de tentativas que o jogador possui, representada pela quantidade de robôs restantes e determinada na variável robotsLeft, em nosso script do Game Controller. Para começar, disponibilizamos o projeto até onde desenvolvemos na aula passada, aqui.

Antes de alterarmos qualquer script, no entanto, precisamos começar adicionando a nova interface à nossa cena. Faremos isso da mesma maneira como vimos na aula passada, pelo menu GameObject -> UI -> Text e GameObject -> UI -> Image. Utilizaremos uma combinação desses dois componentes para exibir a quantidade de tentativas restantes.

Ao criarmos os novos elementos, como vimos na aula anterior, serão criados dois outros objetos em nossa hierarquia, o Canvas e o EventSystem. Os nossos objetos serão colocados, por padrão, como filhos do Canvas. Alteraremos, porém, essa hierarquia. Para a nossa primeira interface ficar parecida com a vista na **Figura 1**, adicionaremos o texto como filho do objeto Image. Isso facilitará o posicionamento, na sequência.

**Figura 01** - Interface representando a quantidade de tentativas restantes.


**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

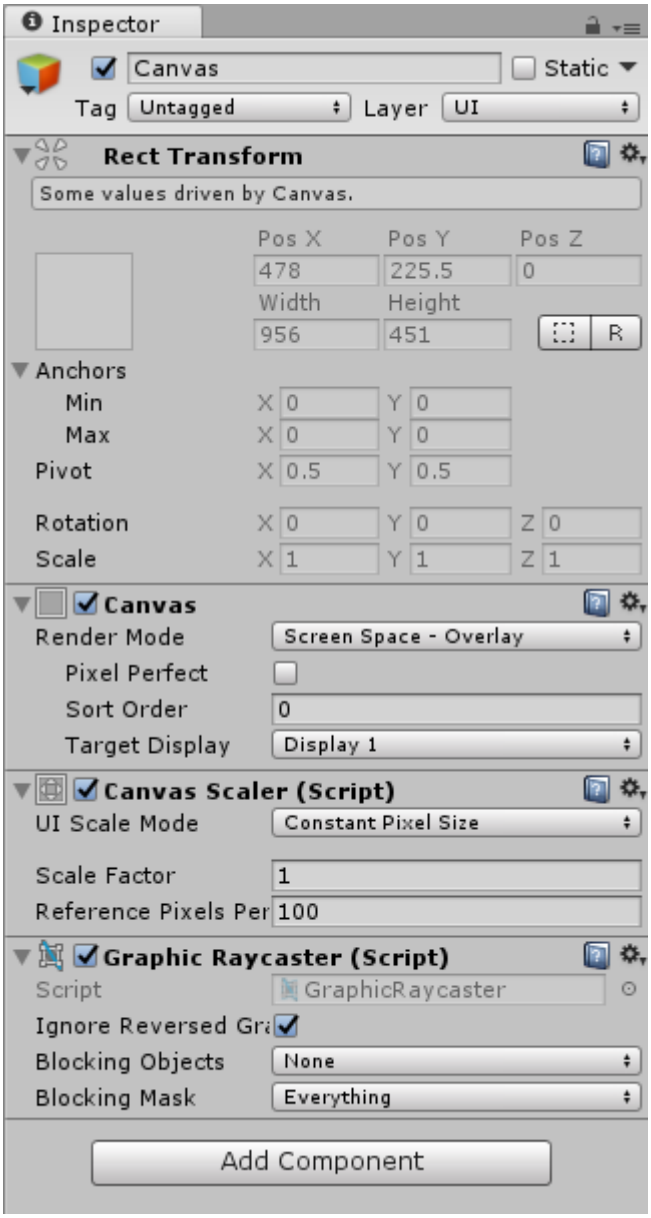
Essa interface será posicionada no canto superior esquerdo da tela e lá permanecerá a todo momento, uma vez que essa informação é relevante durante toda a execução do jogo. Para obter essa imagem utilizada, é só clicar aqui.

Objetivando que essa interface seja exibida sempre no mesmo local, podemos utilizar o Canvas com duas configurações diferentes.

De acordo com o que vimos na aula passada, utilizando o modo de Screen Space – Overlay, podemos posicionar a interface por último, mantendo-a sempre no local determinado em relação à tela. Já se escolhermos o método Screen Space – Camera e colocarmos a câmera padrão, a única disponível em nosso jogo, podemos reproduzir o mesmo comportamento, pois o desenho da câmera será sempre feito só, na cena, portanto a UI será desenhada por último.

Escolha um desses modos e configure o seu Canvas para ocupar toda a tela, como visto na aula sobre introdução à UI. Nós utilizaremos, por ser mais simples, o modo Screen Space – Overlay, colocando essa interface diretamente na tela, após todas as modificações já terem ocorrido e o cenário ter sido desenhado. O nosso objeto Canvas ficará configurado como vemos na **Figura 2**.

