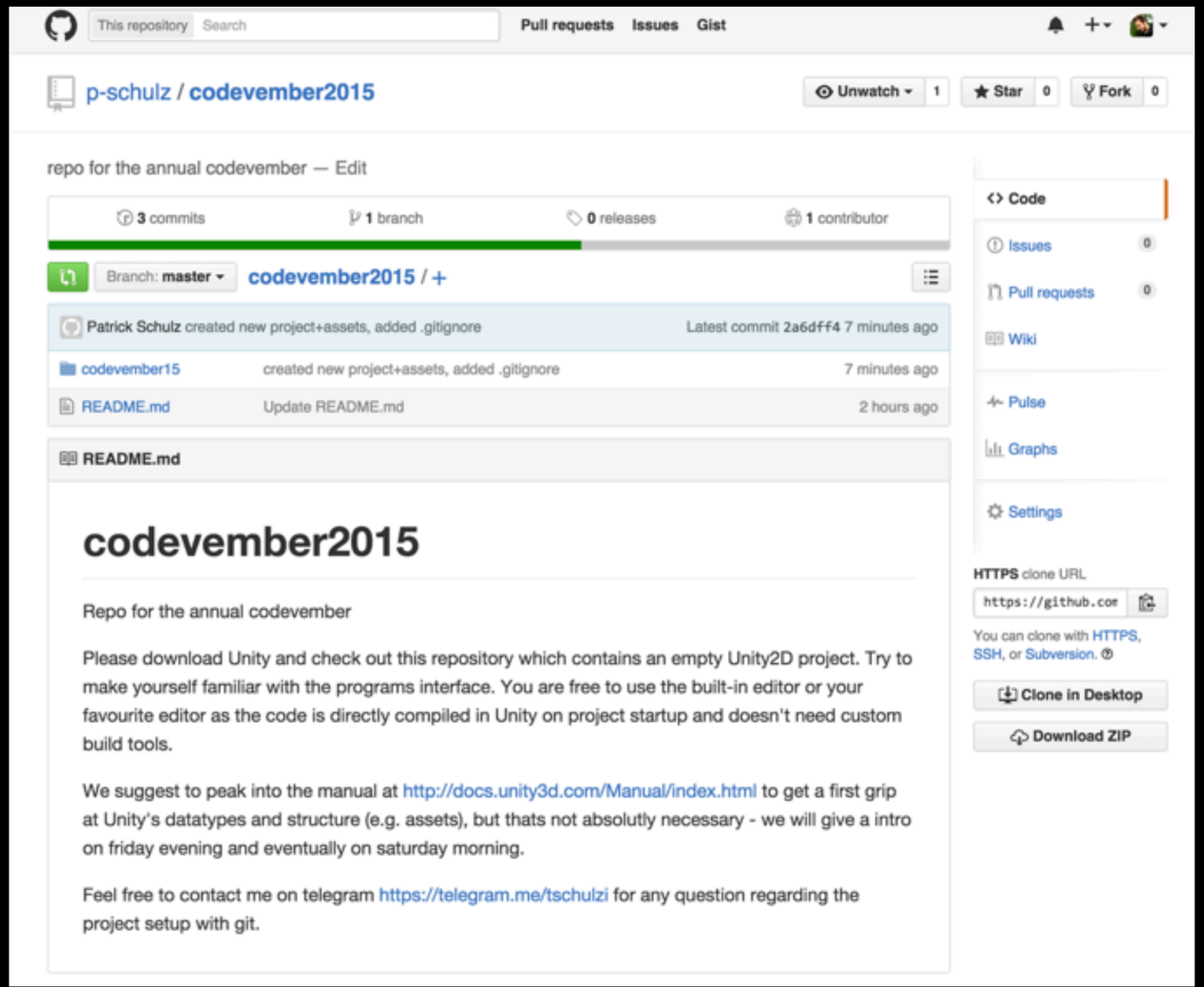
Eine kurze Einführung in Einrichtung, Oberfläche und Funktionen

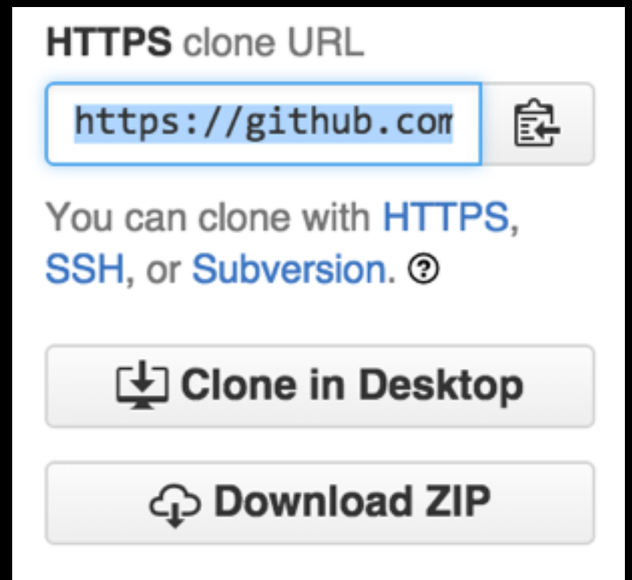
Tübingen - Codevember 2015

01 - Einrichtung und erster Start



Erstmaliger Clone vom Projekt, Beispiel mit HTTPS und clone URL:

Rechts unten den Link kopieren und eure Github Accountinformationen bereithalten...



Das Repo ist verfügbar auf: https://github.com/p-schulz/codevember2015

01 - Einrichtung und erster Start

```
Schulz@Macbook911:~/Documents$ mkdir codevember2015
Schulz@Macbook911:~/Documents$ cd codevember2015/
Schulz@Macbook911:~/Documents/codevember2015$ git clone https://github.com/p-schulz/codev
Schulz@Macbook911:~/Documents/codevember2015$ ll
total 16
drwxr-xr-x   4 Schulz  staff   136B 16 Nov 12:36 .
drwx------+ 31 Schulz  staff   1,0K 16 Nov 13:28 ..
-rw-r--r--@  1 Schulz  staff   6,0K 16 Nov 13:27 .DS_Store
drwxr-xr-x   7 Schulz  staff   238B 16 Nov 13:20 codevember2015
Schulz@Macbook911:~/Documents/codevember2015/codevember2015$ ll
total 32
drwxr-xr-x   7 Schulz  staff   238B 16 Nov 13:20 .
drwxr-xr-x   4 Schulz  staff   136B 16 Nov 12:36 ..
-rw-r--r--@  1 Schulz  staff   6,0K 16 Nov 12:36 .DS_Store
drwxr-xr-x  15 Schulz  staff   510B 16 Nov 13:21 .git
-rw-r--r--   1 Schulz  staff   116B 16 Nov 13:20 .gitignore
-rw-r--r--   1 Schulz  staff   748B 16 Nov 11:42 README.md
drwxr-xr-x  10 Schulz  staff   340B 16 Nov 13:26 codevember15_
```

Ordner erstellen und das Projekt mit Git auf den lokalen Rechner klonen.

01 - Einrichtung und erster Start

```
Schulz@Macbook911:~/Documents/codevember2015/codevember2015/codevember15$ ll
total 128
drwxr-xr-x  10 Schulz  staff   340B 16 Nov 13:26 .
drwxr-xr-x   7 Schulz  staff   238B 16 Nov 13:20 ..
-rw-r--r--@  1 Schulz  staff   8,0K 16 Nov 13:12 .DS_Store
-rw-r--r--   1 Schulz  staff    17K 16 Nov 13:12 Assembly-CSharp-Editor.csproj
-rw-r--r--   1 Schulz  staff    21K 16 Nov 13:12 Assembly-CSharp-firstpass.csproj
drwxr-xr-x   9 Schulz  staff   306B 16 Nov 13:14 Assets
drwxr-xr-x  23 Schulz  staff   782B 16 Nov 13:26 Library
drwxr-xr-x  19 Schulz  staff   646B 16 Nov 13:26 ProjectSettings
-rw-r--r--   1 Schulz  staff   1,9K 16 Nov 13:26 codevember15.sln
-rw-r--r--   1 Schulz  staff   568B 16 Nov 13:10 codevember15.userprefs
```

- Struktur des Projektverzeichnis -

Bitte beachtet, dass der Library-Ordner nie commited und gepushed werden sollte. Auf der nächsten Folie wird darauf eingegangen wie das mithilfe der .gitignore Datei verhindert wird.
Nach Importieren von neuen Inhalten (Bilder, Skripte etc.) sind diese im Assets-Ordner verfügbar (wird auch noch später erläutert).

