

Programação Básica em C# para Unity - MonoBehaviours

Bruno dos Santos de Araújo, MSc

1 de setembro de 2019

Introdução

Game loop

Estrutura de um MonoBehaviour

Introdução

- Este treinamento tem os seguintes objetivos:
 - Entender o conceito de game loop e como se aplica ao Unity;
 - Mostrar a estrutura básica do tipo de script mais comum no Unity: `MonoBehaviour`;
 - Implementar uma versão simples porém funcional do jogo Breakout (também conhecido como Arkanoid).
- Existem diversas abordagens possíveis para os problemas que iremos resolver, neste treinamento utilizaremos as abordagens mais simples e didáticas e não necessariamente as mais eficientes.

Game loop

- Quase todos os jogos utilizam um algoritmo chamado `game loop`, que basicamente possui 3 funções:
 - Ler e processar a entrada do usuário
 - Atualizar o estado interno do jogo
 - Desenhar a tela
- O número de vezes que esse algoritmo é executado por segundo determina a taxa de **quadros por segundo** do jogo, ou **FPS** (frames per second)
 - 30 FPS: a cada 33ms
 - 60 FPS: a cada 16ms

- Visualização do algoritmo:

```
while(gameIsRunnnng) {  
    readInput();  
    update();  
    render();  
}
```

- readInput(): lê a entrada do usuário
- update(): atualiza o estado interno do jogo
- render(): desenha a tela

- Ao implementar código de gameplay no Unity, precisamos sempre implementar os equivalentes a `readInput()` e `update()` dentro de uma subclasse de `MonoBehaviour`.
 - `render()` é implementada via shaders; mesmo que você não escreva nenhum, o Unity utiliza shaders padrão
 - Boa referência sobre o assunto: <https://gameprogrammingpatterns.com/game-loop.html>

Estrutura de um MonoBehaviour

Estrutura de um MonoBehaviour

- MonoBehaviour é a classe da qual se deriva a maioria dos componentes dentro do Unity.
- Ao criarmos um novo MonoBehaviour, seja diretamente como componente num GameObject ou na hierarquia, nos deparamos com o seguinte código:

```
public class NewBehaviourScript : MonoBehaviour
{
    void Start()
    {
    }

    void Update()
    {
    }
}
```

- Podemos observar duas funções que foram criadas por padrão: Start() e Update().

Estrutura de um MonoBehaviour

- `Start()`: esta função é chamada uma vez assim que o script é ativado pela primeira vez, sempre antes da primeira chamada a `Update()`.
 - Normalmente utilizada para inicializar quaisquer variáveis e chamar funções de inicialização.
 - Devido a ser executado só na ativação do script, pode ser utilizada para atrasar a inicialização para o momento que seja mais conveniente.

```
private void Start()
{
    maxHp = 10;
    transform.position = GetInitialPosition();
}
```

Estrutura de um MonoBehaviour

- `Update()`: esta função é chamada a cada quadro quando o script está ativado, e é utilizada para atualizar o estado interno do componente.
 - No caso de ser um objeto controlável pelo jogador, também é onde se leem as entradas (teclas, botões, etc);
 - O tempo entre chamadas é aproximadamente o mesmo e pode variar de acordo com a quantidade de objetos na cena.

```
private void Update()  
{  
    // Update position at each frame  
    transform.position += velocity * Time.deltaTime;  
}
```

- Outras funções comumente utilizadas seguem nos próximos slides.

Estrutura de um MonoBehaviour

- `Awake()`: é chamada uma única vez no ato da criação do `GameObject`, sempre antes de `Start()`;
 - Normalmente utilizada para obter referências a componentes locais do `GameObject` e para instruções que precisam ser executadas sempre antes de `Start()`.
 - Se o objeto não estiver ativado na inicialização da cena, será chamada na primeira vez que o objeto for ativado.
 - Se o objeto estiver ativo mas não o script, `Awake()` será chamada.

```
private void Awake()  
{  
    rigidBody = GetComponent<Rigidbody>();  
    animator = GetComponent<Animator>();  
}
```

Estrutura de um MonoBehaviour

- `FixedUpdate()`: semelhante a `Update()` mas é chamada em tempos fixos a cada passo da Física, que por padrão é 20ms;
 - Utilizada principalmente quando se utilizam funções que agem na física do `GameObject`.

```
private void FixedUpdate()  
{  
    if(Input.GetKey(KeyCode.RightArrow))  
    {  
        rigidbody.AddForce(Vector3.right * forceAmount);  
    }  
}
```

Estrutura de um MonoBehaviour

- `OnEnable()` e `OnDisable()`: chamadas na ativação e desativação do `GameObject`, respectivamente;
- Permitem separar a inicialização de maneira mais granular, e também executar algum código na desativação do objeto.

```
// assume the game has a minimap showing the enemies
private void OnEnable()
{
    AddToMinimap();
}

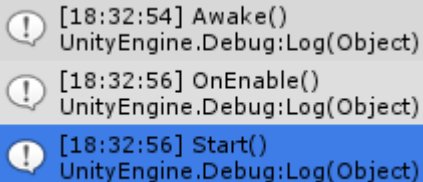
private void OnDisable()
{
    RemoveFromMinimap();
}
```

- `OnDestroy()`: chamada quando o objeto é destruído, seja quando a cena é descarregada ou usando `Destroy()` diretamente.
 - Menos comumente utilizada, mas útil pra quaisquer tarefas de limpeza quando o objeto é destruído.

```
private void Destroy()  
{  
    RemoveFromEnemyList();  
}
```


Exemplo da ordem de execução dos scripts

```
private void Awake() {  
    Debug.Log("Awake()");  
}  
  
private void Start() {  
    Debug.Log("Start()");  
}  
  
private void OnEnable() {  
    Debug.Log("OnEnable()");  
}
```



The screenshot shows the Unity console with three log entries, each preceded by a warning icon (a speech bubble with an exclamation mark). The first entry is "[18:32:54] Awake()" followed by "UnityEngine.Debug:Log(Object)". The second entry is "[18:32:56] OnEnable()" followed by "UnityEngine.Debug:Log(Object)". The third entry, "[18:32:56] Start()", followed by "UnityEngine.Debug:Log(Object)", is highlighted with a blue background, indicating it is the current log entry.

[18:32:54] Awake()
UnityEngine.Debug:Log(Object)
[18:32:56] OnEnable()
UnityEngine.Debug:Log(Object)
[18:32:56] Start()
UnityEngine.Debug:Log(Object)