Programando em C# para Unity

Help
Ao trabalhar ou estudar sobre programação no Unity mantenha sempre esta janela aberta: http://docs.unity3d.com/ScriptReference/

Use e abuse da busca para encontrar detalhes sobre a programação.
Também fique atento às dicas do Editor quando autocompleta o código.

GameObject
O comportamento dos GameObjects é controlado pelos componentes que são anexados a eles. O Unity considera os script como componentes e estes dão interatividade aos objetos do jogo.

Script
Criar uma pasta chamada Scripts em Assets:
Clicar em Assets
Do lado direito clicar com o botão direito – Create – Folder (com nome Scripts)
Abrir a pasta criada com um duplo clique
Clicar com botão direito do mouse na área livre
Create – C# Script com nome JogadorScript

O Unity suporta nativamente 3 linguagens de programação. Quando criamos um script precisamos especificar que linguagem ele usará. Podemos usar JavaScript, CSharp ou Boo.

Quando criamos um script em C# ou Boo e o abrimos no editor, ele já nos mostra uma pequena estrutura com uma classe e a definição de duas funções: Start() e Update(). Se for JavaScript ele não tem classe, apenas as funções.

Recomenda-se que cada script contenha somente uma classe e seja somente sobre um único assunto.

Cada script somente pode usar uma linguagem de programação, mas nós podemos usar scripts das 3 linguagens se quisermos em um único jogo, o que não é comum nem recomendado.

Neste tutorial iremos utilizar a linguagem C# (Csharp).

Inicialização de Variáveis Públicas do tipo GameObject
Estas variáveis devem ser completadas pelo Inspector, arrastando para elas um GameObject da Hierarquia ou um Prefab do Assets.

Anexar o Script à Main Camera
Como estes scripts iniciais usarão apenas comandos da Console, anexaremos à Main Camera. Arraste o nome do script e solte sobre a Main Camera na Hierarquia.

Boa Prática
Nomear as variáveis tipo GameObject com o sufixo Prefab:
public GameObject balaPrefab;

Tecla de Atalho para Help Online no Monodevelop
Selecione a palavra sobre a qual quer ajuda e tecle
Ctrl + '

Orientação a Objetos
Objetos são instanciados (clonados) de classes. Uma classe é justamente um arquivo que traz as definições de seu objeto.
Quando rodamos nosso jogo no Unity cada inimigo deve ser equipado com uma cópia do objeto.

Abrir o Script no Editor
Ao efetuar um duplo clique no nome do script ele é aberto no editor Monodevelop. O Unity, para facilitar nossa vida, já mostra um template de script com alguns comandos como abaixo:

```
using UnityEngine;
using System.Collections;

public class JogadorScript : MonoBehaviour {

	// Use this for initialization
	void Start () {

	}

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

	}
}
```

Renomear Script
Quando criamos um script o Unity dá o nome NewBehaviourScript a ele.
Devemos renomer o mesmo antes que perca o foco. Caso demoremos a fazer isso, após renomear e quando abrir o nome da classe estará como o nome original do script, ou seja, NewBehaviourScript. Precisará renomear a classe para que tenha o mesmo nome do script e funcione corretamente.

Includes
As duas primeiras linhas:
using UnityEngine;
using System.Collections;

Incluem as bibliotecas do Unity para que fiquem disponíveis no script.

Classe
A linha:
public class JogadorScript : MonoBehaviour {

Cria uma classe que herda da MonoBehaviour.
Toda script tem uma classe criada obrigatoriamente com o mesmo nome do script.

Base para a Linguagem C#
As bases de uma linguagem orientada a objetos são as variáveis e os métodos. Em CSharp qualquer código deve ficar dentro de uma classe.

Orientação a Objetos
Como C# é uma linguagem orientada a objetos é mais coerente usarmos o termo método no lugar de função.

Variáveis
É uma pequena seção da memória do computador onde guardamos valores. Podemos comparar as variáveis a gavetas. Cada gaveta guarda o nome da variável, o tipo, a visibilidade e o valor.

Definição/Criação de Variável
private int velocidade;
public int espaco;

Inicializando Variável
velocidade = 60;
espaco = 2;

Usando/Chamando Variável
tempo = espaco / velocidade;

Final de Linha
Toda linha de código em C# termina com ponto e vírgula (;).