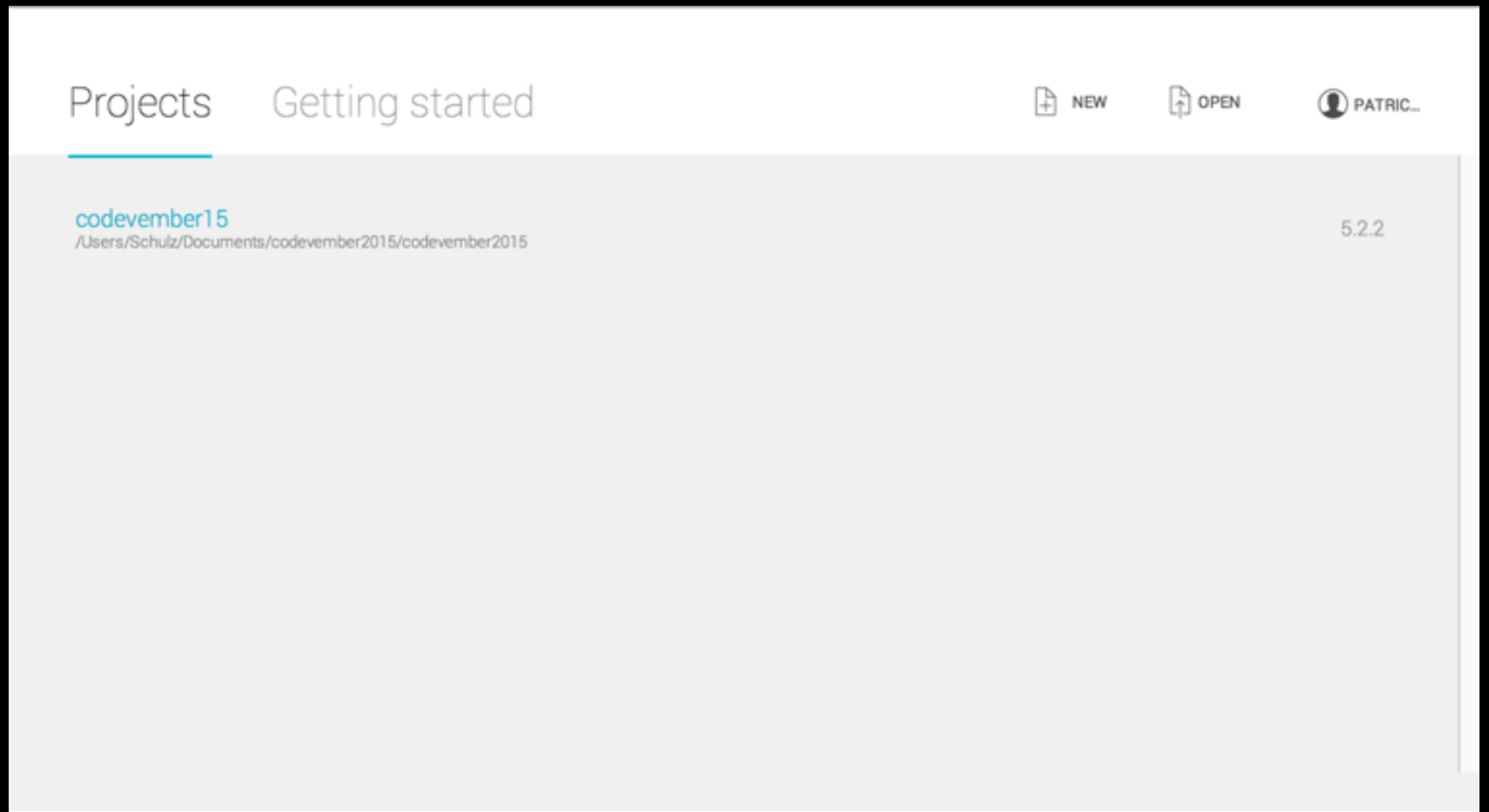
01 - Einrichtung und erster Start

```
codevember15/Temp/
codevember15/Library/
codevember/Obj/

*.csproj
*.unityproj
*.sln
*.user
*.userprefs
*.DS_store
```

Die Datei .gitignore sollte unbedingt in das Git-Wurzelverzeichnis, da sonst beim Pushen und Pullen unnötige Bibliotheks- und Metafiles transferiert werden.
Falls nicht vorhanden selbst erstellen und obigen Text einfügen. Das Projekt selbst wurde mit der Option erstellt, die Metafiles nur lokal zu speichern und für Versionsverwaltung nicht zwingend notwendig im Projektordner zu behalten.

01 - Einrichtung und erster Start

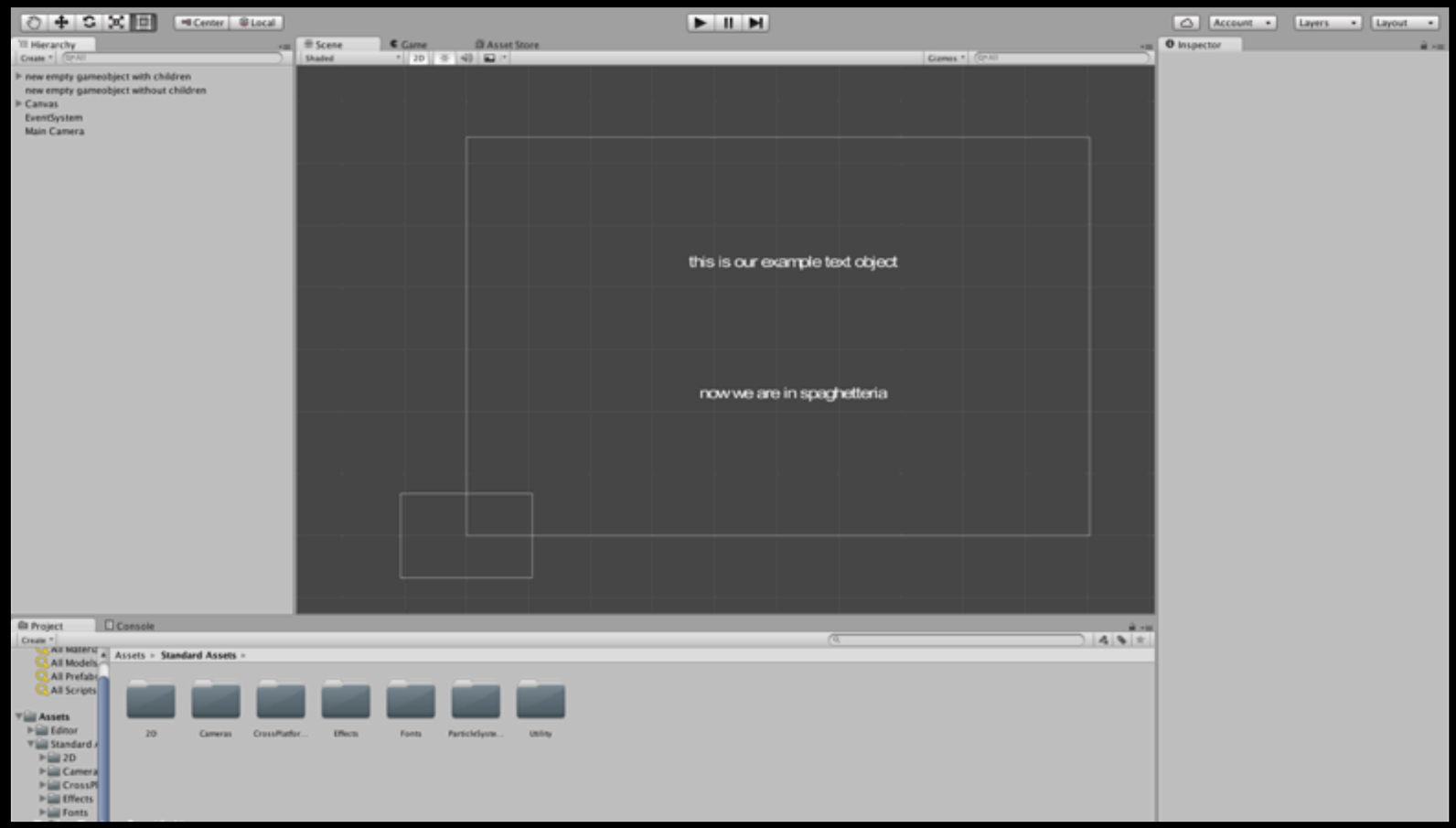


← Erster Programmstart

Die nächsten Folie beziehen sich auf die Benutzung der „Indie-Version“. Sollte aber auch kein Problem sein, da wahrscheinlich niemand Geld für Unity ausgeben wird ;-)

Nach Installation und Login oben auf „OPEN“ klicken und den Ordner „codevember15“ im Git-Verzeichnis auswählen. Das Projekt sollte nun in der Liste stehen und auch beim nächsten Start auswählbar sein.

01 - Einrichtung und erster Start

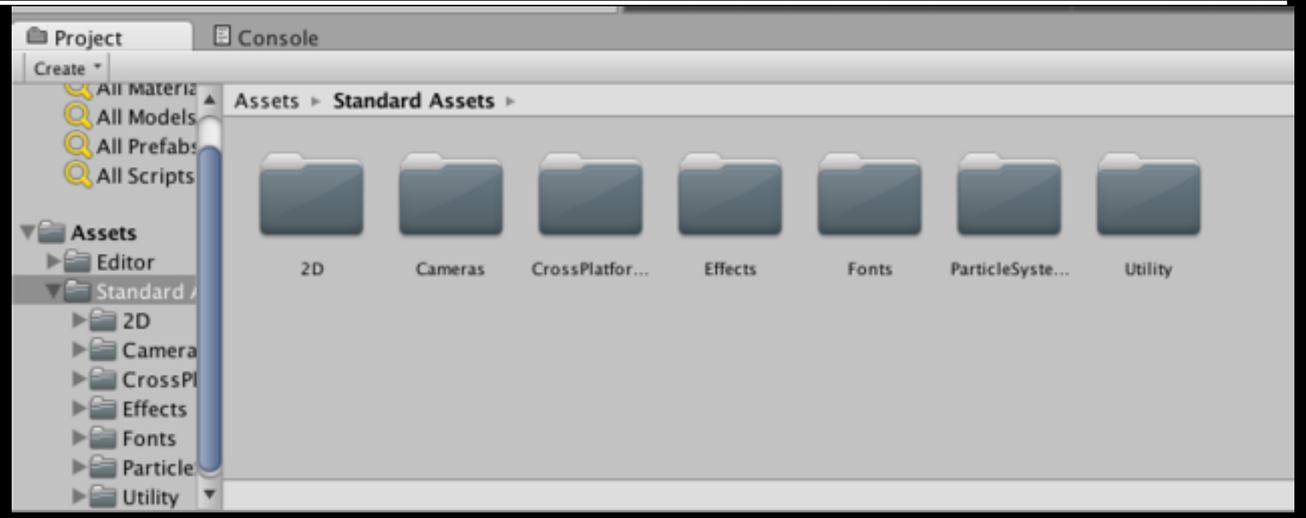


<u>Ansicht nach Programmstart:</u>
Das große Mittelfeld enthält für uns nur zwei wichtige Tabs - Scene und Game. Die Scene ist unsere Standardansicht in der wir die aktuelle Szene gestalten können. Unter Game können wir uns eine Live Version ansehen und einen Eindruck bekommen, was den Anwender wirklich erwartet.

