Autorensysteme / Virtuelle Realität & Animation

Sommersemester 2023

Stefan Wehrenberg

Unity

Übersicht – Unity Grundlagen

- Allgemein
- Benutzeroberfläche
- Basiselemente
- Skripte
- Input Manager
- FBX Import (z.B. Blender)
- Animation
- Audio
- Asset Store
- Export / Build

Allgemein

- Unity ist eine seit 2005 entwickelte Game Engine dessen Oberfläche der von 3D-Modellierungs- und Animationssoftware ähnelt

- Erlaubt Entwicklung für PC, Konsolen und Webanwendungen (2D, 3D, AR, VR etc.)

- Wird neben der Spielebranche auch in der Automobilindustrie, Architektur, Film & Fernsehen, Konstruktion und selbst vom Militär für z.B. Simulationen verwendet

Allgemein

- In Unity entwickelte Spiele umfassen z.B. Pokemon Go, Hearthstone, Beat Saber, Subnautica, Rust, Hollow Knight (ca. 50% aller Spiele, 70% aller mobilen Spiele, die heute den größten Marktanteil haben)

- Neben der Entwicklung mittels der von Unity zur Verfügung gestellten Werkzeuge werden auch C# Skripte eingebunden

- Unter einem jährlichen Gewinn von 100.000$ ist Unity komplett mit einer kostenlosen Lizenz verwendbar

Allgemein

- Rechner in Labor 0.31 haben die aktuellste (15.03.23) **LTS Version 2021.3.21f1** mit Android, Windows und WebGL Build Support installiert

- Um ein neues Projekt zu öffnen, wird im Unity Hub links ***Projects***, dann rechts oben ***New project*** ausgewählt

- Unity bietet eine Auswahl an Templates und Lernprojekten. Obwohl detaillierte Templates existieren, ist das **3D/2D Core** Template zu empfehlen, um volle Kontrolle über die installierten Packages zu haben

Benutzeroberfläche – Szenen

- Szenen in Unity können einzelne Level eines Spiels oder auch Menüs darstellen. Einfache Spiele werden meistens in zwei Szenen entwickelt (UI & Spielumgebung)

- Jedes 3D Projekt startet mit der *SampleScene*, einer leeren (außer Licht und Kamera) Umgebung

- Szenen können exportiert und als Templates wiederverwendet werden. Grundlegend existieren ein paar Basis Templates in Unity

Benutzeroberfläche – Views

- Der Inhalt der aller Szenen wird in der *Hierarchy* dargestellt. Regulär ist diese am linken Rand des Editors zu finden

- Die aktuelle Szene wird im *3D View* in der Mitte des Editors angezeigt.

- Der *Scene Tab* des 3D View zeigt eine dynamische Ansicht, während der *Game Tab* die Ansicht der *MainCamera* darstellt

Benutzeroberfläche – Views

- Der *Play Button* über dem *3D View* lässt eine Vorschau des Spiels laufen. Erneuter Klick auf Play beendet die Vorschau

- Änderungen an Objekten im *Play Mode* werden nicht übernommen und nach Beenden der Vorschau zurückgesetzt

- Der *Stop Button* hält die Preview in ihrem momentanen Zustand an. Skripte werden im gestoppten Zustand nicht weiter aufgerufen

Benutzeroberfläche – Views

- Der *Project Tab* im unteren Fenster zeigt eine Übersicht über alle im Projekt enthaltenen Verzeichnisse und Dateien. U.a. befindet sich hier der *Assets Ordner* in dem Modelle, Materialien, Skripte, Audiodateien, Animationen u.v.m. organisiert werden können

- Der *Packages Ordner*, enthält zu Beginn alle importierten Standardpakete

- Der *Console Tab* ist eine reguläre Debugging-Konsole. Hier werden Fehler in Skripten, Debug-Meldungen oder Textausgaben des Codes angezeigt

Benutzeroberfläche – Views

- Der *Inspector* rechts im Editorfenster gibt detaillierte Informationen zum aktuell ausgewählten *GameObject* und dessen Bestandteilen (*Components*) und erlaubt deren Bearbeitung

- Mittels des *Layout Dropdown* am rechten oberen Eck des Editorfensters kann zwischen verschiedenen Standard Layouts umgeschaltet werden. Die einzelnen Fenster des Unity Editors können aber auch nach Belieben per Drag-and-drop bewegt werden

