

Este um capítulo da obra **Introdução ao desenvolvimento de softwares para analistas do comportamento**
E-book gratuito, download completo em: <http://abpmc.org.br/publicacoes.php?inf=14>

CAPÍTULO 8

Unity: Criando jogos e outras aplicações multi-plataforma

Gerônimo Oliveira da Silva Filho
Pedro Gabriel Fonteles Furtado
Felipe Lustosa Leite
Hernando Borges Neves Filho

Sumário Completo da Obra

CAPÍTULO 1

Por que estudantes, profissionais e pesquisadores de psicologia deveriam aprender programação?

CAPÍTULO 2

Como ler este livro – e se tornar um programador

CAPÍTULO 3

A lógica de programação

CAPÍTULO 4

Introdução ao desenvolvimento de interfaces gráficas com Lazarus e Free Pascal

CAPÍTULO 5

Visual Basic.NET

CAPÍTULO 6

Introdução à linguagem de programação R aplicada à pesquisa e intervenção comportamental

CAPÍTULO 7

Introdução ao MED PC

CAPÍTULO 8

Unity: Criando jogos e outras aplicações multi-plataforma

CAPÍTULO 9

Ensino e pesquisa no século XXI: Um manifesto pelo ensino de programação na graduação em Psicologia




Como citar este capítulo (APA):

Silva Filho, G. O., Furtado, P. G. F., Leite, F. L. & Neves Filho, H. B. (2018). Unity: Criando jogos e outras aplicações multi-plataforma. Em H. B. Neves Filho, L. A. B. Freitas & N. C. C. Quinta (Orgs.). *Introdução ao desenvolvimento de softwares para analistas do comportamento* (pp. 138-155). Campinas: ABPMC.

ISBN: 978-85-65768-07-8



Unity: Criando jogos e outras aplicações multi-plataforma

Gerônimo Oliveira da Silva Filho |  |

Pedro Gabriel Fonteles Furtado |  |

Felipe Lustosa Leite |  |  |

Hernando Borges Neves Filho |  |  |

Unity não é uma linguagem de programação, mas sim uma *game engine* disponível para Windows, Linux e Mac. *Game engines* são, de acordo, com Goldstone (2009), “As porcas e parafusos que ficam nos bastidores de todo jogo virtual”. De maneira mais objetiva, são softwares equipados de bibliotecas para facilitar a criação de jogos. A mensagem de marketing do Unity a descreve como uma *game engine* para todos os casos, tendo suporte tanto a 3D quanto a 2D e um grande leque de plataformas para qual o Unity consegue exportar (como por exemplo dispositivos mobile, videogames e aparelhos de realidade virtual). Unity também conta com comunidades ativas de usuários que compartilham suas experiências e dúvidas. Links para algumas dessas comunidades virtuais são encontradas no final do capítulo.

O Unity permite programar em duas linguagens: C# e Javascript. Sua versão de Javascript é, na verdade, uma variante chamada informalmente de Unityscript, não tendo todas as funcionalidades da linguagem original. O uso de C#, atualmente, é baseado na versão 6.0 da linguagem (Lian, 2017) . Mais detalhes sobre as duas linguagens fogem do escopo deste capítulo mas disponibilizaremos algumas fontes para informações mais detalhadas ao final.

Alguns jogos bastante populares, disponíveis em diferentes plataformas, foram desenvolvidos usando o Unity, como: Monument Valley, Kerbal Space Program, Temple Run Trilogy, Assassin’s Creed: Identity, Hearthstone: Heroes of Warcraft, entre outros. Apesar de todo funcionamento do Unity ser voltado para o desenvolvimento de jogos, outros tipos de aplicações

podem tirar vantagem dos recursos oferecidos pela *engine*. No contexto de pesquisa e desenvolvimento de tecnologia em Análise do Comportamento, o Unity pode se mostrar como uma boa ferramenta de desenvolvimento de softwares em vários segmentos. De maneira geral, o Unity pode oferecer uma valiosa contribuição no desenvolvimento de quaisquer aplicações que utilizem recursos como: gráficos 3D, simulações físicas, networking, suporte multimídia e capacidade multiplataforma. Em pesquisas que envolvem comportamento social e cultura, por exemplo, sua capacidade networking pode ser bastante útil. Quando se trata de desenvolver programas que envolvem realidade-virtual, como em pesquisas com VRET (Virtual Reality Exposure Therapy), o Unity é uma opção atraente. Esses e outros recursos podem ser muito úteis para um Analista do Comportamento e serão melhores explorados na próxima seção onde discutiremos as vantagens e desvantagens no uso do Unity.