Daneben existiert links die Projekthierarchie, in der alle Objekte der Szene aufgelistet sind. Wird ein Objekt angewählt, so erscheinen rechts im Inspector alle Eigenschaften und Parameter dieses Objekts. Zuletzt sehen wir unten unsere gesamte Ordnerstruktur und den Tab für die Konsole.

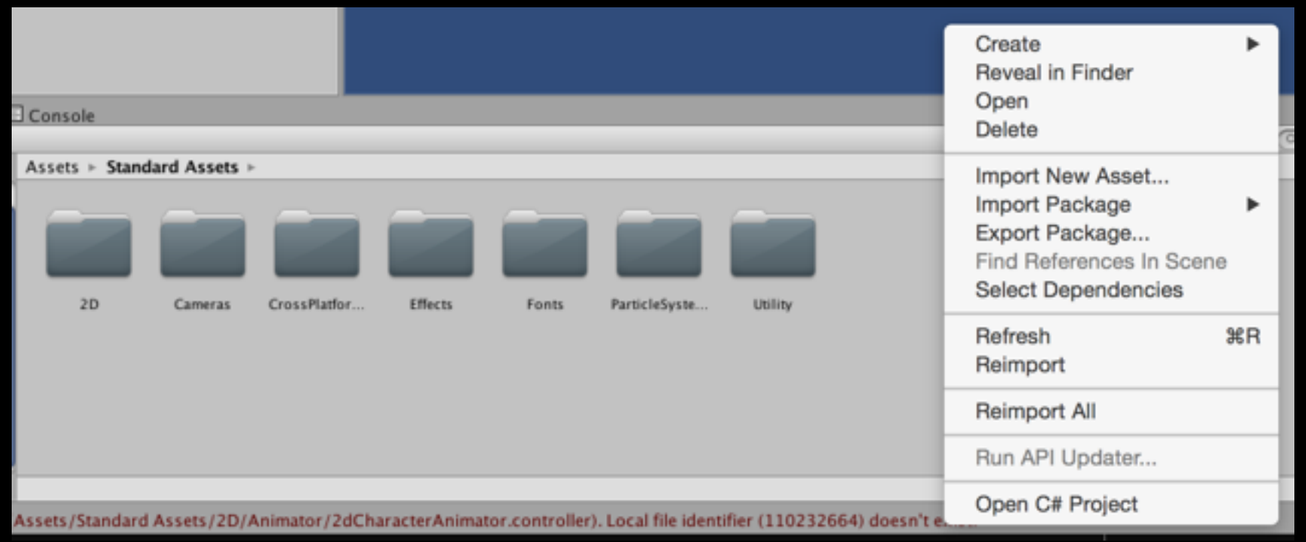
02 - Die Programmoberfläche: Projektstruktur

Im Projektverzeichnis finden wir alle Dateien die in unserem Projekt zur Verfügung stehen.



Rechte Maustaste

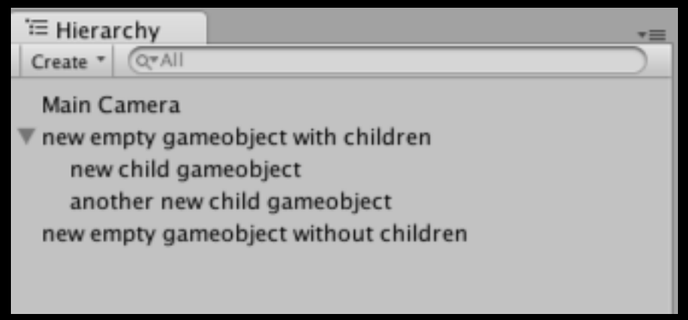
Über einen Klick auf die rechte Maustaste in einem Ordner können neue Objekte oder Assets Erstellt oder Importiert werden.