Basiselemente – GameObjects

- Jedes Element der Szene wird in Unity grundlegend als *GameObject* bezeichnet

- Dazu gehören jegliche 3D-Objekte, der Spieler selbst und die Kamera, aber auch UI-Elemente, Lichter und Effekte

- Mittels des *GameObject DropDown Menüs* (links oben), per Rechtsklick in die *Hierarchy* oder mittels des Plus-Symbols links über der *Hierarchy* können neue *GameObjects* eines beliebigen Typs erzeugt werden

Basiselemente – GameObjects

- Die *GameObjects* können in Kind-Eltern-Beziehungen zueinander stehen. Diese Beziehungen können durch Drag-and-Drop für bestehende Objekte erstellt werden oder indem diese direkt als Kind erstellt werden

- Jede Veränderung des *Parent* beeinflusst auch das *Child GameObject*. Die Werte des Child sind relativ zum Parent. So ist beispielsweise Position (0,0,0) des *Child* immer die aktuelle Position des *Parent*

Basiselemente – Components

- Jedes *GameObject* besteht aus einer variablen Anzahl von *Components*. Diese sind die funktionelle Grundlage für jedes *GameObject*

- Die *Components* enthalten verschiedenste Eigenschaften, die das Verhalten eines GameObjects maßgeblich beeinflussen

- Im Folgenden gehen wir auf ein paar der wichtigeren *Components* genauer ein

Basiselemente – Components

- **Transform**: Die *Transform Component* legt die Position eines *GameObjects* in Relation zu ihrem *Parent* fest (auf höchster Ebene die Szene selbst). Dazu gehören Translation, Rotation und Skalierung entlang der Koordinatenachsen

- **Mesh Filter**: Legt fest, welches *Mesh Asset* mit dem ausgewählten *GameObject* verbunden ist. *Mesh Assets* sind die grundlegende Struktur (*Vertices*, *Edges*, *Faces* etc.) von 3D-Modellen. Zu einem *Mesh Filter* gehört regulär eine *Mesh Renderer Component*

Basiselemente – Components

- **Mesh Renderer:** Diese *Component* sorgt für die Darstellung im 3D Raum und legt u.a. Material und Beleuchtungseinstellungen für ein zugehöriges *Mesh* fest

- **Collider:** *Collider* werden verwendet, um Kollisionen mit anderen Objekten zu erkennen. *Collider* können aus primitiven Formen (*Box*, *Capsule* und *Sphere*) oder basierend auf einem bestehen *Mesh* erzeugt werden.
  - Die Kollisionserkennung für *Mesh Collider* ist genauer, aber leistungsintensiver, basierend auf der Komplexität des *Meshes*. *Collider* sind im 3D View als grünes Wireframe zu erkennen

Basiselemente – Components

- **Rigidbody:** Erlaubt dem *GameObject* in Echtzeit auf physikalische Effekte zu reagieren. Dazu gehören u.a. Anziehungskraft, Masse, Trägheitsmoment und Schwung

- **Constraints:** Ähnlich wie in Blender werden hiermit Bedingungen zwischen mehreren Objekten, hier *GameObjects*, festgelegt. Zu den *Constraints* gehören Position, Rotation, Skalierung oder auch *Aim* oder *Look At*, die das *GameObject* auf das Verbundene ausrichten.

Basiselemente – Materials

- Texturen sind 2D Bilder, die auf die Oberfläche eines *3D Objects* gelegt werden um, visuelles Detail zu verleihen.

- Die *Materials* definieren physikalische visuelle Eigenschaften einschließlich Farbe, Transparenz, Reflektivität und Texturen Mapping.

- Die *Physic Material Component* legt die physischen Eigenschaften des Materials fest. Dazu gehört die Sprunghaftigkeit (*bounciness*) und Reibung (*friction*)

Basiselemente – Prefabs

- Das *Prefab* ist ein Template eines vorkonfigurierten *GameObjects* mit all seinen *Components*, Eigenschafts-werten und seinen Kind-Elementen (*GameObjects* inklusive deren Konfiguration)

- Um aus einem bestehen *GameObject* ein *Prefab* zu erzeugen, muss dieses lediglich aus der *Hierarchy* in einen Asset-Ordner gezogen werden

- Änderungen werden automatisch für alle *GameObjects* übernommen, die auf dem *Prefab* basieren, sofern das Attribut nicht manuell geändert wurde.