**Figura 02** - Objeto Canvas configurado para posicionar a nossa GUI.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

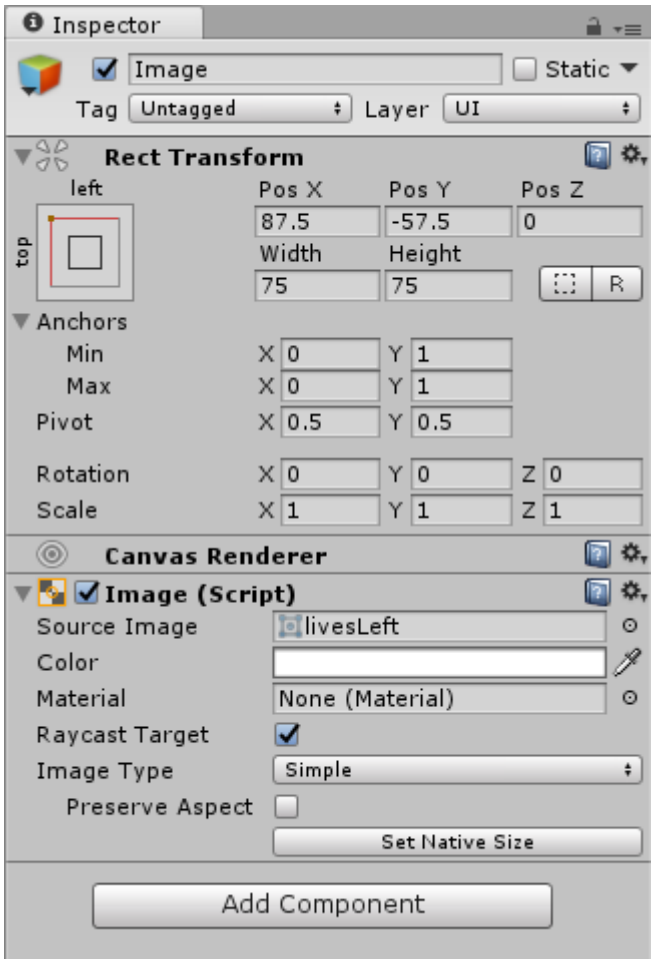
Com o Canvas configurado, o próximo passo é configurarmos a imagem e o texto para que eles se relacionem adequadamente e sejam posicionados no local correto do Canvas. Como a imagem foi colocada como objeto pai em relação ao texto, conforme dissemos anteriormente, começaremos configurando a imagem.

Importe o asset disponibilizado anteriormente, com a cara de nosso robô, e, então, selecione esse asset como o Source Image de nossa imagem da GUI. Isso fará com que a cara dele já apareça em nossa tela, facilitando a nossa noção quando posicionarmos o elemento. Agora, para a âncora!

Lembra de conversarmos na aula passada sobre âncoras? Que poderíamos ancorar objetos em relação aos seus pais e, assim, lidar com o posicionamento? É exatamente dessa maneira que posicionaremos os nossos objetos. Selecione a imagem e, então, escolha a âncora Top/Left para ela, incluindo o posicionamento (segure a tecla alt ao selecionar). Isso fará a base do posicionamento da nossa imagem ser o canto superior esquerdo da tela, o qual corresponde justamente a onde gostaríamos de posicioná-la, como dito anteriormente.

No entanto, ela ficará bem próxima à borda, prejudicando um pouco a visualização. Para melhorar esse aspecto, configuraremos a posição de nosso elemento. Com a âncora posicionada, ao utilizar o valor Pos X = 87.5 e Pos Y = -57.5, estaremos colocando o objeto para a direita e para baixo em relação ao canto superior da tela, chegando à configuração desejada. Por fim, eu gostei de utilizar o tamanho 75 x 75, em Width e Height. Fica a seu critério, porém, o quão grande você gostaria que sua interface fosse. Caso utilize um tamanho diferente, atente para reposicionar o texto, próximo elemento a ser alterado. As propriedades da imagem podem ser vistas na **Figura 3**.

**Figura 03** - Propriedades da imagem após configurada.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

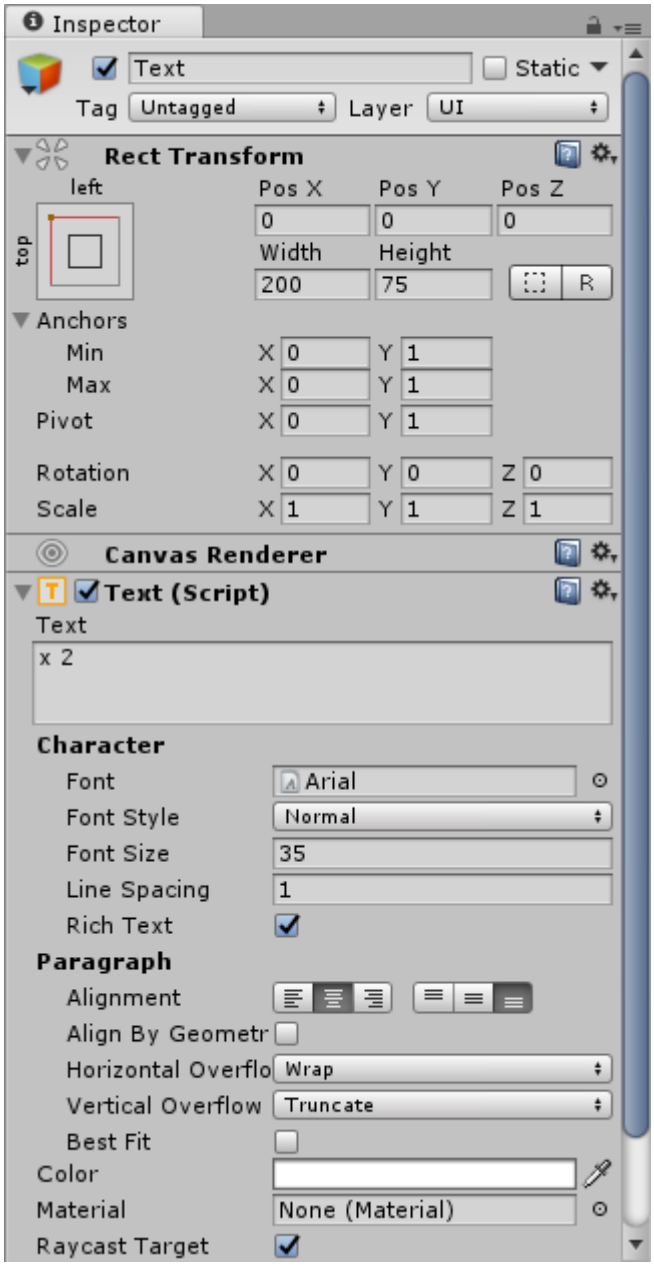
O elemento texto foi colocado como filho da imagem por um só propósito: aproveitar o seu posicionamento. Como esses dois elementos estão ligados na interface, é interessante que eles estejam também ligados hierarquicamente, de modo a garantir que, mesmo ao movermos a tela e os objetos, os dois componentes continuarão ligados, como deveriam estar.