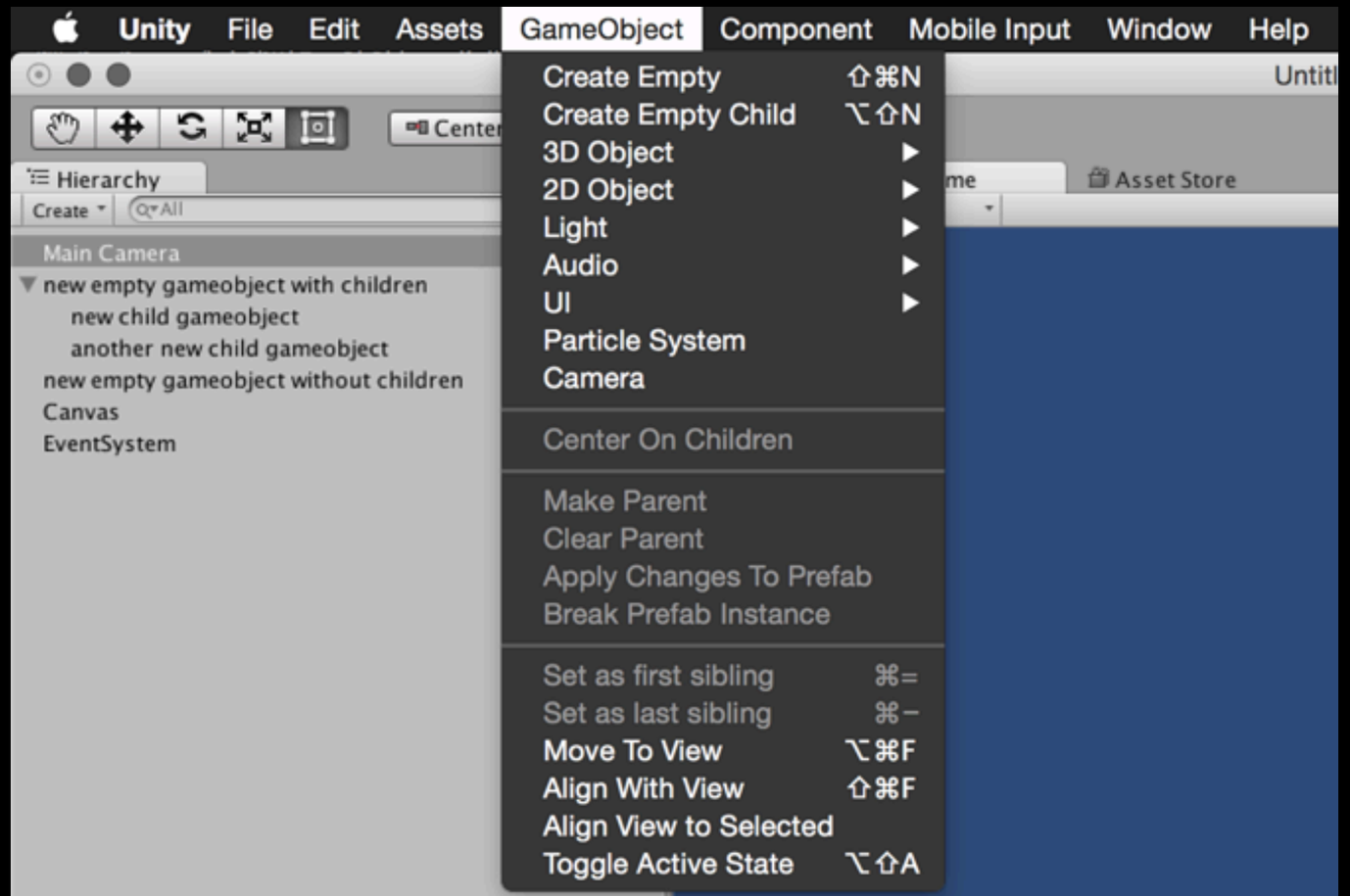


02 - Die Hierarchie und ihre Objekte

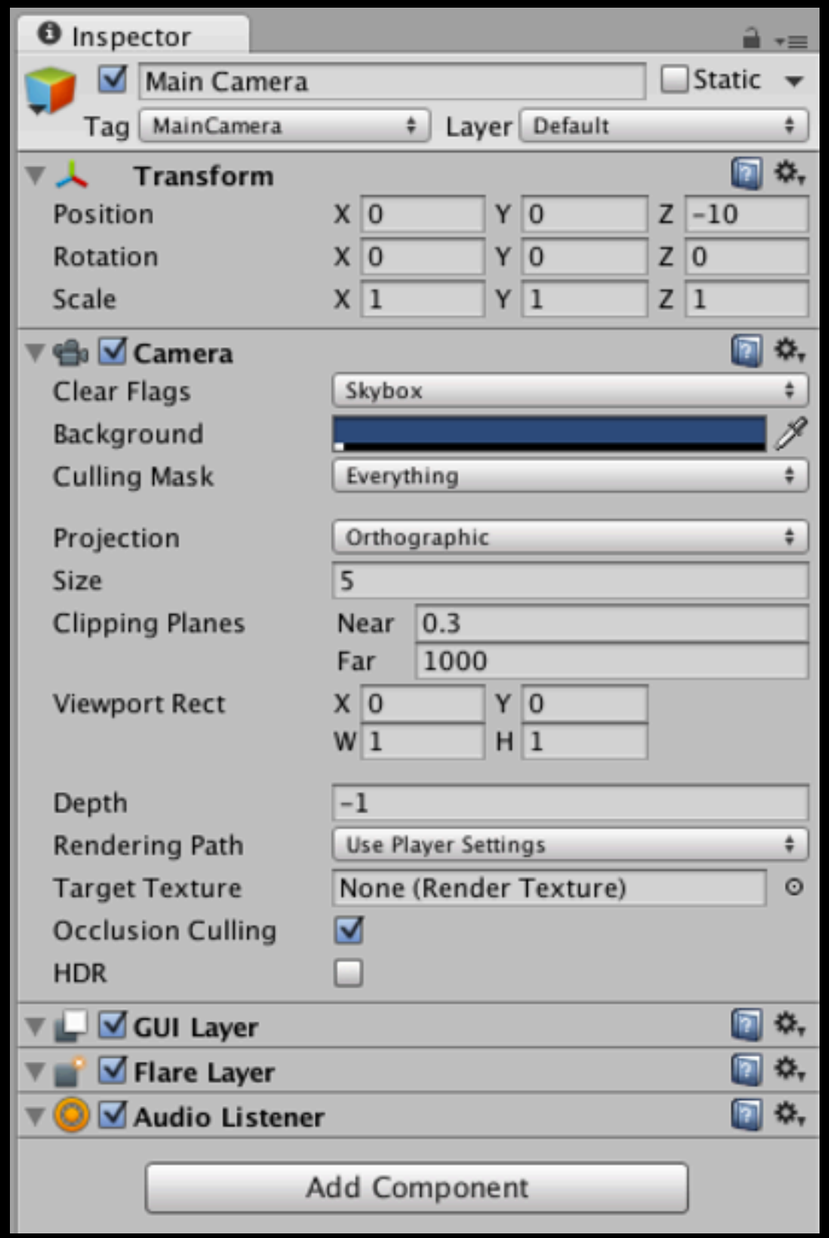
In der Hierarchie sind alle Objekte der aktuellen Szene aufgelistet. Deaktivierte Gameobjects sind ausgegraut.

Zur Erstellung neuer Objekte kann man die rechte Maustaste oder den Reiter „GameObject“ in der Menüleiste benutzen.
Gameobjects dürfen (und sollen in einigen Fällen) auch leer bleiben. Kind-Objekte unterhalb eines Gameobjects können dadurch zusammengefasst ausgewählt werden um einige Funktionen zu erleichtern.





02 - Der Inspector



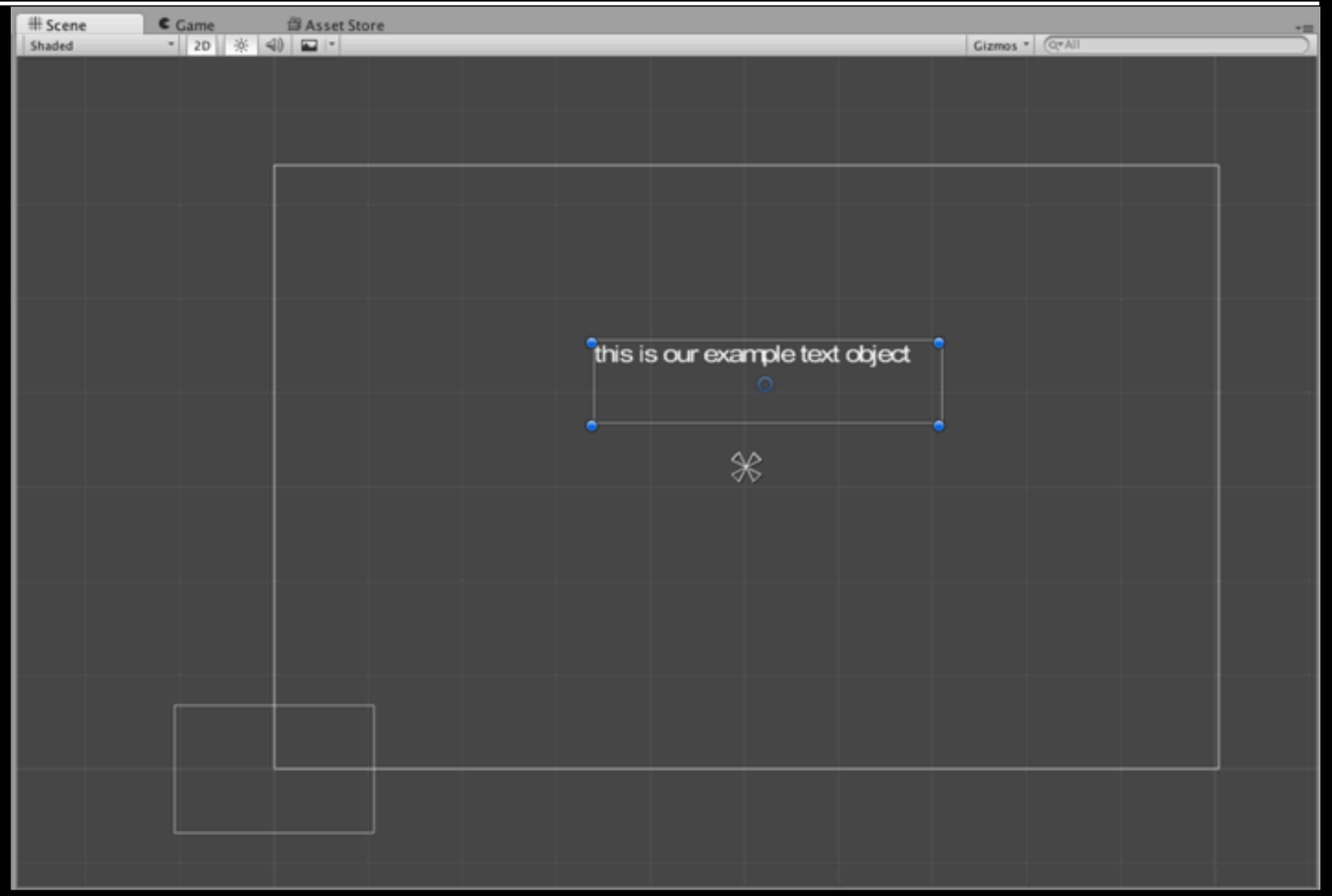
In diesem Beispiel wurde das Gameobject „Camera“ in der Hierarchie ausgewählt. Nun können alle Eigenschaften (z.B. Position und Ausrichtung) dieses Objekts geprüft und verändert werden. Lasst euch nicht von der Fülle an Informationen abschrecken! ;-)

Später gehe ich noch im Detail darauf ein, wie man einem Gameobject Komponenten (Hitboxes, Skripte, Effekte usw.) hinzufügen kann und warum dieses Konzept essentiel für Unity-Projekte ist.

02 - Unsere Scene-View

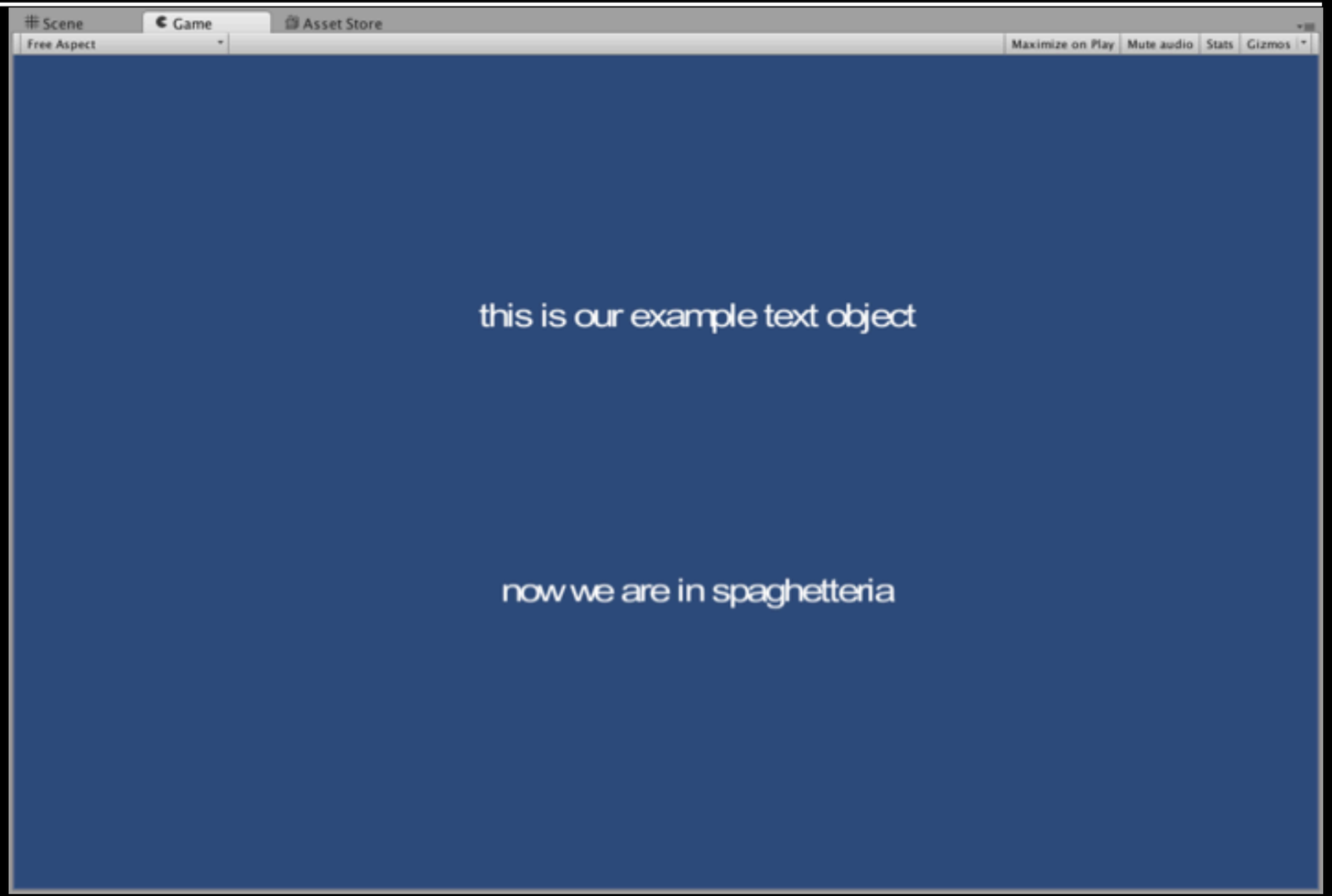
Alle Gameobjects (bis auf ein paar wenige besondere) sind in der Scene sichtbar. In unserem 2D Projekt können sie leicht herumgeschoben werden.