Adquirindo o Unity

O Unity possui uma modalidade gratuita que contempla todas as suas funcionalidades e está disponível para qualquer usuário ou empresa que não obtenha mais de 100.000 dólares de receita bruta anual. A maioria das funcionalidades que discutiremos ao longo do capítulo estão disponíveis nesta versão gratuita da *engine*, quando houverem exceções sinalizaremos que o recurso é relativo a uma ferramenta criada por terceiros .

Vantagens e desvantagens

Apresentamos aqui uma lista de seis vantagens e duas desvantagens para ajudar o leitor ou leitora a decidir se o Unity é a escolha certa para o desenvolvimento de sua aplicação.

Vantagem 1: Suporte a 3D facilitado

Unity vem com suporte a gráficos 3D, incluindo um editor de cena. É muito rápido e simples adicionar um modelo 3D a Unity e é tudo feito sem escrever uma única linha de código.

Vários aspectos que seriam complicados de programar, como, por exemplo, sistemas de iluminação, já estão prontos para uso.

Vantagem 2: Suporte a realidade virtual

Unity tem suportes à vários aparelhos de realidade virtual. Na realidade virtual, o jogador veste um óculos e interage com um mundo virtual, como se estivesse dentro dele. De acordo com Haydu, Kochann e Borloti (2016), a tecnologia de realidade virtual parece trazer vantagens para as intervenções psicoterapêuticas de transtornos de ansiedade, como as fobias. Várias pesquisas vêm sendo desenvolvidas nessa área e o Unity se mostra como uma ótima opção para o desenvolvimento desse tipo de aplicação.

Vantagem 3: Multiplataforma facilitado

Unity tem suporte a computadores, dispositivos mobile, videogames (Playstation, Xbox, etc) e os já citados aparelhos de realidade virtual. Tirando otimizações e bugs, a ideia é que tudo que você criar funcione do mesmo jeito em todas as plataformas, simplificando bastante o tempo de desenvolvimento de *software* que envolva mais de uma dessas plataformas. No contexto de pesquisa, a possibilidade de um desenvolvimento simultâneo para uma multitude de dispositivos, em especial para dispositivos mobiles , pode diminuir os custos de desenvolvimento dos experimentos, ajudar na coleta de dados, além de aumentar bastante a flexibilidade do pesquisador de como utilizar o software desenvolvido.

Vantagem 4: Suporte a networking

Nesse contexto, networking quer dizer utilizar a internet para permitir que mais de um usuário acesse o mesmo aplicativo. Unity permite sincronizar diferentes objetos e informações entre os computadores dos jogadores, economizando bastante tempo de programação desse aspecto. Ao final do capítulo deixamos uma indicação de um tutorial a esse respeito. Isso abre

um leque grande para pesquisas em comportamento social e cultural. Já encontramos algumas pesquisas utilizando softwares para simular microssociedades em laboratório (e.g Camargo, 2014). Aplicações desse tipo podem se beneficiar dos recursos de networking oferecidos pelo Unity.

Vantagem 5: Prototipagem rápida

No desenvolvimento de software muitas vezes é interessante criar uma versão incompleta, somente com funcionalidades básicas (ou até mesmo funcionalidades que parecem funcionar, mas na verdade não são flexíveis o bastante para solucionar o problema), para ver se um software é viável ou não antes de se comprometer mais recursos. Chamamos isso de prototipagem (Rouse, 2005).

Unity oferece diversos suporte para criar jogos ou aplicativos multimídia rapidamente: rápido suporte a imagens, modelos 3D, partículas, editor de scene, simulação de física, sons, animação, tratamento de entrada, saída, etc. Essa rapidez no desenvolvimento reduz o tempo e o custo associados a criar protótipos.