---

Dito isso, configuraremos o objeto Text para também ter uma âncora com posicionamento e pivô (alt e shift!) na parte superior esquerda. Assim, o texto estará diretamente ligado à imagem e poderemos movê-lo à posição adequada. Mas ele já parece bem adequado! Basta alterarmos o seu tamanho e o seu alinhamento. Para o tamanho, utilizamos Width = 200 e Height = 75, a fim de suportar um Font Size = 35. O alinhamento é centralizado na vertical e inferior na horizontal, a fim de o texto

ficar próximo de como mostramos na **Figura 1**. Por fim, não esqueçam de alterar o texto para “x 2” (lembrem do espaço!) para termos o valor inicial já configurado. As propriedades alteradas podem ser vistas a seguir, na **Figura 4**.

**Figura 04** - Propriedades do texto, configurado como filho da imagem em nossa interface.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

Pronto! E com esses dois elementos posicionados, podemos, agora, começar a parte mais complicada da alteração da interface: o script em si. Mas fique tranquilo! Apesar de ser mais complicado que posicionar elementos, ainda é bem simples! :D

## 1.2 Alterando a UI Através de um Script

O elemento que guarda a quantidade de vidas restantes e que está presente em cada uma das cenas é o elemento Game Controller. Portanto, esse será o elemento o qual alteraremos para passar a suportar as alterações de interface. Abram o script do GameController e façam as alterações iniciais.

A primeira coisa necessária para o nosso script funcionar é uma referência ao objeto que queremos alterar. Assim, criaremos uma variável do tipo Text em nosso script do Game Controller. Para isso, no entanto, precisamos realizar uma nova importação em nosso código, dessa vez, da UnityEngine.UI.

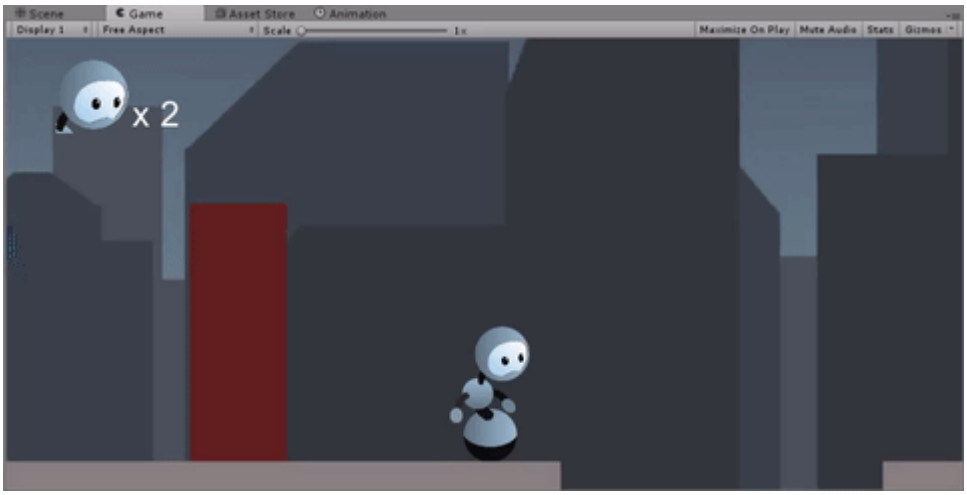
Em seguida, precisamos adicionar uma linha a mais aos métodos de fim de fase e de quebra – uma linha que altere o conteúdo do texto da variável Text criada para o novo valor da quantidade de vidas restantes, mesmo que esse valor tenha sido incrementado ou decrementado. Vejamos o código modificado na **Listagem 1**.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameController : MonoBehaviour {