Unser Projekt beinhaltet zu Beginn schon einen Canvas (kleines Rechteck unten links) und eine Camera (Sichtbereich ist das große Rechteck). Alles was sichtbar ist und innerhalb des Sichtbereichs der Kamera liegt wird später gerendert.



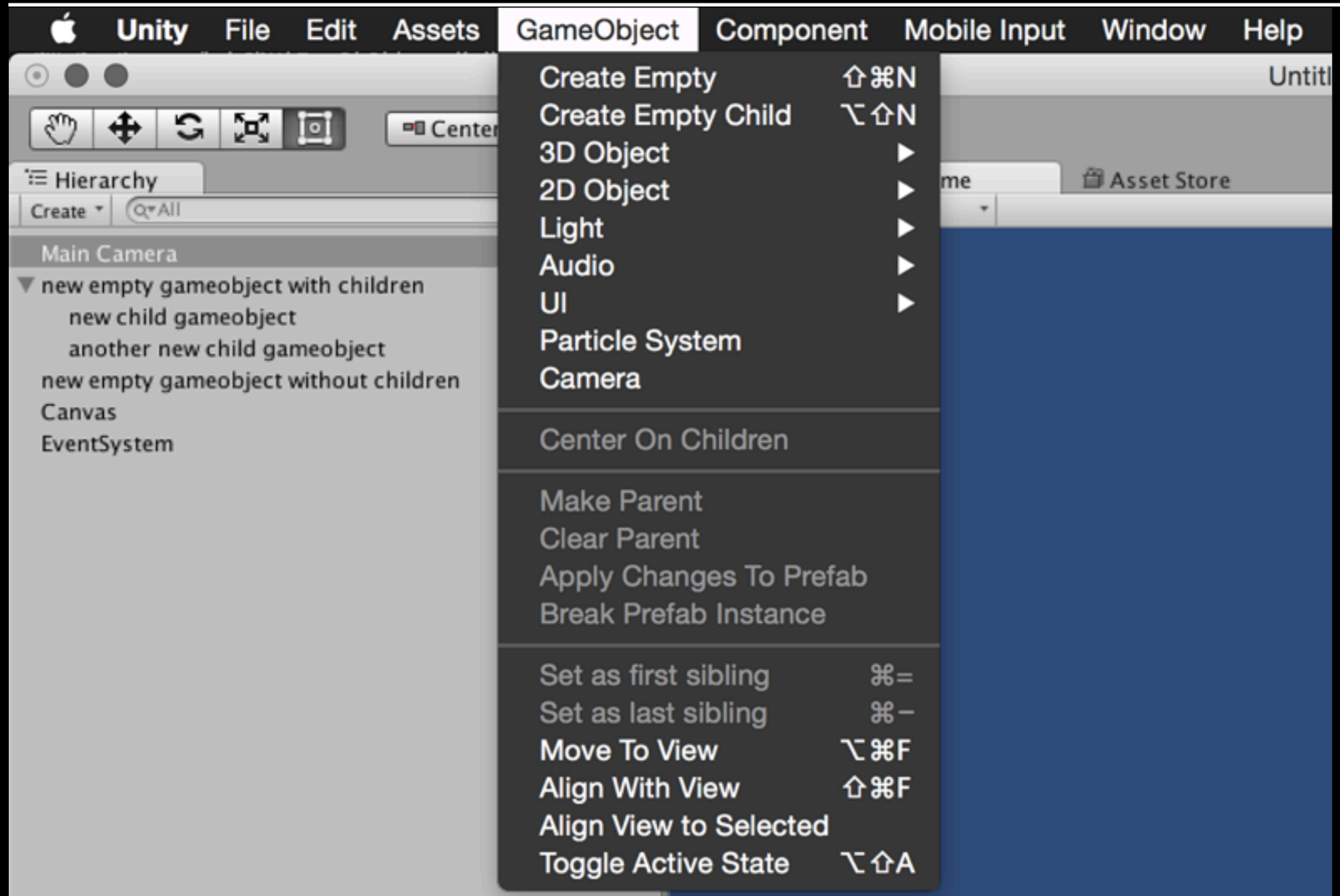
02 - Die Game-View

Wie schon vorhin kurz erwähnt kann das Ergebnis jederzeit in der Game-View betrachtet werden. Hat bis hier her alles bei euch geklappt, solltet ihr in etwa so was hier sehen ⟶



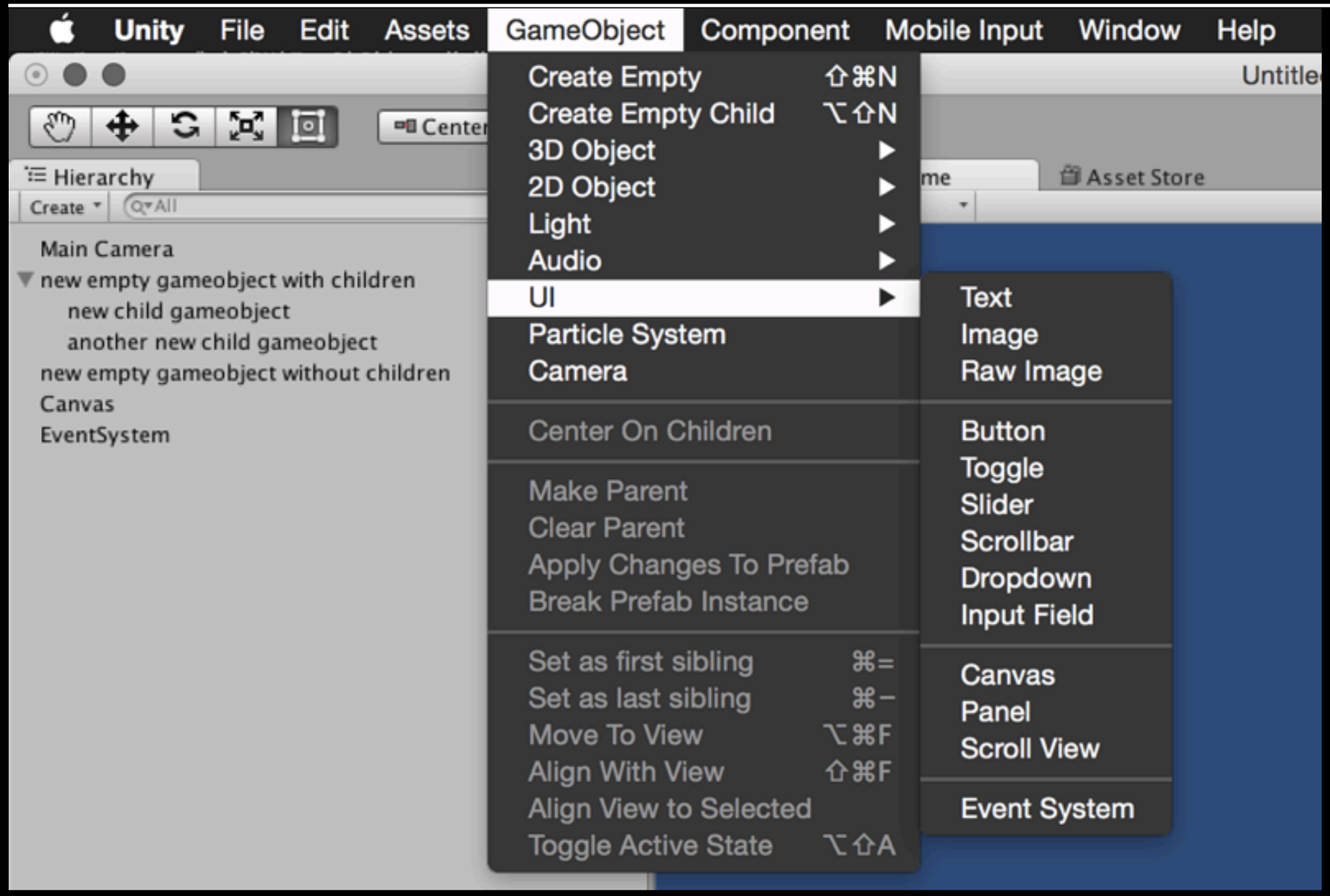
Details zur Game-View werden bei Bedarf noch Live erklärt.

