```
using System.Text;
using System.Net;
using System.Net.Sockets;
using static System.Console;
using Unity.GameObject;

private class MyVeryOwnPrivateClass
{
  public static void Main()
  {
    const string textToSend = "potato";
    const string localhost = "127.0.0.1";
    const int port = 80;

    var data = Encoding.UTF8.GetBytes(textToSend);
    var ip = IPAddress.Parse(localhost);
    var ipEndPoint = new IPEndPoint(ip, port);

    var mainCharacter = new GameObject(loadAsset("yomama.3ds");

    using(var socket = new Socket(AddressFamily.InterNetwork,
    SocketType.Stream, ProtocolType.Tcp))
    {
      socket.Connect(ipEndPoint);
      var byteCount = socket.Send(data, SocketFlags.None);
      WriteLine("Epic Win", byteCount);
      var buffer = new byte 256];
      byteCount = socket.Receive(buffer, SocketFlags.None);

      if(byteCount > 0)
      {
          WriteLine("This is pain!", byteCount);
          var answer = Encoding.UTF8.GetString(buffer);
          WriteLine("Get rect...", answer);
      }
    }
    return 0;
  }
}
```

Tübingen - Codevember 2015

C# Einführung für Unity (wir brauchen kein .NET)

Ja, wie war das noch mal mit diesem objektorientierten Programmieren...?

Hier soll jetzt nicht noch mal OOP komplett aufgeführt werden. Allerdings werdet ihr vielleicht ein paar Hinweise für den Anfang brauchen wenn ihr noch nie mit Unity gearbeitet habt. Deshalb folgen auf den nächsten Folien für die Programmierer ein paar wenige Basics, damit der Einstieg schnell läuft. Auch hier gilt ‚Probieren geht über studieren' - also testet doch gleich ein paar Zeilen die ihr hier bekommt!

01 - Einbinden der Standard-Libs

In Unity bindet man standardmäßig zwei Basisbibliotheken ein:

```
using UnityEngine;
using System.Collections;
```

Damit haben wir den größten Teil aus Unity‘s Klassen und
den C#-Standard Bibliotheken abgedeckt. Falls also jemand bereits
C#-Kenntnisse hat, wird er sein Wissen auch in Unity anwenden können.

02 - Klassen und vordefinierte Funktionen

Sobald ihr ein Script erstellt, achtet auf den Namen der Klasse. Dieser muss nämlich identisch mit dem Dateinamen sein und die Klasse MonoBehaviour erweitern:

```
// Klassenname = Dateiname
public class Klassenname : MonoBehaviour
{
    // TODO:
    void EmptyFunction() { return; }
}
```

In Unity gibt es einige abstrakte Methoden, welche sehr wichtig sind, da sie automatisch zu bestimmten Zeiten oder Ergeignissen aufgerufen werden:

```
void Start()
void Update()
void LateUpdate()
void OnMouseDown()
```

02 - Vordefinierte Funktionen (Fortsetzung I)

```
void Start()
{
    // Funktionsinhalt wird einmal aus-
    // geführt, z.B. wenn das Objekt
    // gerade erstellt wurde.
}
```

```
void Update()
{
    // Funktionsinhalt wird in jedem
    // neuen Frame ausgeführt. Achtung:
    // Schlechter Code kann die Bild-
    // wiederholungsrate massiv senken!
}
```

02 - Vordefinierte Funktionen (Fortsetzung II)

```
void LateUpdate()
{
    // Funktionsinhalt wird in jedem Frame
    // ausgeführt, allerdings nachdem er
    // gerendert wurde.
}
```

```
void OnMouseDown()
{
    // In Verbindung mit z.B einem Collider
    // wird der Funktionsinhalt ausgeführt,
    // sobald ein Mausklick auf dem Objekt
    // bzw. dessen Collider ausgeführt
    // wurde.
}
```

03 - Tastaturinput abfangen

Beispiel Code zur Bestimmung von Tastatureingaben:
(eine Liste mit den verwendbaren Keycodes findet sich in der Doku)

```
void Update () {
        bool down = Input.GetKeyDown(KeyCode.Space);
        bool held = Input.GetKey(KeyCode.Space);
        bool up = Input.GetKeyUp(KeyCode.Space);

        if(down) {
            graphic.texture = downgfx;
        }
        else if(held) {
            graphic.texture = heldgfx;
        }
        else if(up) {
            graphic.texture = upgfx;
        }
        else {
            graphic.texture = standard;
        }
    }
```