Vantagem 6: Visual scripting (pago)

Visual scripting é uma forma de programar conectando blocos. Leigos costumam ter mais facilidade com esse tipo de programação. Unity tem várias maneiras de prover isso mas elas são todas pagas, disponibilizadas por terceiros. Unity planeja, no futuro, adicionar *visual scripting* a *engine* (Unity Technologies, 2015). Quando isso acontecer, a expectativa é que seja algo incluso em todas as versões.

Desvantagem 1: Inflexibilidade

Para programadores experientes, pode ser complicado de se adaptar a maneira como a Unity organiza seus códigos de *script* e ela não oferece outras maneiras organizar o código e os

objetos da engine. Enquanto que em muitas linguagens o programador consegue fazer tudo por código, em Unity é necessário utilizar também a interface, o que pode não ser intuitivo para quem já tem experiência com programação.

Desvantagem 2: Performance

Processamento de grande volume de dados estatísticos e outras coisas, se possível, não deveria se realiza pelo Unity utilizando linguagens de script. Unity é otimizado para jogos e não para executar código complexo que pode demorar bastante tempo para ser executado. Pesquisas em Análise do Comportamento geralmente não desenvolvem softwares que geram dados pesados o suficiente para impactar a performance da Unity, então talvez você não precise se preocupar com isso.

Conceitos básicos e interface

Nesta seção são introduzidos alguns conceitos essenciais para que se entenda como o Unity funciona (Goldberg, 2009). *Assets*, *Scenes*, *Components*, *GameObjects* e *Scripts* serão apresentados e discutidos. Optamos por usar ao longo do capítulo os nomes dos elementos básicos do Unity em inglês para facilitar sua identificação já que não há uma versão do software em português.

O conceito de **Scene** (Cena) é central em Unity. Uma *Scene* pode ser pensada como um arquivo que guarda uma fase, uma área específica ou um menu do jogo. Dividir seu jogo em muitas *scenes* pode ajudar na prototipagem, uma vez que você pode testar suas *scenes* individualmente e de forma rápida. Para que o jogador consiga visualizar o que está havendo, é necessário que na *scene* exista uma **Câmera**. A câmera é configurada pelo programador para mostrar ao usuário o que se deseja.

Chamamos os arquivos usados em um projeto de Unity de **Assets** (Recursos) , e eles são armazenado em uma pasta com o mesmo nome.

Objetos em Unity são chamados de *GameObject*. Isso é um conceito diferente de programação orientada a objeto. Todo *GameObject* em uma *scene* tem posição, rotação e escalamento, isso diz respeito a um *Component* chamado *Transform*. Um **Component** (Componente) pode funcionar de várias formas, “Eles podem servir para criar comportamentos, definir aparência, e influenciar outros aspectos de uma função do objeto no jogo. Adicionando um *Component* a um *GameObject*, você pode imediatamente aplicar partes da *game engine* em seu objeto” (Goldberg, 2009, p.15). O *Transform* é um dos muitos componentes que já vem pré-definidos na *engine*.

Para um maior controle do funcionamento de seus *GameObjects* você pode programar scripts e adicioná-los como *components* nos *GameObjects*. **Scripts** são *assets* que contém a definição de uma classe em C# que é derivada de outra classe chamada de *MonoBehavior*. Todo Component, inclusive os predefinidos, são derivados de *MonoBehavior*.

Para entender o funcionamento da Unity é importante que você também esteja familiarizado com sua interface. A interface tem um papel central no desenvolvimento de aplicações quando usamos essa engine. Antes de prosseguirmos para um passo a passo de como desenvolver um aplicativo, vamos apresentar os elementos básicos da interface para que se torne mais fácil acompanhar nosso exemplo (Figura 1).