    private int robotsLeft = 2;
    private string sceneName;
    public static GameController instance = null;
    public Text robotsLeftText;
    // Use this for initialization
    void Start () {
        if (instance == null) {
            instance = this;
        } else if (instance != this) {
            Destroy(gameObject);
        }
        DontDestroyOnLoad(gameObject);
    }
    public void Break () {
        robotsLeft -= 1;
        if (robotsLeft > 0) {
            robotsLeftText.text = "x "+ robotsLeft;
            Debug.Log(robotsLeft);
        }
        Invoke("RestartLevel", 3f);
    }
    public void LevelEnd () {
        robotsLeft += 1;
        robotsLeftText.text = "x "+ robotsLeft;
        Debug.Log(robotsLeft);
        Invoke("NextLevel", 3f);
    }
    private void RestartLevel () {
        sceneName = SceneManager.GetActiveScene().name;
        if (robotsLeft >= 0) {
            SceneManager.LoadScene(sceneName);
        } else {
            SceneManager.LoadScene("GameOver");
        }
    }
    private void NextLevel () {
        int sceneIndex = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(sceneIndex+1);
    }
}
```

**Listagem 1** - Adicionando uma alteração ao nosso elemento de UI.
**Fonte:** Elaborada pelo autor.

Pronto! Com essas alterações, já temos uma garantia de o valor do texto da UI ser alterado sempre que a variável for alterada. Para podermos testar, falta apenas irmos até a interface e arrastarmos o objeto de texto para o espaço da variável pública criada no script, a fim de definirmos quem é o objeto com o qual estamos lidando. Vejamos o funcionamento na **Figura 5**.

**Figura 05** - Elemento da UI sendo alterado quando o personagem altera a sua quantidade de chances restantes.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

Isso só tem um problema. Tente perder outra tentativa, após a primeira, e observe o console.

```
MissingReferenceException: The object of type 'Text' has been destroyed but you are still trying to access it.

Your script should either check if it is null or you should not destroy the object.
```

Eita. Deu erro. E agora?

A cena foi recarregada, não é? Se ela foi reiniciada, aquele objeto Text não existe mais, concorda? Um novo foi criado em seu lugar. Mas o nosso script ainda está referenciando o primeiro. E, então, as coisas dão errado. Bem errado! Mas como podemos resolver esse problema?

Não é complicado! Para solucioná-lo, podemos criar, em nosso script do GameController, um método novo responsável por, toda vez que um novo nível surgir, seja ele reiniciado ou fruto de um avanço ao próximo, encontrar o campo de texto daquele nível e atualizar a variável, bem como o seu conteúdo.

---

Esse método, no entanto, precisa ter uma interface um pouco diferente. Nós precisaremos adicioná-lo ao *delegate* do SceneManager e, para isso, ele terá de receber como parâmetro duas cenas. Não detalharemos esse processo, mas a assinatura do método será:

```
1 void GetRobotsLeft(Scene previousScene, Scene newScene);
```

E, então, precisaremos adicioná-lo ao SceneManager.activeSceneChanged. E depois removê-lo, no método OnDestroy(), que é o método chamado em um objeto quando ele deixa de existir. Com todas as alterações, o nosso código ficará como visto na **Listagem 02**.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class GameController : MonoBehaviour {

    private int robotsLeft = 2;
    private string sceneName;
    public static GameController instance = null;

    public Text robotsLeftText;

    // Use this for initialization
    void Start () {
        if (instance == null) {
            instance = this;
        } else if (instance != this) {
            Destroy(gameObject);
        }
        DontDestroyOnLoad(gameObject);
        SceneManager.activeSceneChanged += GetRobotsLeft;
    }

    void OnDestroy() {
        SceneManager.activeSceneChanged -= GetRobotsLeft;
    }

    void GetRobotsLeft(Scene previousScene, Scene newScene) {
        if (newScene.name.CompareTo("GameOver") != 0) {
            robotsLeftText = FindObjectOfType<Text>();
            robotsLeftText.text = "x " + robotsLeft;
        }
    }

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

    }

    public void Break () {
        robotsLeft -= 1;
        if (robotsLeft > 0) {
            robotsLeftText.text = "x "+ robotsLeft;
            Debug.Log(robotsLeft);
        }
        Invoke("RestartLevel", 3f);
    }
    public void LevelEnd () {
        robotsLeft += 1;
```

```
52        robotsLeftText.text = "x "+ robotsLeft;
53        Debug.Log(robotsLeft);
54        Invoke("NextLevel", 3f);
55    }
56
57    private void RestartLevel () {
58        sceneName = SceneManager.GetActiveScene().name;
59        if (robotsLeft >= 0) {
60            SceneManager.LoadScene(sceneName);
61        } else {
62            SceneManager.LoadScene("GameOver");
63        }
64    }
65
66    private void NextLevel () {
67        int sceneIndex = SceneManager.GetActiveScene().buildIndex;
68        SceneManager.LoadScene(sceneIndex+1);
69    }
70 }
```

**Listagem 2** - Código do script Game Controller com alterações para criar callbacks de alteração de cena.
**Fonte:** Elaborada pelo autor.

Foram poucas as alterações feitas, mas elas foram bem importantes, então, falaremos um pouco sobre cada uma. A primeira delas inclui o método criado na lista de métodos a serem chamados pelo SceneManager quando uma nova cena for carregada. Por isso, somamos o método ao que já está lá, utilizando o operador **+=**.

Em seguida, adicionamos ao nosso script o método OnDestroy() para remover o método GetRobotsLeft da lista de métodos a serem chamados pelo SceneManager quando o GameControlle deixar de existir. Assim, economizamos processamento em nosso jogo.

Por fim, temos o método GetRobotsLeft em si, onde faremos as alterações de UI toda vez que um nível for reiniciado. Como na cena de GameOver não teremos qualquer alteração, começamos o método testando se o parâmetro newScene, que contém justamente a nova cena carregada é diferente de GameOver. Se for diferente, procuramos o objeto do tipo Text daquela cena, com o método FindObjectOfType, o qual conhecemos antes, e então alteramos o seu valor para “x “ + robotsLeft, que é a junção de “x(espaço)“ com a quantidade de tentativas restantes.