03 - Gameobjects erstellen



Gameobjects könnt ihr mithilfe der Menüs in der Menüleiste oder über das Menü mit der rechten Maustaste erstellen. Zunächst interessieren uns nur leere und UI-Gameobjects. Über UI kann zum Beispiel ein Canvas, sprich „Leinwand“, erstellt werden. Darauf können wir später unser Szene detailreich gestalten.

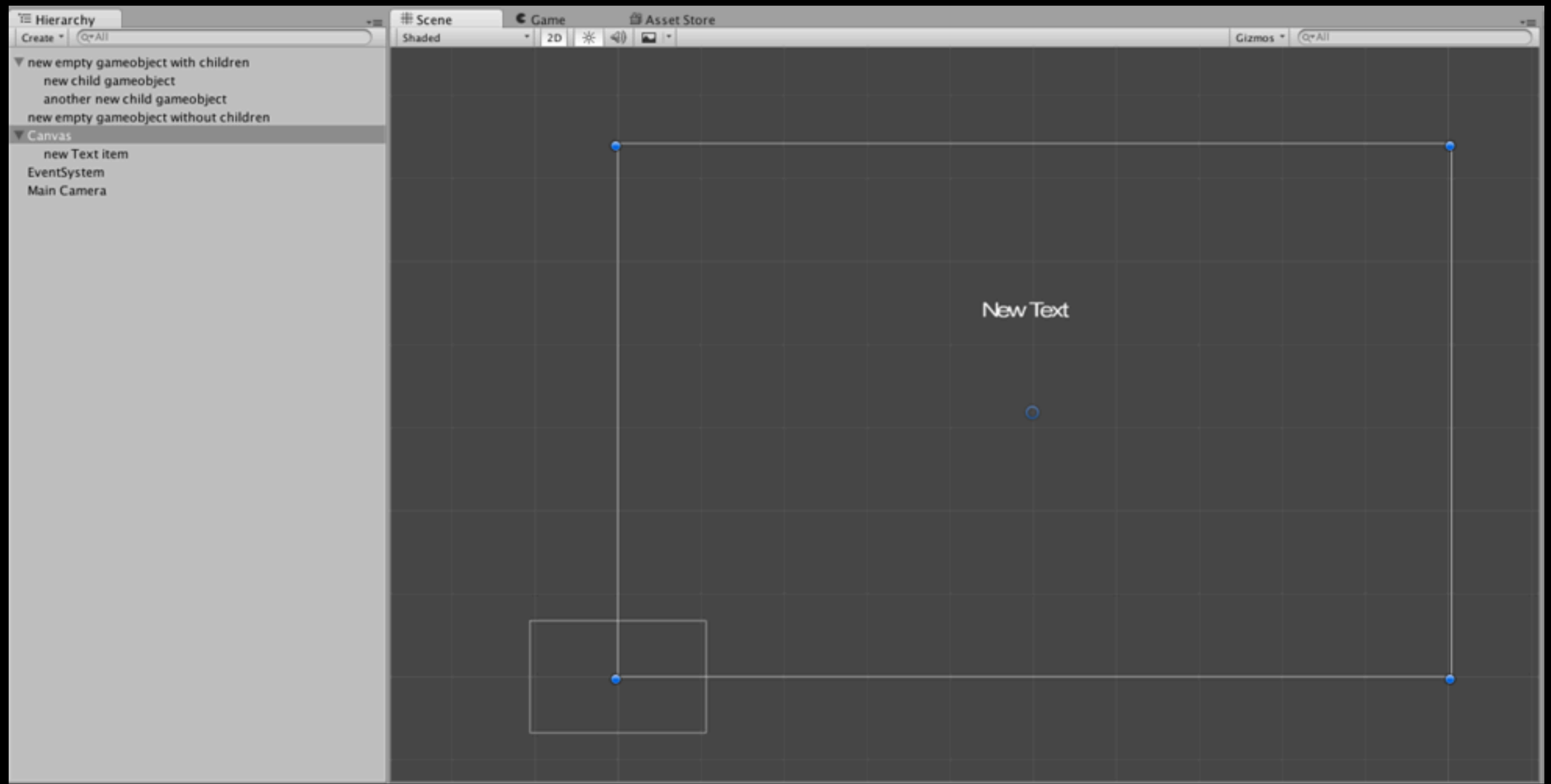
03 - Gameobjects Inhalt geben



An dieser Stelle ist es als Übung erwünscht, dass ihr versucht Bilder und Buttons zu erstellen, und euch diese Objekte im Inspector anschaut. Überlegt euch nachdem ihr die Datentypen und Strukturen von Unity kennt, was für Komponenten man diesen Objekten hinzufügen könnte und wie sie dann eingesetzt werden.

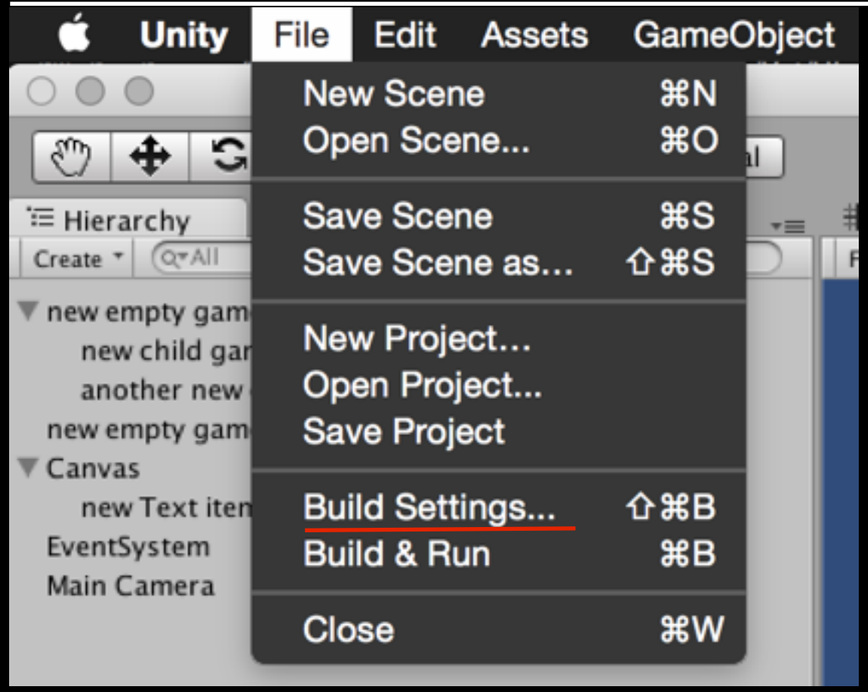
Habt ihr schon die eingerückten Elemente bemerkt? Versucht doch mal die Objekte in der Liste zu vertauschen und zu verschachteln.