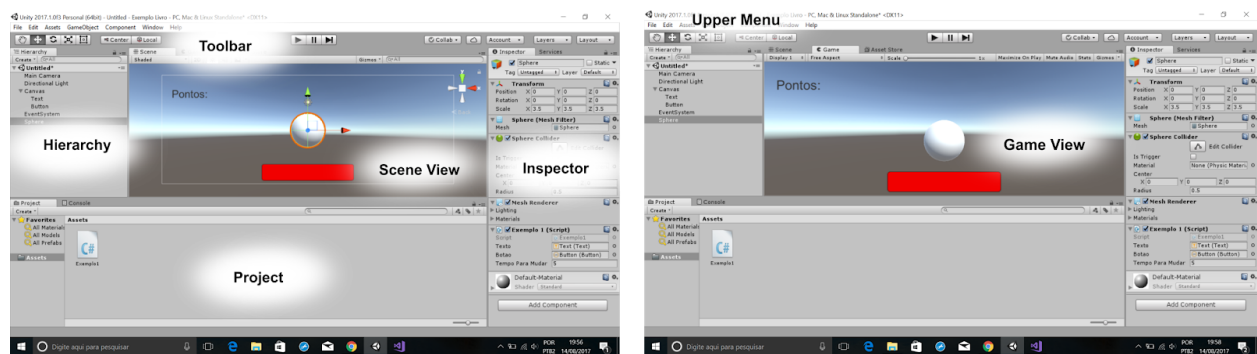


Figura 1. Principais elementos da interface do Unity em duas capturas de tela.

Como podemos ver na Figura 1, a interface da Unity é dividida em janelas com abas. Há algumas opções de layouts que você pode escolher. Dependendo da sua preferência e necessidade, você também pode mudar livremente as janelas e abas de local, agrupá-las,

desagrupa-las e fazer seu próprio layout. Neste capítulo, em nossas fotos utilizamos o *layout default*. Aconselhamos que por hora você faça o mesmo para facilitar sua execução do nosso exemplo mais a frente. Perceba que é possível identificar o nome de cada janela na aba que fica na parte superior esquerda de cada uma delas. Segue uma breve descrição das principais janelas da interface.

Scene View (Visualizador da cena)

O *Scene View* nos permite visualizar e editar nossa *scene*. É possível selecionar *GameObjects* no *Scene View*. As propriedades dos *GameObjects* selecionados irão aparecer no *Inspector* e poderão ser facilmente modificadas.

Game View (Visualizador do Jogo)

Como mencionamos, na Unity você precisa usar câmeras para mostrar sua *scene* para jogador. O *Game View* mostra a visão da(s) câmera(s) do seu jogo. Assim, através do *Game View* você pode visualizar o que o jogador verá quando ele estiver jogando. No *layout default* as janelas *Scene View* e *Game View* estão agrupadas e você pode visualizá-las clicando em suas respectivas abas.

Hierarchy (Hierarquia)

Na *Hierarchy* podemos ver uma representação textual de todos os *GameObjects* da *scene* e visualizar rapidamente como eles estão ligados entre si. Ao clicarmos em um *GameObject* na *Hierarchy* selecionamos ele na *scene* e também podemos acessar suas propriedades no *inspector*. Vale observar que apesar de todos os objetos que tem uma visualização na *scene* poderem ser encontrados na *Hierarchy*, nem todos os objetos que estão na *Hierarchy* tem uma visualização na *scene*. Esses objetos sem visualização tem informações de posição, rotação, etc, mas não são desenhados na tela, logo o jogador não sabe que eles existem.

Inspector (Inspetor)

Nos permite visualizar e modificar as propriedades de um objeto selecionado. No inspector é possível, por exemplo, mudar a cor de um botão, editar o conteúdo de um texto, modificar a localização de um *GameObject* na *scene* e também adicionar um *script* ao *GameObject*.

Project (Projeto)

Nos permite organizar os arquivos do projeto. Esses arquivos podem ser modelos 3D, músicas, sons, imagens, etc. Alguns arquivos selecionados aqui também podem ser modificados no inspetor.

Toolbar (Barra de ferramentas)

Nos dá acesso a algumas ferramentas essenciais. A esquerda temos as principais ferramentas para manipulação da *scene* e dos objetos na *scene* (Hand tool, Translate tool, Rotate tool, Scale tool) respectivamente. No centro temos os botões de play, pause e step que permitem começar um teste do jogo, pausar o jogo e encerrar um teste do jogo, respectivamente. À direita você pode acessar o Unity Cloud service e mudar o layout de sua interface.