Assim, resolvemos dois problemas com um código só. Primeiramente, quando iniciarmos uma nova cena, o novo valor de robotsLeft já será automaticamente levado até a UI, sem qualquer atraso ou necessidade de outro código. Segundo, agora temos sempre a variável robotsLeftText atualizada e pronta para ser utilizada pelos métodos de Break() e LevelEnd(), quando necessário. Tudo certo! Execute o jogo e teste o novo resultado!

Discuta com seus colegas no encontro presencial ou nos fóruns da turma os resultados que temos até aqui em nosso jogo! Será uma oportunidade de compartilharmos nossos avanços e esclarecermos as dúvidas existentes! ;)

# 2. Adicionando uma Barra de Energia

---

Agora que temos as tentativas restantes funcionando adequadamente, conheceremos um novo componente para adicionar à nossa UI, como a barra de energia restante ao nosso robô – o componente Slider, visto na **Figura 6**.

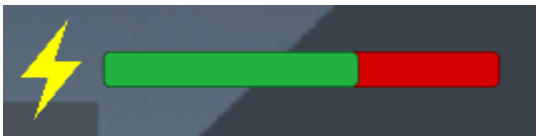
**Figura 06** - O componente Slider.

**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

Esse componente pode ser utilizado em diferentes locais de um jogo, principalmente quando colocado de modo interativo. Ele pode representar o volume em um jogo, a qualidade gráfica, o brilho, etc. Assim, tudo que possuir um valor mínimo, um máximo e precisar ser alterado gradativamente pode ser representado por esse componente. Basta adicioná-lo, permitir que o usuário o altere clicando na bolinha e então, ao concluir a alteração de valor, uma função será chamada automaticamente para que algo seja feito com o novo valor. Bem interessante!

Mas não é dessa maneira que utilizaremos um Slider em nosso jogo, por agora. Adicionaremos um Slider não interativo, sem a bolinha de clique, para ser utilizado como maneira de representar a energia restante de nosso robozinho, afinal, essa é uma grandeza que possui um valor mínimo e um valor máximo, correto? E aí, após algumas alterações, o nosso componente ficará como visto na **Figura 7**.

**Figura 07** - Componente Slider utilizado para representar uma barra de energia.



**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

Legal, não? Pois vamos adiante, conhecer melhor esse componente e alterá-lo para que fique como o demonstrado na **Figura 7**!

## 2.1 Criando o Espaço para os Elementos de Energia

Como vocês observaram, na **Figura 7** a barra de energia não foi incluída unicamente. Como todos os elementos de um HUD (sigla de Heads-Up Display, que representa os elementos como energia, tentativas, etc., compondo a tela de um jogo) bem elaborado, o componente que representa algo vem posicionado com uma imagem ou um componente identificador qualquer, o qual permite ao usuário perceber o que é aquele elemento que está ali, na interface. Em nosso caso, adicionamos uma imagem junto ao Slider indicando que aquela barra mede a energia.

Uma prática comum no desenvolvimento de interfaces é criar um elemento pai que possa guardar e servir como referência para todos esses elementos filhos, parte de uma pequena interface. Já fizemos algo similar em aulas anteriores, quando criamos componentes vazios para guardar todas as plataformas, ou todos os backgrounds, por exemplo. Agora, no entanto, faremos isso para um conjunto de elementos de HUD.

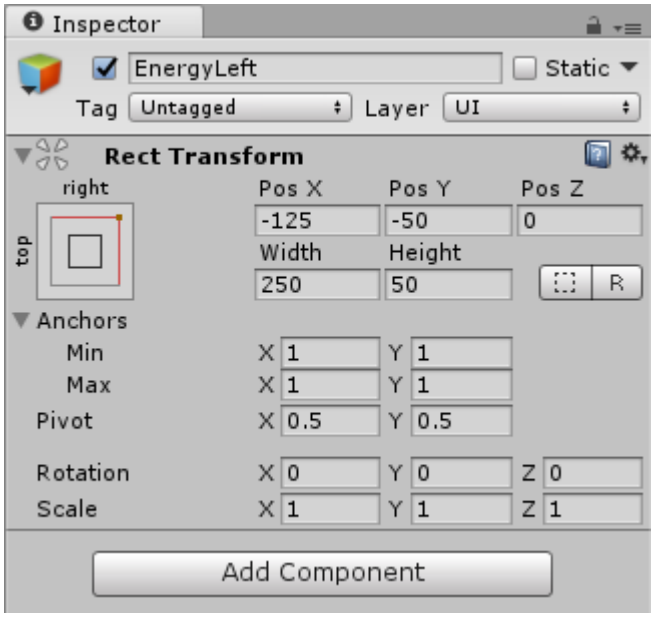
Para que possamos criar esses componentes novos, criaremos, primeiramente, um objeto filho vazio em nosso Canvas. Para isso, como já fizemos antes, clique com o botão direito no Canvas, na aba Hierarchy, e selecione a opção Create Empty. Ao fazer isso, um novo objeto denominado GameObject surgirá como filho do Canvas, como já estamos acostumados. Ao clicar nele, no entanto, você poderá notar uma diferença. Ele foi criado com um Rect Transform e não com um Transform padrão!