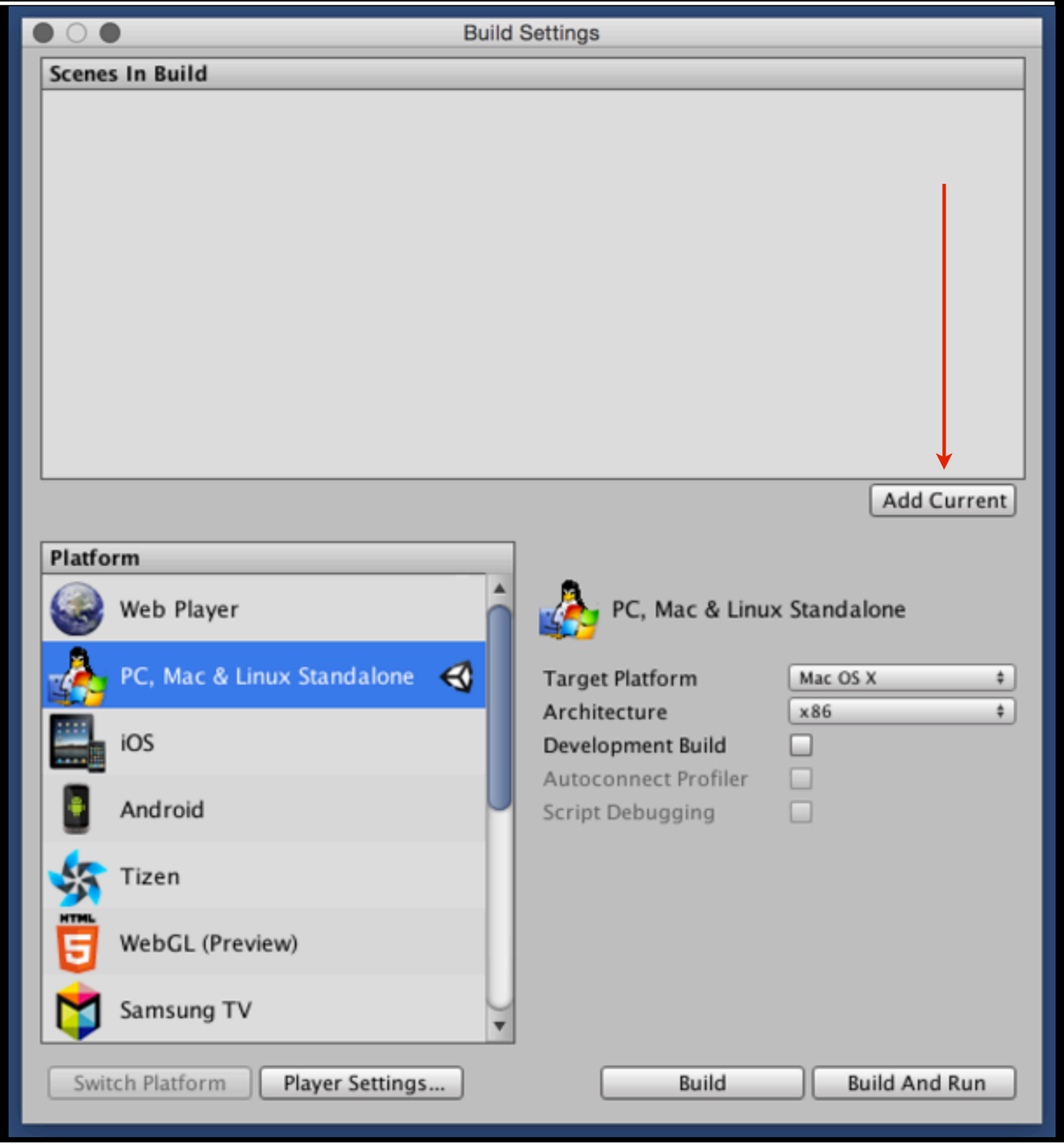
03 - Die Szene gestalten



Objekte in der Szene können einfach über die Hierarchie oder das entsprechende Blueprint ausgewählt werden. In der Scene-View könnt ihr es dann einfach verschieben und neu positionieren. Fügt als Übung neue Text-Objekte ein und stellt sie schön dar!

04 - Allgemeine Projekteinstellungen





Damit der Build später auch funktioniert, müssen die Szenen die später verfügbar sein sollen auch in den Build integriert werden. Dazu geht ihr auf die Build-Settings und fügt die aktuelle Szene zu (ist bereits hinzugefügt im Startprojekt).

05 - Das Szenen-Konzept

In diesem einfachen Beispiel repräsentiert jedes Rechteck eine Szene. Diese können durch bestimmte Ereignisse wechseln.

Ereignisse werden ausgelöst, wenn ein bestimmtes Skript darauf wartet, dass sich z.B. der Szenen- oder Spielstatus verändert und eine entsprechende Funktion aufruft. Aber Achtung: Nicht für jede kleine Veränderung sofort eine neue Szene erstellen!

06 - Unity Datentypen verstehen

Spätestens jetzt müsst ihr wohl oder übel einen kleinen Blick in die Dokumentation von Unity werfen (http://docs.unity3d.com/ScriptReference/).

06 - Unity Datentypen verstehen

Die verwendete Sprache wird C# sein. Erfahrungen mit verwandten Sprachen wie C, C++ oder Java sind von Vorteil da die Syntax C-ähnlich ist, allerdings nicht voraussetzung.

Skripte können direkt neu erstellt und bearbeitet werden (Projektstruktur, Create New: Script). Vorteil von C# ist dabei, dass jedes erstellte Skript eine eigene Klasse darstellt, und jede Klasse alle anderen verfügbaren Klassen von Beginn an sieht. Das Schwierige jetzt:

Wie spreche ich Gameobjects an und was kann ich damit machen?!

Dafür bietet uns Unity die hauseigene Klasse GameObject.

06 - Unity Datentypen verstehen

C# ist heftig objektorientiert, also hoffe ich dass ihr objektorientiert coden könnt (die jenigen von euch die Coden wollen). Als kurze Erinnerung:

Gameobjects haben Attribute:
- Ist das Objekt aktiv?
- Ist das Objekt statisch?
- Wo liegt unser Gameobject gerade? (Transform)

Gameobjects haben Public-Funktionen:
- Aktiv oder Inaktiv setzen
- Komponenten hinzufügen oder entfernen

Gameobjects haben statische Funktionen:
- Objekt finden, klonen, zerstören, bei Szenenwechsel nicht zerstören etc.

Bitte schaut euch in der Dokumentation ein paar weitere Klassen an die interessant sein könnten. Hier ein paar Anregungen ;-)

Animation **Audio** **(Box)Collider** **Canvas**

Camera **Event** **GUI** **Input**

Material **Object** **Particle** **Physics2D**

Rect :D **Screen** **Sprite** **Vector2**

Ich hoffe diese kleine Einführung macht euch den Einstieg einfacher und hat ein klein bisschen was gebracht. Fröhliches Coden!

Tübingen - Codevember 2015

```
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;
  }
}
```

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!

07 - 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.

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()
```

08 - 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!
}
```

08 - 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.
}
```

09 - 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;
        }
    }
```

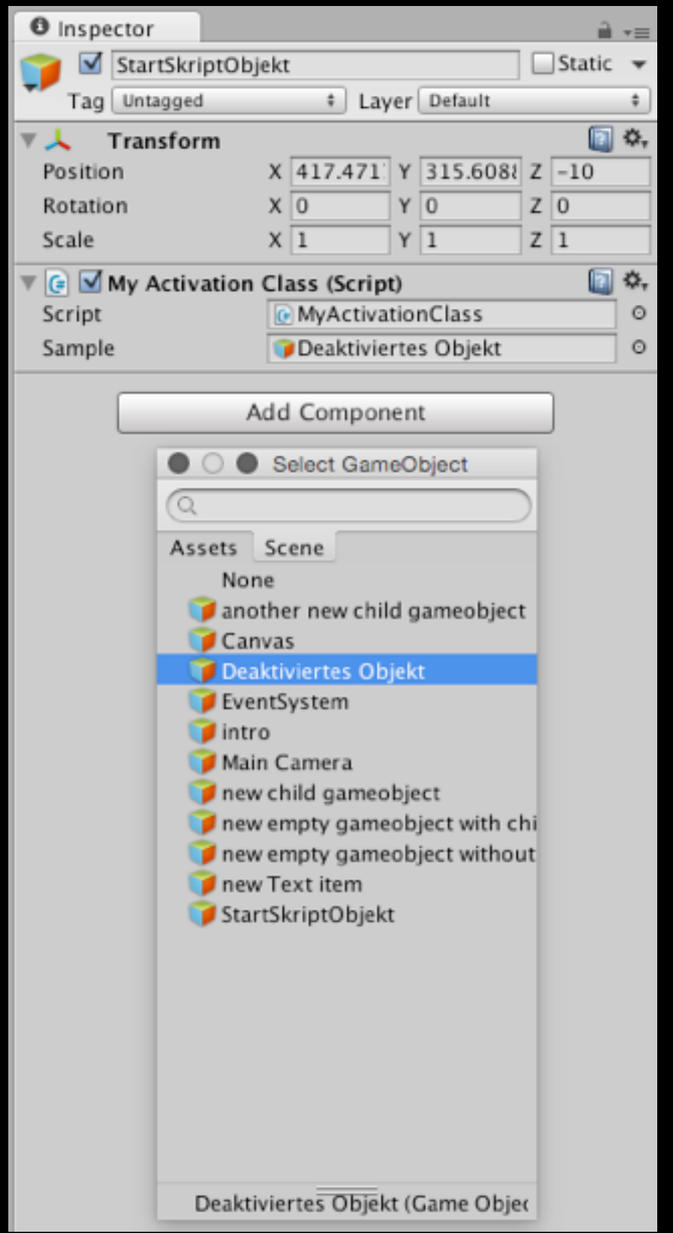
10 - Attribute in der Klasse deklarieren