Upper menu (Menu superior)

Permite diversas funções , como adicionar objetos, arquivos, scripts aos objetos, mudar as configurações do Unity e do jogo, etc.

Scene Gizmo (gizmo da cena)

O gizmo é um objeto interativo que mostra uma visualização do ângulo pelo qual a scene está sendo visualizada no *scene view*, como pode ser visto na Figura 2. Clicar no *scene gizmo* permite mudar rapidamente este ângulo.



Figura 2. O *scene gizmo* visualizado no *scene view*.

Exercício

Todos(as) aqueles(as) que já cursaram disciplinas básicas de AEC devem estar familiarizados com o processo comportamental a qual nos referimos por discriminação simples, que consiste, grosseiramente, em responder diferencialmente a um certo estímulo. Nos laboratório costumamos estabelecer um controle discriminativo entre uma luz e a resposta de pressão a barra. Quando a luz está acesa o rato recebe alimento ao pressionar a barra e quando a luz está apagada ele não recebe. Dizemos então que a luz acesa se torna um Estímulo Discriminativo ao passo que o rato passa a emitir consideravelmente mais respostas na presença da luz do que em sua ausência.

Escolhemos então simular em nossa aplicação um processo de discriminação simples, por meio de basicamente três elementos: um contador de pontos, uma esfera que muda de cor e um botão.

Não por acaso cada um dos três será responsável por um dos termos da tríplice contingência. A esfera, ou melhor, a cor da esfera, vai fornecer o contexto da resposta

(Sd/Sdelta). O botão será nosso operandum, é ele que vai permitir que o usuário emita uma resposta. Por fim, o contador de pontos será responsável por disponibilizar a consequência, uma vez que a resposta for emitida no contexto adequado.

Vamos construir um pequeno jogo onde uma esfera mudará de cor com uma certa periodicidade. Se o jogador apertar o botão enquanto a esfera estiver vermelha ele ganhará pontos a cada clique mas se ele apertar o botão e a esfera estiver branca, nada acontecerá. Então, vamos ao tutorial!

Usando Unity pela primeira vez

Se é a primeira vez que você está usando o Unity será necessário que você crie uma conta. É um processo simples que levará poucos minutos. Após sua conta ser ativada, basta clicar em New na aba Project e um novo projeto será criado. Para informações sobre o download e instalação do Unity disponibilizaremos um link ao final do capítulo.

Ajustando a visão

Com seu projeto criado e Unity aberto, primeiramente devemos ajustar a visão na scene para que possamos acompanhar visualmente nosso progresso. Para tanto, clique no **Scene Gizmo** com o botão direito do mouse e escolha a opção *Back*.

Criando o Objeto texto

Agora criaremos um objeto do tipo *Text* na interface do usuário. No **Upper Menu**, clique em GameObject > UI > Text. Observe que na **Hierarchy** apareceram dois novos itens : *Canvas* e *Text*.


Canvas é um espaço a parte do resto do jogo, onde ficam todos os elementos da Interface de Usuário (UI) que o jogador poderá interagir durante o jogo. Para tornar mais fácil a visualização do nosso exercício, optamos por renderizar o *Canvas* de frente para a câmera

principal de nossa scene. Para tanto selecione o item *Canvas* na **Hierarchy**. Agora no **Inspector** (Se você está usando o Unity pela primeira vez e a aba Services está sendo exibida, para visualizar o Inspector clique na aba com esse nome do lado direito da tela) procure o campo *Render Mode* e mude para a opção *Screen Play - Camera*. Ainda no Inspector, você deve clicar no pequeno círculo ao lado do campo *Render Camera*. Ao fazer isso será aberta uma janela, você deve selecionar a opção *Main Camera* e então fechar a janela.