---

Isso é uma modificação interessante em relação ao que estávamos acostumados. Esse componente, ao ser criado como um Rect Transform, nos permite modificá-lo como um elemento da UI, mesmo sendo apenas um elemento vazio. Esse tipo de elemento é conhecido como *container* e é bastante utilizado! Agora vocês já o conhecem! :D

Vamos logo posicioná-lo em seu canto para que possamos, em seguida, adicionar os elementos de interface a ele. Como já utilizamos o canto superior esquerdo para a contagem de tentativas restantes, vamos utilizar o canto superior direito para a nossa barra de energia. Selecione o componente novo criado, altere o nome dele para EnergyLeft e então, na aba Inspector, selecione a posição e a âncora Top Right para ele. Em seguida, altere a Pos X = -125 e o Pos Y = -50. O Width utilizado foi 250 e o Height 50. O componente, após todas as alterações, deve ficar como visto na **Figura 8**.

**Figura 08** - *Container* que será utilizado para os elementos da barra de energia.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

Com o elemento já criado e posicionado, podemos passar aos elementos de interface em si. Começaremos pela imagem, que já conhecemos, e passaremos ao Slider na sequência.

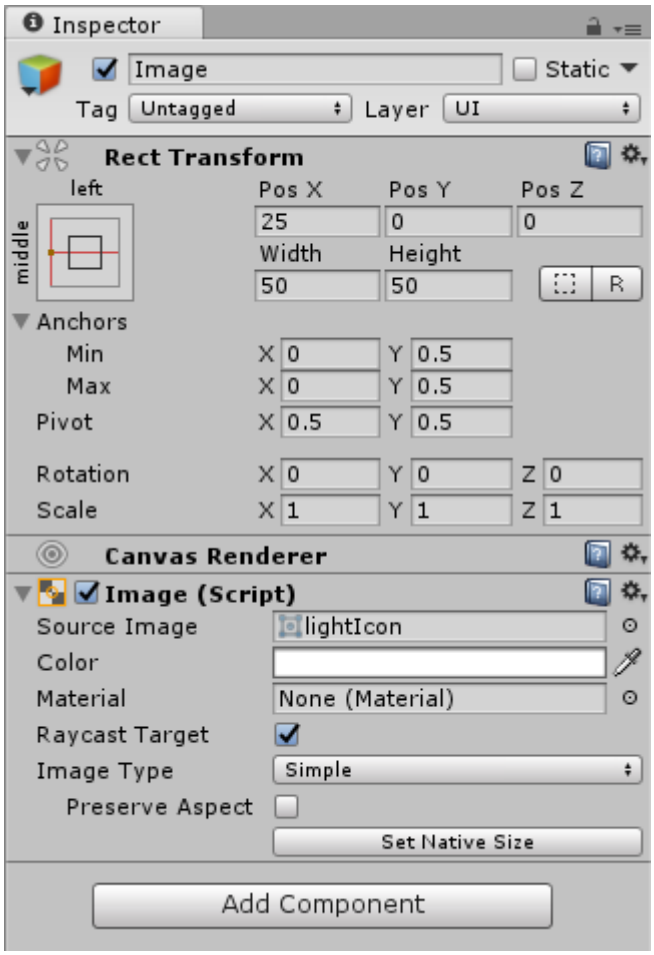
## 2.2 Adicionando a Imagem de Referência à Barra de Energia

Após construirmos o *container* necessário, vamos adicionar a imagem que indica a utilidade daquela barra verde, a qual posicionaremos em nossa cena. Se você quiser usar a imagem utilizada, você pode encontrá-la aqui.

Para adicionarmos a imagem, não há muita novidade. Apenas atentaremos para adicioná-la como filha do objeto EnergyLeft! Para isso, clique com o botão direito nesse objeto e selecione a opção UI -> Image. Em seguida, altere a imagem utilizada para o ícone desejado, arrastando a imagem até o objeto ou selecionando a propriedade Source Image dele para ser a imagem escolhida. Feito isso, o ícone de raio, ou qualquer outro que você tenha escolhido, já deverá estar aparecendo em nosso componente.

O próximo passo é alterar o tamanho e o posicionamento desse componente, através da âncora e das propriedades Width e Height. Para o nosso ícone, o tamanho 50x50 já é bom o bastante, então, selecione esses valores para as propriedades Width e Height. Em seguida, devemos movê-lo para o canto esquerdo horizontalmente, enquanto o mantemos centralizado verticalmente. Isso representa a âncora middle – left. Lembre-se de utilizar a tecla Alt para também mover a referência de posição do objeto. Feito isso, ele estará posicionado centralizado na borda de nosso *container*. Como não é esse o objetivo, altere a Pos X para 25. Isso o colocará no lugar correto, como visto na **Figura 7**. Os valores alterados do objeto Image podem ser vistos na **Figura 9**.