Exemplo:
public int number1 = 2;

O C# exige que uma variável seja inicializada (receba valor) antes de ser usada.

Visibilidade/Modificadores de Acesso

public – são variáveis que podem ser acessadas também por outros scripts. Aparecem no Inspector para atribuição e alteração
private – são acessíveis somente no script onde são criadas. Não aparecem no Inspector

l**ocal** – são acessíveis somente dentro da função onde foram criadas. Não aparecem no Inspector
Quando não especificamos a visibilidade é assumida private, que é a default.
No JavaScript, para ser public, basta que seja criada fora de função.

Exemplo:
public int nomeVariavel = 3;
ou
public void Start()

Comentários

// Comentários de linha única

/* Comentários
para várias
linhas
*/

Indentação

Importante para a organização e legibilidade do código

```
void Start(){
        if(x == 0){
                y = 3;
        }
}
```

Editar Script

Edite o script e adicione as variáveis abaixo da linha da definição da Classe. Assim:

```
using UnityEngine;
using System.Collections;

public class VeriaveisScript : MonoBehaviour {
        // Definição das variáveis
        public int velocidade;
        public int distancia;
        public int tempo;

        void Start () {
                // Inicialização
                distancia = 120;
                tempo = 2;

                // Usando
                velocidade = distancia / tempo;
                print (“A velocidade é de “ + velocidade);
        }
}
```

O comando print é mostrado apenas na barra de status e na janela Console.
Selecionar a janela Console e clicar em Play.

```
public class MetodoScript : MonoBehaviour {

	public int arma = 0;

	// Use this for initialization
	void Start () {
		SacarArma ();
	}

	// Update is called once per frame
	void SacarArma () {
		// Armas com codigo: 1-espada, 2-flecha, 3-revolver, 4-rifle
		arma = Random.Range (1, 5);

		switch(arma){
		case 1:
			print ("Voce encontrou a espada");
			break;
		case 2:
			print ("Voce encontrou a flecha");
			break;
		case 3:
			print ("Voce encontrou o revolver");
			break;
		case 4:
			print ("Voce encontrou o rifle");
			break;
		default:
			print ("Voce nao encontrou nada");
			break;
		}
	}
}
```

Execute algumas vezes.

Tipos de Dados
int – inteiros
float – ponto flutuante, são números decimais. Em C# devem ter seus valores escritos com um f ao final, para distinguir dos inteiros. Ex: float speed = 3.2f;
string – são dados do tipo texto
Visibilidade – Níveis de Acesso

Nomes de Variáveis e Métodos
Alguns nomes são proibidos, pois tem seu uso reservado pelo Unity, como é o caso de algumas letras:
x, y, z
Que são usadas internamente pelo transform.

Expressões e Precedência

Precisamos aprender a manipular as expressões em programação, assim como em Matemática. Aprender sobre a ordem de precedência, quem é executado primeiro, quem depois, etc.

Exemplo:
int variavelInt;
float variavelInt2;

variavelInt2 = 3;

variavelInt = 10 + 3 * variavelInt2;

Melhor é fazer assim, usando parêntesis, caso contrário dará um resultado errado:

variavelInt = (10 + 3) * variavelInt2;

Comando Condicional - if

Criar um script chamado IfScript. Remova o script Variaveis da Main Camera e anexe este.
A mais comum forma de tomar decisões é usando o comando if.

Operador NOT
if(!variavel)...

```
using UnityEngine;
using System.Collections;

public class IfScript : MonoBehaviour {
	// Definição das variáveis
	public int velocidade;
	public int distancia;
	public int tempo;

	// Use this for initialization
	void Start () {
		// Inicialização
		distancia = 180;
		tempo = 10;

		velocidade = distancia / tempo;

		if(velocidade > 70){
			print (“Desacelere, pois sua velocidade é de “ + velocidade);
		}
	}
}
```

Execute.