Agora que seu *Canvas* pode ser visto próximo ao que vai ser exibido pela câmera, devemos trazer o objeto texto que você criou para frente da câmera. Para isso você deve selecionar o item *Text* na **Hierarchy**, dessa maneira você poderá ver as propriedades desse item no **Inspector**. Alterando os valores dos campos Pos (X,Y,Z) você pode modificar a localização desse objeto em cada uma das dimensões do espaço da *scene*. Para trazermos o objeto para o centro da *scene* podemos ajustar os valores dos três campos de Pos (X,Y, Z) para (0,0,0) ou clicar na engrenagem ao lado de *Rect Transform* e selecionar a opção *reset*.

Você já deve ser capaz de ver o objeto texto na *scene*. Agora procure a sessão *Text (script)* no inspector. Lá você vai encontrar um campo chamado *best fit*. Habilite o *checkbox* e como isso você verá o texto se expandir automaticamente.

Ainda na sessão *Text (script)* você vai encontrar o campo *Text*. Você deve ajustar esse campo de acordo com o texto inicial que você deseja que esse objeto apresente. No nosso caso, vamos escrever “Pontos: ”.

Agora você deve posicionar o texto onde ele deve ficar. Isso pode ser feito de duas maneiras: ou alterando os valores dos campos Pos(X,Y,Z) ou com o *mouse* diretamente na **Scene View**. Para posicionar com o mouse é só selecionar a Translate tool  na **Toolbar**, depois clicar no objeto que deseja mexer (no nosso caso, o texto) e puxar o eixo da dimensão em que você quer deslocar o objeto. Lembre que você pode ficar alternando entre as janelas **Scene View** e **Game View**. Na **Game View** você pode ver como seus objetos estão enquadrados pela câmera. Isso é importante na hora de escolher o lugar de um objeto, pois o usuário verá sua *scene* de acordo com o enquadramento de uma câmera. Em nosso exemplo, como o *canvas* está alinhado ao enquadramento da câmera, o enquadramento corresponderá a um contorno retangular que

você está vendo na *scene*. Para seu texto ficar exatamente na mesma posição da Figura 1, configure os campos Pos(X,Y,Z) com os seguintes valores (-270,90,0), respectivamente.


Criar botão

Vamos criar agora outro elemento da interface de usuário: um Botão. No **Upper Menu**, clique em GameObject > UI > Button .

Você criou um botão na *scene*. Agora vamos posicioná-lo onde ele deve ficar. Você pode mudar sua posição de modo similar ao que você fez com o texto, alterando o valor dos campos Pos (X,Y,Z) ou arrastando o botão na scene. Para seu botão ficar exatamente na mesma posição da Figura 1 configure os campos Pos(X,Y,Z) com os seguintes valores: (0,-110,0), respectivamente.

Com seu botão selecionado na **Hierarchy**, procure no **Inspector** o campo *Color*. Clique no retângulo branco do campo *color* para escolher a cor do seu botão. Escolha uma cor de sua preferência e depois feche a janela de seleção de cor. Para que seu botão fique da mesma cor do exemplo você pode copiar este código “FF0000FF “ no campo *Hex Color* da janela de seleção de cor.

Agora, para que nosso botão fique mais parecido com uma barra vamos deletar o texto do botão. Para tanto, basta expandir o item Button na **Hierarchy** clicando na setinha à esquerda do item. Com isso, o *GameObject Text* aparecerá abaixo do *GameObject Button*. Selecione o *Text* e aperte Delete no teclado. Perceba que agora o texto que estava no meio do botão desapareceu da scene.

Podemos também mudar as dimensões de nosso botão. Esse processo é similar ao deslocamento e ele pode ser feito tanto pelo ajuste de valores no *Inspector* como diretamente na *scene*. Para mudar pelo **Inspector**, você pode alterar os campos *Scale* (X,Y,Z). Cada um dos campos (X,Y,Z) diz respeito a uma das dimensões do espaço da scene. Você pode também selecionar a Scale Tool  na **Toolbar** e arrastar um dos eixos para redimensionar a dimensão correspondente, se você preferir modificar as dimensões na *scene*. Se você quiser que seu botão

tenha as mesmas dimensões da foto de exemplo você pode configurar os campos Scale (X,Y, Z) com os valores: (1.5, 1.5, 0), respectivamente.