**Figura 09** - Valores para o objeto Image, filho de EnergyLeft.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

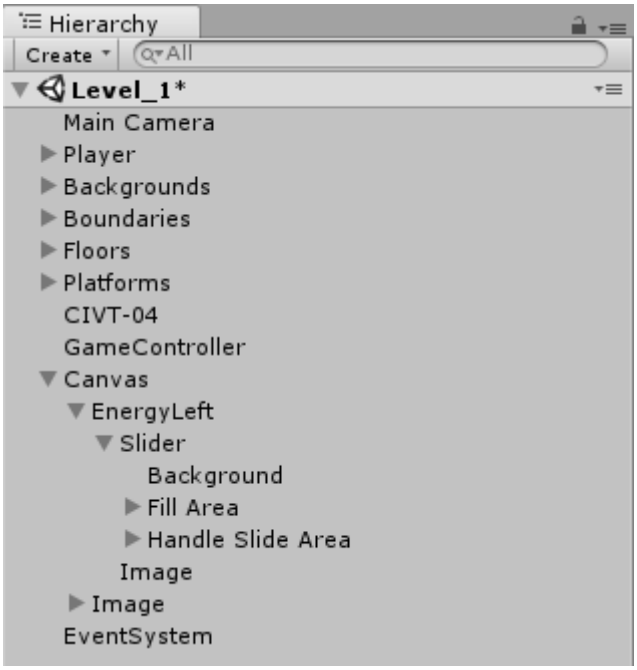
Com a imagem configurada, precisamos agora adicionar o nosso Slider à cena. Lembre-se que esse Slider não é interativo e vai representar a energia restante do personagem, de 0 a 100, a partir de uma variável do próprio Player. Não é objetivo dessa aula, no entanto, alterar o valor desse slider, uma vez que os elementos com dano só serão adicionados na aula que vem, mas já vamos preparar tudo de interface, certo?

---

## 2.3 Adicionando o Slider como Barra de Energia

O último objeto que adicionaremos à nossa interface de EnergyLeft é o Slider de energia restante. Para adicioná-lo clique com o botão direito sobre o EnergyLeft e selecione a opção UI -> Slider. Isso criará um novo Slider, como visto na **Figura 6**. Precisamos, no entanto, alterar um pouco esse componente para utilizá-lo em nosso projeto. Vejamos, na **Figura 10**, a estrutura de um Slider quando é criado.

**Figura 10** - Estrutura do Slider a ser utilizado para a Barra de Energia.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

O Slider, como vemos na **Figura 10**, ao ser criado, traz três elementos filhos. Esses elementos representam as partes do Slider. O background representa o fundo do elemento. O Fill Area representa o que vai cobrir o fundo à medida que o Slider altera o seu valor. Já o Handle Slide Area é o objeto que representa o Handler, a bolinha a qual permite ao usuário interagir com o Slider.

## Atenção!

Como não queremos que o usuário interaja com o nosso Slider, primeiramente devemos deletar o filho Handle Slide Area. Isso fará a bolinha desaparecer, deixando-nos apenas com a barra em si. Vamos, então, ao Slider em si fazer algumas alterações em seu componente!

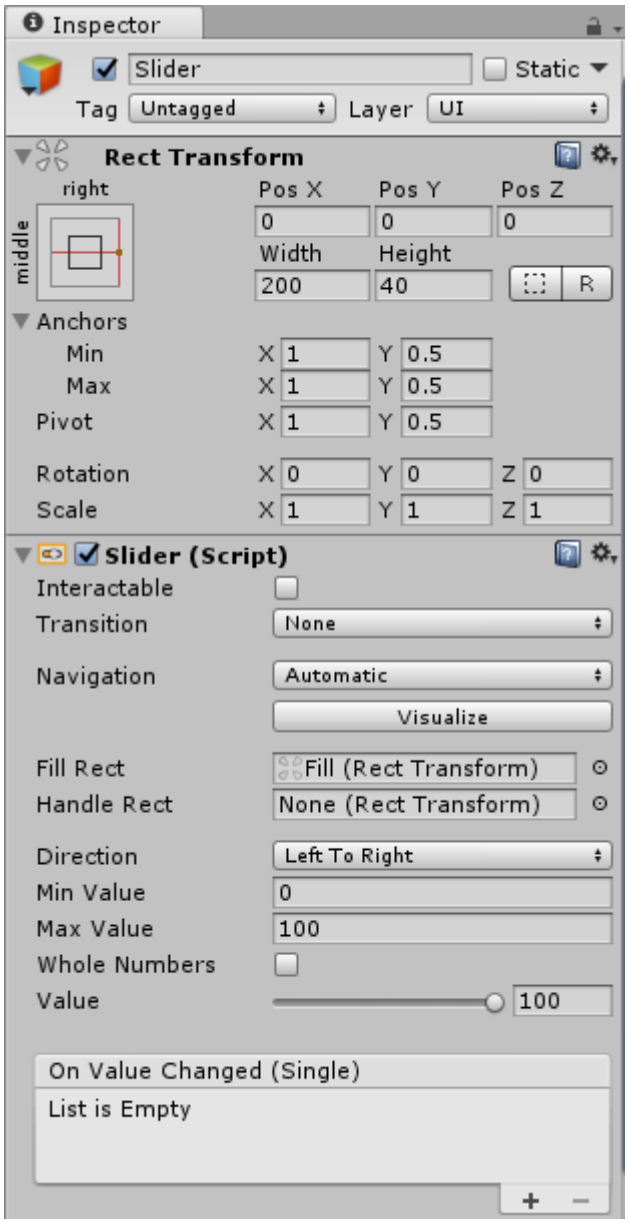
A primeira coisa que precisamos modificar no Slider é a propriedade **Interactable**. Essa propriedade define se é possível interagir com o objeto ou não. Como não queremos isso, devemos desmarcar essa propriedade, tornando essa barra não interativa. Em seguida, temos a propriedade **Transition**, a qual indica como será a transição entre os estados da bolinha de movimentação da barra.

Como não temos a bolinha e não utilizaremos isso para nada, podemos remover qualquer transição, alterando o valor dessa propriedade para None. Assim, todos os valores relacionados a ela desaparecem de nosso editor.