Basiselemente – Tags

- Tags sind Referenzwörter, die *GameObjects* zugewiesen werden können, anhand derer diese identifiziert werden können. Es kann nur ein Tag zugewiesen werden

- Mittels der Tags können *GameObjects* innerhalb der Skripte einfacher, und weitaus Performance-optimierter, ermittelt werden

- Tags können oben im *Inspector* für das ausgewählte *GameObject*, direkt beim Namen, zugewiesen werden. Neben den regulär definierten Tags können auch Neue definiert werden

Skripte – Default Editor

- Bei der Installation von Unity wird regulär eine aktuelle Version von Microsoft Visual Studio installiert

- Optional kann auch ein anderer Editor genutzt werden. Dieser muss unter *Edit -> Preferences -> External Tools* als Standard festgelegt werden

- Auf den Laborrechnern sollte sowohl Microsoft Visual Studio als auch Visual Studio Code (Extensions werden per User installiert – braucht keine Berechtigung) installiert sein

Skripte – Start und Update

- Das regulär generierte Skript enthält von Beginn an zwei Methoden, die keinen Rückgabewert haben. Die *Start()* und die *Update()* Funktion. *Start()* wird einmalig im ersten Frame ausgeführt, zu dem das zugehörige *GameObject* in der Szene aktiv ist.

- Die *Update()* Methode wird für jedes Frame, in dem das *GameObject* nach Start des Spiels aktiv ist, ausgeführt. Regulär läuft Unity bei 60 FPS, was bedeutet, dass das Update 60 Mal in der Sekunde aufgerufen wird, solang das *GameObject* aktiv ist.

Skripte – Nützliche Funktionen

- Die *GameObject.Find(string name)* Methode erlaubt die gezielte Suche nach einem *GameObject*. Sie sollte aber aus Performance-Gründen nicht jedes Frame aufgerufen werden

- Die Verwendung von aussagekräftigen Tags und den Methoden *GameObject.FindWithTag(string tag)* und *GameObject.FindGameObjectsWithTag(string tag)* erlaubt eine Filterung der gesamten Menge der *GameObjects*. Daraus folgt eine deutlich bessere Performance

Siehe https://docs.unity3d.com/ScriptReference/GameObject.html für eine Übersicht aller Methoden von GameObject

Skripte – Nützliche Funktionen

- Mit den *GetComponent(Type component)* Methoden können einzelne *Components* in *GameObjects* ermittelt werden, um auf deren Eigenschaften zuzugreifen

- Die *transform.Translate(Vector3 vector)* und *transform.LookAt(GameObject object)* Methoden eines *GameObjects*, erlauben dessen Translation im Koordinatensystem des *Parent*

- Mit der Variablen *Time.deltaTime*, die das Intervall in Sekunden seit dem letzten Frame darstellt, kann eine flüssige Bewegung erzeugt werden

Skripte – Nützliche Funktionen

- Die *Debug.Log(string text)* Funktion gibt den übergebenen Debugging-String *text* in der Konsole aus.

- Die private Funktion *OnCollisionEnter(Collision other)*, wird bei Beginn jeder *Collision* aufgerufen. Lediglich die Überprüfung auf eine *Collision* wird bei jedem Update Aufruf überprüft. Der übergebene *Collision* Parameter enthält unter anderem das *GameObject* mit dem das Objekt an dem das Skript hängt zusammengestoßen ist

Skripte – Nützliche Funktionen

- Die *Instantiate(GameObject object)* Methode erzeugt ein *GameObject* basierend auf einem referenzierten *GameObject* oder *Prefab object*

- Die *Destroy(Object object)* Funktion zerstört das referenzierte Objekt *object* (*GameObject* oder *Component*)

- Die Funktion *Input.GetKeyDown(string key)* lässt die Betätigung einer Taste überprüfen. Die ähnliche *Input.GetKey(string key)* Funktion überprüft gedrückt Halten einer Taste

Skripte – Public Variablen

- Variablen, die in Skripten als *public* deklariert werden, können direkt im *Inspector* neuen Werten zugewiesen werden

- Diese Funktion ist nützlich, wenn *GameObjects* z.B. dasselbe Skript verwenden sollen, jedoch mit unterschiedlichen Parametern, oder um Referenzen zu Objekten zu erzeugen

- Sind die öffentlichen Variablen *GameObjects*, oder andere Assets, können diese direkt in den *Inspector* gezogen werden, um sie per Drag-and-drop zuzuweisen