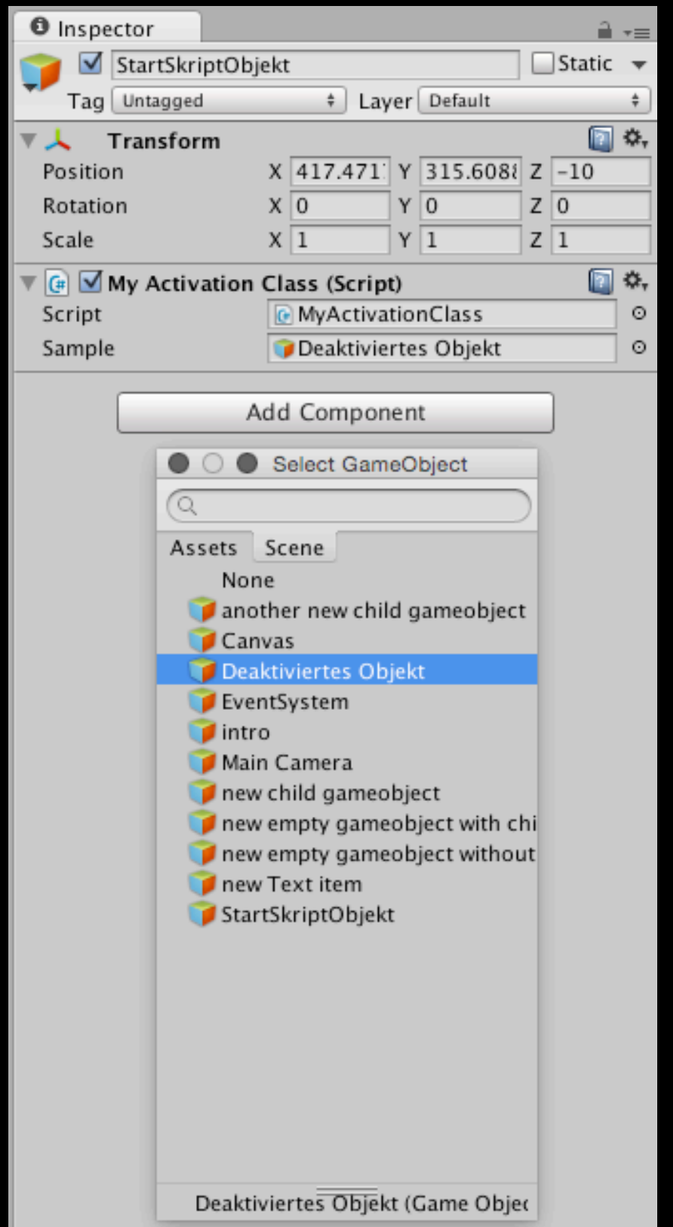
04 - Attribute in der Klasse deklarieren

Ohne Attribute in der Klasse fest zu definieren, reicht es aus sie zu deklarieren und danach im Inspector aus bereits vorhandenen Objekten zu setzen, falls das Script einem GameObject als Komponente hinzugefügt wurde.

```
using UnityEngine;
using System.Collections;

// Dieses Script aktiviert beim Start
// ein anderes Gameobject.
public class MyActivationClass : MonoBehaviour
{
    public GameObject sample;

    void Start()
    {
        sample.SetActive(true);
    }
}
```



05 - Ich bin ein Gameobject, wie spreche ich mit mir selbst?

C++ oder Java Liebhaber kennen es...

Ich habe ein Script geschrieben und es an ein Gameobject als Komponente gehängt. Oft passiert es aber, das „this“ nicht funktionieren wird um auf das eigene Objekt zu verweisen. Dazu verwendet man das Schlüsselwort „gameObject“ (Case-Sensitive!).

Möchte man etwa, dass ein Gameobject sich selbst deaktiviert könnte man das hier benutzen:

```
gameObject.SetActive(false);
```