Atribuição
= é usado para atribuir, para gravar um valor em uma variável. Exemplo:

velocidade = 5; // Diz que a variável recebeu 5, agora tem 5 armazenado nela

== usado nas comparações com if, para comparar uma variável com um valor ou com outra variável. Exemplo:

if (x == 3)
ou
if (x == z)

Operadores

Operadores de Comparação

==, >, <, >=, <=, !=

Operadores Lógicos

&& - AND
|| - OR
! - NOT

Operadores Matemáticos

+, -, *, /, %, ++, --

*=, +=, /=, -=, %=

Comando Switch - Instruções switch agem como condicionais if. Elas serão úteis para quando você quiser comparar uma única variável contra uma série de constantes.

Este comando substitui vários ifs em algumas vezes.

Crie um script SwitchScript contendo:

```
using UnityEngine;
using System.Collections;

public class SwitchScript : MonoBehaviour {

	public int arma = 0;

	void Start () {
		// Armas com codigo: 1-espada, 2-flecha, 3-revolver, 4-rifle
		arma = Random.Range (1, 5);

		switch(arma){
		case 1:
			print ("Voce encontrou a espada");
```

```
                break;
            case 2:
                print ("Voce encontrou a flecha");
                break;
            case 3:
                print ("Voce encontrou o revolver");
                break;
            case 4:
                print ("Voce encontrou o rifle");
                break;
            default:
                print ("Voce nao encontrou nada");
                break;
            }
        }
}
```

Execute várias vezes.

Métodos
É onde acontecem as ações. Um método contem um trecho de código, é um pequeno programa dentro do script.
Geralmente o código nos scripts é executado uma linha por vez.
Quando criamos nosso código geralmente queremos executá-lo mais de uma vez e queremos poder executar apenas um trecho de código.

O código dos métodos inicia na próxima linha após o início da chave e termina na linha imediatamente anterior ao final da chave.

Métodos são usados para oferecer comportamento aos GameObjects e também para reutilizar código.

Todo código executável em scripts está dentro de métodos.

Se nosso código executa algumas tarefas várias vezes ele está nos dizendo que devemos criar um método e apenas chamá-lo sempre que quisermos executar o código.

Exemplo: precisamos multiplicar dois números em várias partes do código. Então criamos um pequeno método chamado ProdutoDois() e apenas o chamamos quando precisarmos multiplicar os dois numeros.
Editar o script e adicionar duas variáveis:

```
public int numberOne = 2;
public int numberTwo = 9;
```

Insira no método Update():

```
    if (Input.GetKeyUp(KeyCode.Return)) // Quando teclar Enter
        ProductTwo(); // irá processar o método
```

Adicionar abaixo do método Update():

```
void ProductTwo(){
        Debug.Log(numberOne * numberTwo);
}
```

Um método também quando chamado ele processa todo o código do seu conteúdo. Métodos tem duas chaves, no início uma chave abrindo e ao final fechando a chave.

Os métodos que criamos somente serão executados quando forem chamados. Os métodos do Unity Start e Update são executados automaticamente pelo próprio Unity.

Chamando/executando um Método

```
ProductTwo();
```

Um método pode ter muitas linhas de código em seu interior. Um método é chamado um pequeno programa dentro de outro.

Tipos de Métodos:

Simples, apenas executando um trecho de código:

```
void ProductTwo(){
        Debug.Log(numberOne * numberTwo);
}
```

Método que recebe parâmetros – recebe variáveis e processa em seu conteúdo

```
void ProductTwo(int x, int y){
        Debug.Log(x * y);
}
```

Método que Retorna valores – pode receber variáveis e retorna o valor de uma variável de certo tipo

```
public int ProductTwo(int x, int y){
        int produto = (x * y);
        return produto;
}
```

Nomes de Métodos
-visibilidade
-tipo de dados que o método retornará
-nome do método, CamelCase e seguido de ()
-código delimitado por chaves { }

```
visibilidade tipoDeRetorno NomeMetodo(){
        corpo do método;
}
```

void – usado quando o método não retorna nada.

Parâmetros – usados na definição da função: firstNumber e secondNumber.

Argumentos – usados na chamada da função: 3 e 4.

O comando return deve ser o último comando da função.