Input Manager

- Der Input Manager ist unter ***Edit -> Project Settings -> Input Manager*** zu finden

- Hier können unter *Axes* die regulären Tasten-Zuweisungen der Input API eingesehen und bearbeitet sowie Neue definiert werden

- Es ist immer möglich im Skript, mit den zuvor beschriebenen Funktionen, mittels Strings direkt auf die Tastennamen zuzugreifen, anstatt der *GetAxis()* oder *KeyCode()* Funktionen

FBX Import (z.B. Blender)

- 3D Modelle aus z.B. Blender können importiert werden, indem die entsprechende *FBX-Datei* in den Asset-Ordner gezogen wird

- Hierbei werden *Meshes*, *Materials* und *Animationen* direkt in einem eigenen Verzeichnis importiert.

- Wird das gesamte importierte Objekt in die Szene gezogen, wird die *Animation* vorerst noch nicht korrekt ausgeführt. Dafür ist ein *Animation-Controller* nötig

Animation

- Animationen werden in Unity durch eine Animator *Component* zu einem *GameObject* hinzugefügt.

- Diese benötigt einen *Animation Controller*, der festlegt, welche Animationen wann abgespielt werden sollen. Der Controller wird direkt im Assets Ordner mittels ***Rechtsklick -> Create*** hinzugefügt.

- Die restlichen Optionen der Animator Component sind für einfache Animationen irrelevant. Mehr zu Animationen in diesen Youtube-Videos: Link

Audio

- Wie andere Assets können Audiodateien einfach per Drag-and-drop in den Asset-Ordner gezogen werden

- Diese können dann als *AudioSource Component* an ein *GameObject* angehängt werden

- Die *Play()* Methode einer *AudioSource Component* erlaubt das gezielte Abspielen der Audiodatei. So können Soundeffekte beispielsweise im Skript direkt an Events geknüpft werden.

Audio

- Regulär werden Audiodateien als Background Sound ohne Quelle im 3D-Raum abgespielt.

- Mittels *3D-Sound* kann auch Ton hinzugefügt werden, der Lautstärke oder sogar Tonhöhe abhängig von der Position des *Audio Listeners* ändert, so wie sich Klang im realen Raum verhält.

- Dies kann durch die Aktivierung von *Spatial Blend* und Anpassung der Min./Max. *Distance* & *Rolloff* erreicht werden

TextMeshPro Package

- Erlaubt das Einfügen von Text und anderen UI Elementen als *GameObject* und ist im UI-Bereich der *GameObjects* zu finden

- Benötigt zusätzliche *Packages*, auf die Unity automatisch beim ersten Einfügen von *TextMeshPro* Elementen hinweist

- Wird oft für kleine UI Displays als *Childobject* direkt an die *MainCamera* gehängt und dynamisch im Skript editiert oder in separaten Szenen erzeugt (Unterscheidung zwischen reinem Text und UI!)

Asset Store

- Im Asset Store (https://assetstore.unity.com/) kann eine riesige Auswahl an kostenlosen und kostenpflichtigen Assets über den Unity Account geladen werden

- Die Assets sind nach Download über den *Package Manager* in separaten *Packages* im *Project Tab*

- Wichtig: Geben Sie bei der Abgabe unbedingt alle Quellen (wie z.B. Assets aus dem Store) an, die verwendet wurden.

Export

- Um das Spiel z.B. als Windows Executable (*.exe*) zu exportieren, wird unter ***File -> Build Settings*** die gewünschte Voreinstellung vorgenommen und exportiert

- Dieser Prozess kann je nach Asset-Menge und Kompression recht langwierig sein

- In ***File -> Project Settings -> Player*** oder den *Player Settings* links unten in den *Build Settings* können u.a. Name, Version, Icon und Auflösung des exportierten Spiels festgelegt werden

Tutorials & Online Quellen

- https://learn.unity.com/
  - Unity Essentials Pathway
    - Sehr lang (8-12h, Grundlagen ca. 3h)
    - Sehr ausführlich und interaktiv

- https://www.youtube.com/watch?v=VnN5MYQnGak
  - Youtube: Interaktiver Unity Komplettkurs (2.5h)

- https://www.linkedin.com/learning/unity-grundkurs-1-schritt-fur-schritt-zum-ersten-eigenen-spiel?u=82265442
  - LinkedIn Learning: Interaktiver Unity Grundkurs (3.5h)