Criando uma esfera

Chegamos ao último elemento de nossa scene: a Esfera. Diferente do Texto e do Botão ela não faz parte da interface do usuário, ela é um objeto 3D renderizado fora do *canvas*. Em termos simples, objetos 3D fazem parte da dimensão do jogo, onde ficam diversos outros elementos gráficos. No Canvas, só temos objetos de UI.

No **Upper Menu**, clique em GameObject > 3D > Sphere e a esfera será criada. Você pode redimensioná-lo exatamente da mesma maneira que fez com o Botão. Para que a esfera tenha as mesmas dimensões da foto do exemplo, configure os campos *Scale* (X,Y,Z) com os valores: (3.5, 3.5, 3.5), respectivamente.

É importante notar que a visão que você tem da esfera na **Scene View** é diferente da visão na Game View. Isso se dá pelo que discutimos antes sobre a esfera existir na dimensão do jogo enquanto que o Canvas existe numa dimensão separada. Eles são desenhados na câmera em momentos diferentes, então fique atento na hora de ajustar as dimensões da sua esfera.

Criando e adicionando um script

Vamos agora criar um *script* que vai servir para estabelecer o funcionamento dos elementos do nosso jogo e vamos adicionar esse script a nossa esfera. Para isso, selecione a esfera na **Hierarchy** e agora no **Inspector** clique em *Add Component*. Será aberta uma nova janela com uma lista de componentes que podem ser adicionados. Ao final da lista você vai encontrar o item *New Script*, clicando nele você poderá escolher o nome do *script* que você deseja criar. Em nosso exemplo optamos por “Exemplo1”. É importante que você nomeie seu *script* exatamente com esse nome, pois o código que você usará neste script constará com esse nome. Quando escolher o nome clique em *Create and Add*.

Escrever o script

Encontre o *script* que você acabou de criar em **Project** e clique nele duas vezes para abrir o programa de modificar *scripts*. Quando você clica, o Unity automaticamente vai abrir o programa apropriado se o programa já não estiver aberto. No Mac OS e no Windows, o Unity vai optar por instalar o Visual Studio, mas é possível utilizar o MonoDevelop. No Linux, a única opção seria o MonoDevelop.

Cole o código abaixo no espaço do código no programa que você estiver utilizando. Todas as linhas do código estão comentadas para que você possa entender o que cada linha faz. Após colar o código, clique em *Anexar ao Unity* na barra de ferramentas para que o arquivo seja atualizado no Unity.

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

public class Exemplo1 : MonoBehaviour
{
    //Variável que vai receber a esfera
    private Renderer esfera;
    //Variável para o texto da interface que vai fazer a contagem de pontos
    public Text texto;
    //Botão
    public Button botao;
    // Variável que vai guardar quantos pontos o usuário tem
    private int pontos;
    //Tempo que a esfera fica em cada estado
    public float TempoParaMudar = 5f;
    //Variável que conta quanto tempo falta para mudar o estado da esfera
    private float tempo;

    //método do Unity chamado quando o jogo começa
```

```

public void Start()
{
    //atribuí a variável esfera a esfera em questão
    esfera = GetComponent<Renderer>();
    //avisa o botão para alertar quando o jogador apertar o botão
    botao.onClick.AddListener(BotaoApertar);
}

//método chamado quando o botão é apertado
private void BotaoApertar()
{
    //se a esfera estiver vermelha
    if (esfera.material.color == Color.red)
    {
        //jogador ganha um ponto
        pontos++;
        //muda o texto que mostra os pontos
        texto.text = "Pontos: " + pontos;
    }
}

}

//Método chamado a cada frame (60 vezes por segundo)
public void Update()
{
    //Muda a variável de tempo de acordo com o tempo mudado
    tempo = tempo - Time.deltaTime;
    //se o tempo chegar a zero
    if (tempo < 0)
    {
        //reseta o valor do tempo
        tempo = TempoParaMudar;

        //se a esfera estiver vermelha
        if (esfera.material.color == Color.red)
        {

```