Estruturas de Laço

Laço for
Crie um script chamado ForScript contendo:

```
using UnityEngine;
using System.Collections;

public class ForScript : MonoBehaviour {

	void Start () {
		BuscaInimigo ();
	}

	// Update is called once per frame
	void BuscaInimigo () {
		for (int i=0; i < 5; i++) {
			print ("Valor de i " + i);
		}
	}
}
```

foreach – laço que mostra os elementos de um array

```
using UnityEngine;
using System.Collections;

public class ForeachLoop : MonoBehaviour {
    void Start ()  {
        string[] strings = new string[3];

        strings[0] = "First string";
        strings[1] = "Second string";
        strings[2] = "Third string";

        foreach(string item in strings) {
            print (item);
        }
    }
}
```

Execute e veja o resultado.

Arrays/Matrizes

Um array é tipo uma variável e é uma forma de uma variável conter mais de um tipo de dados.

Criar 3 empty GameObjects chamados:
gato, cachorro e porco

Criar a tag Inimigo
Selecionar os 3 e adicionar a tag a eles

Exemplo:

```
public GameObject[ ] players;
```

```
using UnityEngine;
using System.Collections;

public class ArrayScript : MonoBehaviour
{
	public GameObject[] players;

	void Start (){
		players = GameObject.FindGameObjectsWithTag("Inimigo");

		for(int i = 0; i < players.Length; i++){
			Debug.Log("Inimigo numero "+i+" tem o nome "+players[i].name);
		}
	}
}
```

Classes
Classes contém variáveis e métodos e processam seus dados.
Em Unity praticamente não lidamos com a classe pois ele já cria para nós, mas todo script já vem com uma classe, cujo nome é o mesmo do script.

Como usar classes para armazenar e organizar suas informações e como criar construtores para trabalhar com partes de sua classe.

Exemplo:

```
using UnityEngine;
using System.Collections;

public class Jogador : MonoBehaviour {
   public class Armas {
      public int balas;
      public int granadas;
      public int raquetes;

      public Armas(int bal, int gra, int raq){
         balas = bal;
         granadas = gra;
```

```
            raquetes = raq;
        }
    }

    // Abaixo temos a variável minhasArmas do tipo Armas
    public Armas minhasArmas = new Armas(10, 7, 25);

    void Update (){
        Movimento();
        Atirar();
    }

    void Movimento(){
            Debug.Log (“Movimento”)
    }

    void Atirar(){
            Debug.Log (“Atirar”)
    }
}
```

Outro exemplo

```
using UnityEngine;
using System.Collections;

public class VariaveiseFuncoes : MonoBehaviour
{
    int meuInt = 5;

    void Start ()
    {
        meuInt = MultiplicarPorDois(meuInt);
        Debug.Log (meuInt);
    }

    int MultiplicarPorDois (int numero)
    {
        int ret;
        ret = numero * 2;
        return ret;
    }
}
```

Componente Script
Lembrar que um script é um componente no Unity que deve ser anexado a um GameObject para funcionar. Ao anexar o nome do script aparece ao final das propriedades do Objeto no Inspector. Também aparecem as variáveis public.

Os scripts é que dão interatividade e vida aos jogos.

Variáveis armazenam dados.
Métodos permitem a execução de tarefas.

Sintaxe dos pontos

transform.position.x;

Alterações das Propriedades
Podemos alterar as propriedades do Inspector tanto na cena quanto quando o jogo está executando. Alterações quando executando aparecem imediatamente mas se perdem logo que o jogo é parado.

O valor entrado nas variáveis no Inspector sobrescreve o valor original do script.

Resetando Mudanças
Para desfazer mudanças feitas nas variáveis de um objeto, melhor de um componente, basta clicar com o botão direito na pequena roldana à direita do nome do componente e clicar em Reset.

Nomes de Variáveis
Crie nomes simples e claros para suas variáveis.

Lembre da convenção:
-quando formado por uma única palavra, ele deve ser toda em minúscula:

private int speed;

-quando palavra composta, a primeira é com tudo minúsculo e a segunda iniciada com maiúscula, camelCase:

private int speedPlayer;

Nomes de Métodos
Seguem as mesmas regras, apenas com () ao final, mas assim CamelCase, diferente das variáveis devem iniciar com maiúscula como Start().

Nomes de Classes
Estes também devem ser CamelCase mas com inicial maiúscula.