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);
    }
}
```



11 - 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!).

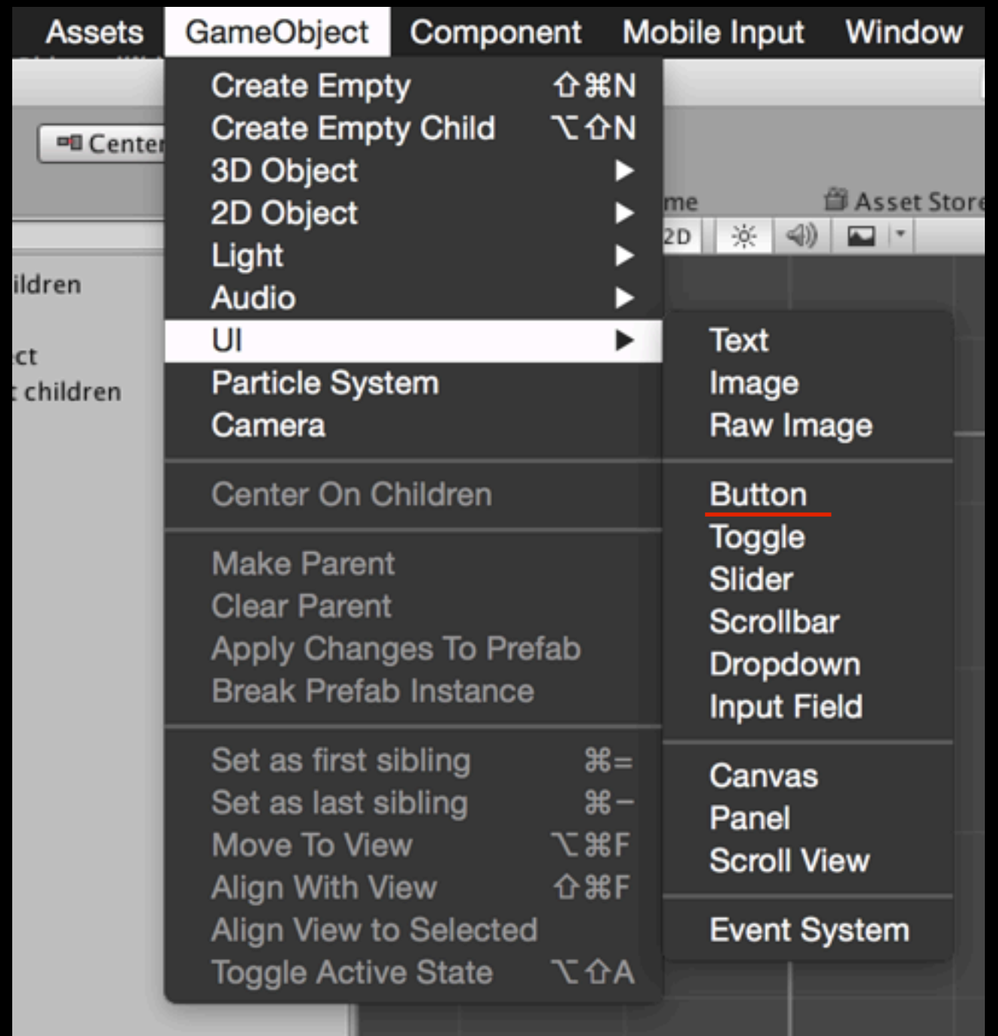
Möchte man etwa, dass ein Gameobject sich selbst deaktiviert könnte man das hier benutzen:

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

12 - Beispiel zum Laden einer Szene

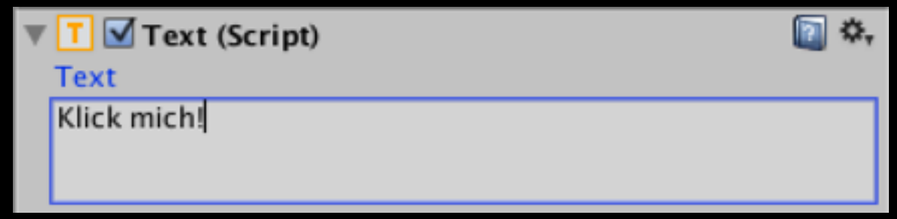
Mit dem bisherigen Wissen, sollte es nun möglich sein ein Objekt zu erstellen, mit dem wir eine Szene wechseln können.

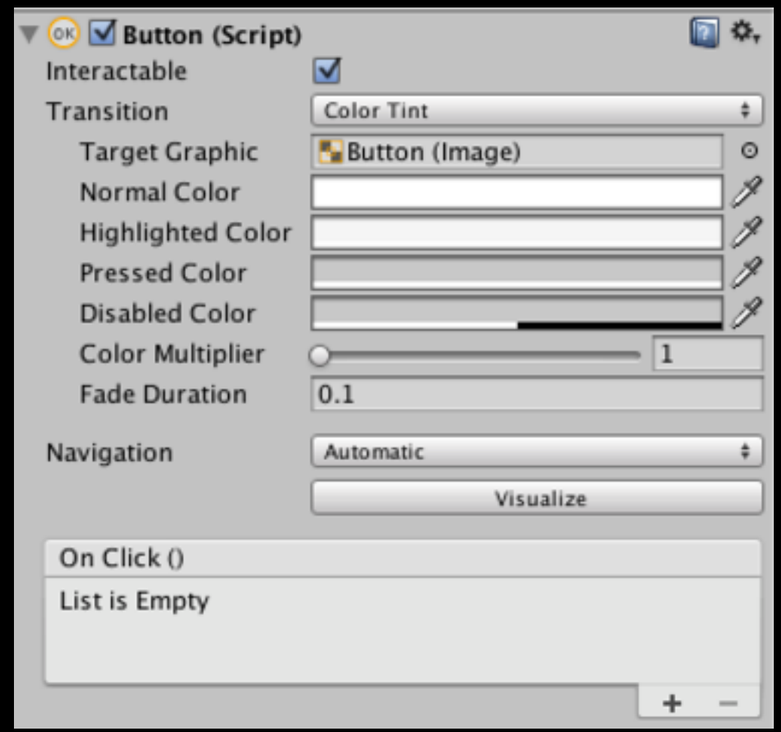
Vorausgesetzt wir haben mindestens zwei Szenen, sollten wir ein neues Gameobject in der ersten Szene erstellen. Hier würde sich ein Button anbieten.
Diesem sollten wir einen Collider als Komponente hinzufügen, damit unser Button auch weiß, dass etwas mit ihm gemacht wurde.



12 - Beispiel zum Laden einer Szene

Wie ihr sehen könnt, wurde unser Button automatisch mit einem Kind Gameobject versehen Namens "Text" versehen. Wir geben diesem "Text" etwas mehr Bedeutung und ändern den Inhalt zu "Klick mich!".

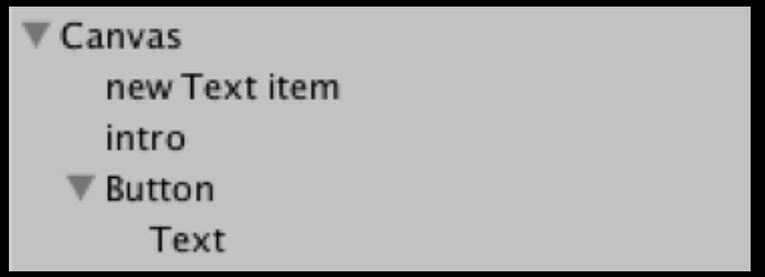


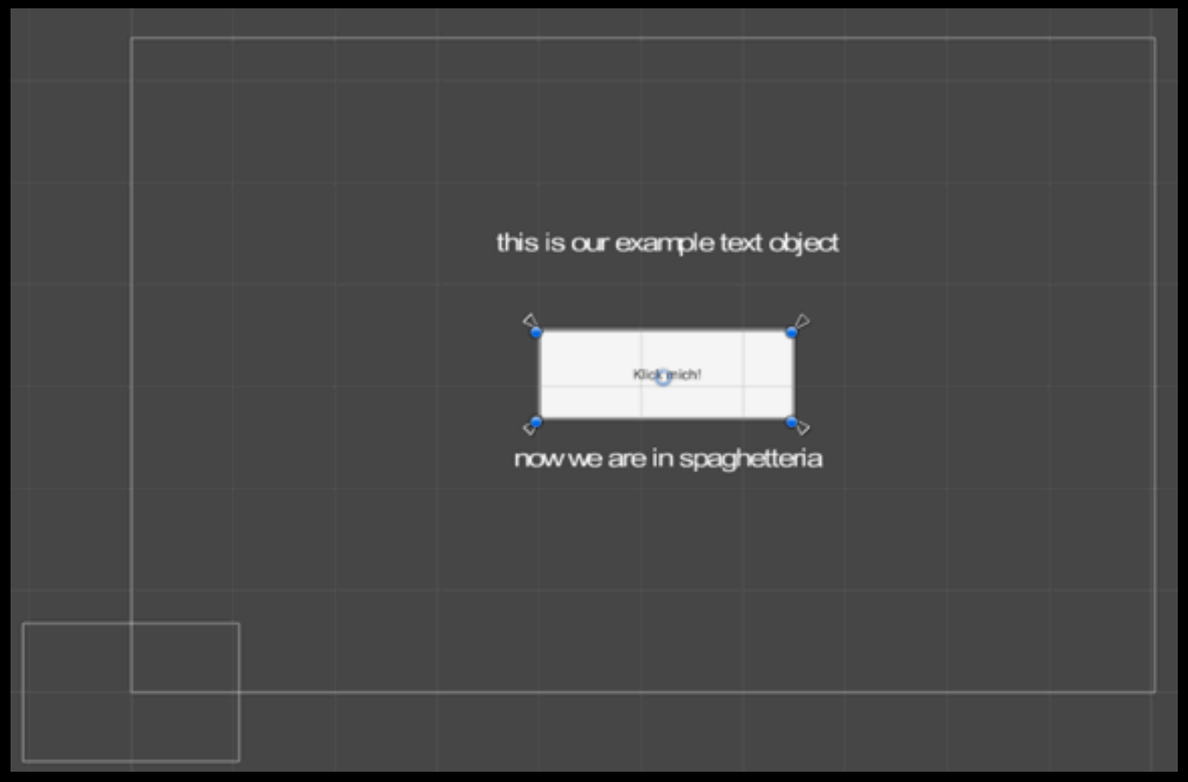


Beachtet hierbei, dass der Button bereits ein Button-Script hat, welches ein "OnClick" Event starten kann, da es aus der UI-Liste stammt.
Unter der OnClick() Funktion können wir ein Script auswählen, welches dann die Funktion ausführt, die unsere Szene wechselt.

12 - Beispiel zum Laden einer Szene

Damit der Button sichtbar ist, schieben wir ihn in der Hierarchie als Kind Gameobject zu unserem Canvas und vergrößern ihn wein wenig. Damit stellen wir sicher, dass man den Button auch sieht.





12 - Beispiel zum Laden einer Szene

Fleißige und motivierte Menschen können doch jetzt mal versuchen, über die Doku die richtigen Funktionen zum Szenenwechsel zu finden und einfach mal ein kleines Script für den Button schreiben!

Viel Erfolg ;-)