```

        //muda a cor do material da esfera para branco
        esfera.material.color = Color.white;
    }
    //se a esfera estiver branca
    else
    {
        //muda a cor do material da esfera para vermelho
        esfera.material.color = Color.red;
    }
}
}
}

```

Conectando as variáveis

Agora você deve conectar o seu objeto texto e o botão ao *script* que está na esfera. Para tanto, selecione a esfera na hierarquia. Você encontrará agora no **Inspector** uma sessão com o nome *Exemplo 1 (Script)*. Nesta sessão, você poderá configurar todas as variáveis que você declarou como públicas no seu *script*. Declarar muitas variáveis como públicas normalmente é uma prática ruim em outras linguagens de programação, mas no Unity é algo necessário para que possamos visualizar os campos do *script* no editor. Dito isso, incluímos em "Fontes para Aprofundamento" um *link* da documentação de uma maneira de evitar tornar suas variáveis públicas e ainda assim visíveis na engine.

No nosso exemplo deixamos três variáveis públicas, **Texto, Botão e Tempo Para mudar**. Os campos Texto e Botão devem receber os objetos que você criou em sua *scene*. Você deve clicar no círculo ao lado direito do campo. Fazendo isso, será aberta uma nova janela onde você deve selecionar o objeto que deseja atribuir ao campo. Faça isso com ambos os campos. A variável "Tempo Para Mudar" diz respeito ao tempo em segundos que demora para a esfera mudar de cor. Ajustamos o valor inicial para 5 segundos, mas você pode mudá-lo como desejar. Com esses campos devidamente configurados aperte o play na toolbar para testar seu jogo.

Fontes para aprofundamento

1. Tutoriais oficiais (inglês) - <https://unity3d.com/learn/tutorials>
 - a. Tutorial oficial para networking
<https://unity3d.com/learn/tutorials/topics/multiplayer-networking>
2. API (application programming interface, interface de programação do aplicativo)
 - a. <https://docs.unity3d.com/ScriptReference/>
 - b. Maneira de evitar marcar variáveis como pública:
<https://docs.unity3d.com/ScriptReference/SerializeField.html>
 - c. Muito importante, contém todas as classes e funções do Unity
3. Tutoriais no Youtube
 - a. Pesquisar Unity no youtube revela muitos vídeos educativos, inclusive em português
4. Tutorial em português na SBGames
 - a. <http://www.sbgames.org/papers/sbgames09/computing/tutorialComputing2.pdf>
5. Tutorial da fábrica de jogos
 - a. <http://www.fabricadejogos.net/posts/tutorial-criando-um-jogo-de-plataforma-em-unity-parte-1/>
6. Comunidades virtuais
 - a. Em inglês
 - i. <https://forum.unity3d.com/>
 - ii. <https://www.reddit.com/r/Unity3D/>

Referências

Haydu, V. B., Kochhann, J., & Borloti, E. (2016). Estratégias de terapias de exposição à realidade virtual: uma revisão discutida sob a ótica analítico-comportamental. *Psicologia Clínica*, 28(3), 15-34. Recuperado de http://pepsic.bvsalud.org/scielo.php?script=sci_arttext&pid=S0103-56652016000300002&lng=pt&nrm=iso

- Goldstone, W. (2009). *Unity game development essentials*. New York: Packt Publishing Ltd.
- Camargo, J. C. (2014). *Desenvolvimento Sustentável: Uma Análise Experimental do Comportamento de Extração de Recursos em Microsociedades de Laboratório*. (Dissertação de Mestrado). Programa de Pós-Graduação em Análise do Comportamento, Universidade Estadual de Londrina, Londrina, Brasil.
- Lian, A. (2017, 11 de julho). *Introducing Unity 2017 – Unity Blog*. Recuperado de <https://blogs.unity3d.com/en/2017/07/11/introducing-unity-2017/>
- Rouse, M. (2005, 12 de agosto). *What is prototype? - Definition from WhatIs.com*. Recuperado de <http://searchmanufacturingerp.techtarget.com/definition/prototype>
- Unity Technologies. (2015, 25 de junho). *Roadmap*. Recuperado de <https://unity3d.com/unity/roadmap>