Onde Declarar Variáveis
O local do código onde a variável é declarada é importante em termos de visibilidade. Variáveis declaradas no início do script, logo abaixo da declaração da classe

Escopo das Variáveis

```
using UnityEngine;
using System.Collections;

public class LearningScript : MonoBehaviour {// --------Bloco 1
	public string block1 = "Bloco 1";
	// Use this for initialization
	void Start () { //  -------- Bloco 2
		Debug.Log (block1);

		string block2 = "Bloco 2";
		Debug.Log (block2);
		{// --------- Bloco 3
			Debug.Log (block1);
			Debug.Log (block2);

			string block3 = "Bloco 3";
		} // Final Bloco 3
	} // Final Bloco 2

	void Metodo(){// --- Bloco 4
		string block4 = "Bloco 4";

		Debug.Log (block4);
	}// Final Bloco 4

} // Final Bloco 1
```

Bloco 3 – criado apenas com as chaves.
Tente mostrar a variével block3 no segundo bloco ou a block4 no bloco 3 e veja que o editor já reclama.

O escopo é determinado pela chaves dos blocos. As chaves limitam um bloco de código.

Variável é o primeiro bloco de código em C#.
Método é o segundo.

Editor
Quando tentamos trabalhar com uma variável ou método que ainda não foi declarado o editor já nos alerta mudando a cor do mesmo para vermelho. Logo que corrigimos fica preto.

Sintaxe do Ponto

GameObject.Component
GameObject.Component.variavel ou metodo

Funções de Inicialização
As funções que são executadas automaticamente pelo Unity, sem interferência do programador.

Método Start()

Chamado pelo Unity somente uma vez a cada execução do jogo. Primariamente usado para inicializar ou configurar as variáveis.

Método Update()

A maioria do código dos scripts vai dentro deste método. A cena é mostrada várias vezes por segundo, que é chamado de Frames por Segundo – FPS. Após a exibição de cada frame, o método Update() é chamado pelo Unity para executar seu código.
Permite ao jogo detectar entradas (inputs), cliques do mouse, teclas pressionadas, a cada frame.
O método Update() é adequado para uso de movimentação através de teclas.

awake() e start()

```
using UnityEngine;
using System.Collections;

public class AwakeAndStart : MonoBehaviour {
    void Awake (){
        Debug.Log("Awake called.");
    }

    void Start (){
        Debug.Log("Start called.");
    }
}
```

Update() e FixedUpdate()

Como efetuar mudanças a cada frame/quadro com a atualização usando as funções Update e FixedUpdate e suas diferenças.

```
using UnityEngine;
using System.Collections;

public class UpdateAndFixedUpdate : MonoBehaviour {
    void FixedUpdate (){
        Debug.Log("FixedUpdate time :" + Time.deltaTime);
    }

    void Update (){
        Debug.Log("Update time :" + Time.deltaTime);
    }
}
```

Mostrando os métodos de Inicialização em Ação

```
Debug.Log("ExecutionOrder::global");

void Awake() {
   Debug.Log("ExecutionOrder::Awake");
}
void Start () {
   Debug.Log("ExecutionOrder::Start");
}
void Update () {
   Debug.Log("ExecutionOrder::Update");
}
void FixedUpdate() {
   Debug.Log("ExecutionOrder::FixedUpdate");
}
void LateUpdate() {
   Debug.Log("ExecutionOrder::LateUpdate");
}
void OnRenderObject() {
   Debug.Log("ExecutionOrder::OnRenderObject");
}
```

Objetos Internos do Unity

Um script unity é escrito em uma linguagem de programação, com seus recursos básicos e também conta com classes e funções/métodos da biblioeca do Unity. Um script em C# tem diferenças básicas de um script em JavaScript mas tem em comum os recursos da biblioteca do Unity que ambas utilizam.

transform

```
transform.Translate(1, 1, 1);
transform.Translate(Vector3.forward * Time.deltaTime);
transform.Translate(Vector3.up * Time.deltaTime, Space.World);
```

Vector2

Vector2(float x, float y);

Instantiate

var spawnPoint = new Vector2(Random.Range(x1, x2), transform.position.y);
Instantiate(asteroidePrefab, spawnPoint, Quaternion.identity);

Quaternion

Quaternion(float x, float y, float z, float w);

Ver a referência completa dos objetos do Unity aqui:
http://docs.unity3d.com/ScriptReference/
Quando acessamos esta página acima, à direita e acima aparecem as 3 linguagens para que selecionemos a que queremos perquisar.