Ainda temos algumas propriedades a mais para alterar. A propriedade **Direction** indica a direção que a barra aumenta. Manteremos o padrão de Left to Right (da esquerda para a direita). Mas, logo abaixo, alteraremos o valor máximo da barra. Como trabalharemos com uma energia variável entre 0 e 100, alteraremos o **Max Value** (indica o valor máximo da barrinha) para 100. A propriedade **Whole Numbers** indica se a barra trabalhará com números inteiros ou ponto flutuante. Para termos uma maior flexibilidade, utilizaremos o ponto flutuante, deixando a opção desmarcada. Por fim, a propriedade Value indica o valor atual da barra. Como começamos com a energia cheia, podemos alterar esse valor para 100. Perceba que essa propriedade é um Slider para alterar o valor do Slider! :P

Alteradas todas as propriedades do Slider (script), precisamos alterar as propriedades do Rect Transform do Slider para posicioná-lo adequadamente em nosso container. Utilizamos a âncora oposta da imagem, a middle – right, alterando também a posição e o pivô para esses valores (alt e shift!). Por fim, para facilitar a visualização, alteramos a Width para 200 e a Height para 40. As propriedades finais do Slider podem ser vistas na **Figura 11**.

**Figura 11** - Propriedades do Slider para a barra de energia.




**Fonte**: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 17 de mar. de 2017.

---

Agora, aos filhos! O primeiro deles, Background, é um elemento do tipo Image o qual indica as propriedades do Background da barra, a ser exibido quando a barra estiver vazia. Como, usualmente, utilizamos a cor vermelha para indicar falta de energia e a verde para indicar que temos, vamos alterar o Background para vermelho, pois isso deve aparecer sempre que a energia estiver faltando. Então, selecione o elemento Background e altere a sua cor para vermelho (#FF0000FF). As outras propriedades podem ser mantidas como estão.

Já o segundo filho, Fill Area, é um elemento do tipo container, contendo um elemento do tipo Image que representa a parte preenchida da barra em si. O único detalhe que precisa ser alterado no Fill Area é que ele não representa a área da barra toda, uma vez que deixa o espaço da bolinha que nós deletamos. Como não temos mais o botão de interação, podemos alterar a Fill Area para preencher todo o espaço, em vez de deixar uma pequena sobra. Para isso, alteramos o valor de **Right** no Rect Transform dela para **5**, indicando que deve haver um espaçamento menor em relação ao fim do objeto.

Para o elemento Fill, filho de Fill Area, precisamos apenas mudar a sua cor para verde, uma vez que esse componente representa a barra quando estiver cheia. Utilizamos a cor #22BA43FF para esse elemento.

Após configurar o Slider e seus filhos, a barra de vida estará criada! Para ver suas alterações, basta selecionar o objeto Slider na aba Hierarchy e então, no Inspector, alterar o seu valor entre 0 e 100 e observar como a barra irá revelar a sua parte vermelha e se encher de verde novamente, quando voltar a 100. Muito legal, não acha?

Com isso, encerramos a nossa aula de hoje! Como já dissemos, a utilização dessa barra, com dano sendo causado e recuperado, será uma das cenas dos próximos capítulos. Fique ligado! Não perca! :D

Até mais, pessoal!! o/

# Leitura Complementar

Referências ao elemento Slider - https://docs.unity3d.com/ScriptReference/UI.Slider.html

Criação de um elemento de UI similar à nossa barra de vida - https://unity3d.com/pt/learn/tutorials/projects/survival-shooter/health-hud?playlist=17144

# Resumo

Na aula de hoje, aprendemos um pouco mais sobre a interface gráfica, conhecendo o HUD – heads-up display. Aproveitamos os componentes que já conhecíamos para desenvolver o nosso e, em seguida, alteramos o script para que pudéssemos alterar os valores do nosso HUD em tempo de execução.

No script, conhecemos alguns novos métodos, como o onDestroy() e os métodos Delegate do SceneManager. Vimos como podemos detectar uma mudança de cena e agir de acordo com isso. Aplicamos esse conceito a alteração de elementos da UI e detecção deles ao carregar uma nova cena.

Na segunda parte da aula, estudamos o elemento Slider e como podemos modificá-lo para construir uma barra de energia de acordo com algumas partes desse componente. Conhecemos as suas propriedades e as principais utilizações de cada uma delas.

Conhecemos também o conceito de *containers*, para agrupar elementos que fazem parte de uma mesma interface, facilitando o posicionamento e a ancoragem deles. Vimos isso para a barra de energia e também como o Slider utiliza isso por padrão. Deixamos, por fim, o projeto desenvolvido na aula disponível aqui.

Ficou para a próxima aula a alteração do elemento em código, a partir de algum dano sofrido pelo personagem. Esse dano poderá originar de inimigos criados dinamicamente, por exemplo. Como fazer isso? Veremos na próxima aula! Até lá! o/

# Autoavaliação

1. Como podemos criar um método que responda a alterações de cenas?
2. Como referenciamos em código elementos da UI?
3. O que são containers e como eles podem ser utilizados?
4. Quais os três filhos de um componente Slider? Qual a utilidade de cada um?

# Referências


Documentação oficial do Unity - Disponível em: https://docs.unity3d.com/Manual/index.html

Tutoriais oficiais do Unity - Disponível em: https://unity3d.com/pt/learn/tutorials

RABIN, Steve. **Introdução ao Desenvolvimento de Games**, Vol 2. CENGAGE.