Quaternions - É a posição e o ângulo de um objeto a ser rotacionado. Para realizar a rotação de objetos no espaço tridimensional são utilizados quaterniões.
Para representar a direção que um objeto aponta no espaço não basta apenas representar sua posição, é necessário atribuir um ângulo a ela. Em jogos quando um objeto é criado, é definida a sua posição e um ângulo inicial a ele. Quando desejamos fazer alguma rotação nesse objeto, realizamos com base em sua posição e seu ângulo atual, ou seja, em seu quaternião.

Destroy
Destroy(gameObject, 20); // Destroi objeto após 20 segundos

Exemplos de Scripts para Jogos

Movimento

Tiro

Limitando na Tela

Criando um Labirinto via Código

```
using UnityEngine;
using System.Collections;

public class Grade : MonoBehaviour {
	// Referência - http://docs.unity3d.com/Manual/InstantiatingPrefabs.html
	public GameObject cubo;
	private int x,y;

	void Start() {
		LabirintoX(-9, 9, -5, -4);
		LabirintoX(-9, 9, 0, 4);
		LabirintoY(-4, 4, 0, 9);
		LabirintoY(-4, 4, 0, -9);
		LabirintoY(-4, 3, 0, -6);
		LabirintoX(-5, 3, 0, 3);
		LabirintoX(-5, 3, 0, 2);
		LabirintoX(1, 5, 0, -3);
		LabirintoX(1, 5, 0, -2);
		LabirintoX(1, 5, 0, -1);
		LabirintoX(1, 5, 0, 0);
		LabirintoX(-4, 3, 0, -3);
		LabirintoX(-4, -1, 0, -2);
		LabirintoX(-4, -1, 0, -1);
		LabirintoX(-4, -1, 0, 0);
		//LabirintoX(-4, 0, 0, -1);
		/*
		y = -4;
		for(x=-9;x <=9;x++){
			if(x == -6) continue;
			Instantiate(cubo, new Vector3(x, y, 0), Quaternion.identity);
```

```
            }*/
        }

        void LabirintoX(int x1, int x2, int pulo, int y){
            for(x=x1;x <=x2;x++){
                if(pulo != 0 && x == pulo) continue;
                Instantiate(cubo, new Vector3(x, y, 0), Quaternion.identity);
            }
        }

        void LabirintoY(int y1, int y2, int pulo, int x){
            for(y=y1;y <=y2;y++){
                if(pulo != 0 && y == pulo) continue;
                Instantiate(cubo, new Vector3(x, y, 0), Quaternion.identity);
            }
        }
}
```

Corrigindo Erros/Debugando

Editor Monodevelop

O editor Monodevelop ajuda muito na correção de erros. Alguns erros ele acusa logo no script. Tão logo digitemos o nome de uma variável para usá-la e antes de a definir, ele nós alerta deixando o nome em vermelho. Isso significa que ele exige que toda variável só pode ser usada quando definida/criada antes.

Ele também nos ajuda auto-completando a sintaxe de métodos e objetos internos do Unity.

Unity

Quando salvamos um script e voltamos para o Unity geralmente, havendo erro no script, o Unity avisa do erro com mais detalhes que o editor. Ele mostra o erro, o script e a linha, além de outras informações úteis e isso em vermelho na barra de status.

Algumas vezes ele não mostra detalhes sobre o erro, apenas a mensagem. Quando isso acontecer devemos acessar a Console e clicar na mensagem para receber mais detalhes.

Ferramentas

A console ajuda muito na correção de erros.
Podemos usar o comando Debug.Log para mostrar uma mensagem em uma região do script que desconfiamos de erro, ou mostrar o valor de uma variável.
Exemplos:
Debug.Log(“Teste”);

Debug.Log(“Velocidade vale: ” + speed);

Variáveis public

Quando declaramos uma variável public e do tipo GameObject no script, precisamos atribuir um prefab ou objeto a ela pelo Inspector do Unity. Quando esquecemos de fazer isso recebemos a mensagem: **Null Reference Exception**