Unity VR

Übersicht – Unity VR

- Package Import
- Player / CameraRig
- OpenXR Runtime
- Bewegungstypen
- Interaktionen
- Pico Neo Setup
- Development Tipps

- Tutorials und Onlinequellen
- Showcase älterer Projekte

Import eines VR-Package

- Mit einem regulären 3D-Template starten
  - VR Template benötigt einige Voreinstellungen, die zumindest mit Open XR im 3D-Template nicht nötig sind
- Open XR / XR Toolkit (Einrichtung auf folgenden Folien)
  - Kann auf jeglicher Hardware entwickelt werden
- SteamVR, Oculus Plug-Ins etc. (Asset Store)
  - Je nach verwendeter Hardware auf andere Weise einzustellen
  - Siehe jeweilige Dokumentation
  - Das SteamVR Plugin ermöglicht nur Windows Builds. Android Applikationen sind nicht möglich.

Import eines VR-Package

- Installieren des XR Plug-In Management System
  - ***Edit -> Project Settings -> XR Plug-In Management***

- Plug-In Provider festlegen: Open XR
  - Bei einer Aufforderung zum Neustart des Editors zustimmen
  - Open XR übersetzt Eingaben verschiedenster HMD/Controller auf den selben Standard
  - ACHTUNG: Die Pico Neo 3 HMDs im Labor haben noch keinen reinen Open XR Support und müssen mit einem eigenen SDK Package verwendet werden.
    - Sollten Sie inzwischen unterstützt werden müsste ein Pico spezifisches *Interaction Profile* vorliegen (Siehe nächste Folie)

Import eines VR-Package

- Einfügen eines *Interaction Profiles*
  - ***Edit -> Project Settings -> XR Plug-In Management -> OpenXR***
  - Controller Profile die übersetzt werden sollen hinzufügen
  - Bei APK Entwicklung (Standalone HMDs) müssen die Einstellungen im Android Tab erledigt werden, statt im PC Tab

- Sollte ihre Applikation nur auf einem Display des HMD, anstatt auf beiden dargestellt werden
  - Setzen des Render Mode von Single Pass (Duplizieren eines Render auf beide Displays) auf Multipass (Separates Rendern beider Bilder

Import eines VR-Package

- Open XR übernimmt lediglich die Kommunikation mit dem HMD. Für die Übersetzung der Eingaben in verwendbare Befehlsfunktionen wird das XR Interaction Toolkit benötigt
- Installation über ***Window -> Package Manager***
  - Open XR / XR Plug-In Management sollten jetzt bereits in den Projekt Packages enthalten sein
  - Dropdown von *Packages: In Project* zu *Unity Registry* umschalten
  - *XR Interaction Toolkit* auswählen und installieren
  - Im Package Manager des XR Interaction Toolkit unter Samples die *Starter Assets* importieren

Import eines VR-Package

- Den Controller Objekten müssen im *XR Controller Skript* die passenden Presets zugewiesen werden (Button oben rechts an der Component)
- In den Assets ***Samples -> XR Interaction Toolkit -> 2.0.4*** (Oder ihre geladene Version) ***-> Starter Assets*** finden Sie vorgefertige Interaktions-Presets
- Hier befinden sich fünf Presets: *Continuous Move*, *Continuous Turn*, *Left Controller*, *Right Controller* und *Snap Turn*

Erstellen eines VR Players

- Sind alle Installationen und Einstellungen erfolgreich verlaufen kann über ***GameObject -> XR*** entweder das *XR Origin* (Ohne Controller) oder das *XR Origin (VR)* (mit Controllern) eingefügt werden

- Sollte bereits eine *MainCamera* vorhanden gewesen sein sollte diese automatisch mit allen Child-Elementen in das *XR Origin* verschoben werden. (Muss eventuell manuell erledigt werden)

- Auf verwendete globale Open XR Umgebung achten
  - Siehe nächste Folie

OpenXR Runtime

- Entsprechend für OpenXR Toolkit (Default oder Oculus Software) oder SteamVR (Advanced Settings) einstellen

- Controller sollten nicht funktionieren falls eine inkorrekte Umgebung oder Interaction Profile ausgewählt wurde (Einstellung in Oculus Software Settings / *SteamVR -> Advanced Settings*, Interaction Profile in *Project Settings -> XR Plug-in Management -> Open XR*)

Bewegungstypen in VR / Locomotion

- Wenn das *XR Origin (VR)* gewählt wurde, ist zusätzlich das *XR Interaction Manager GameObject* erzeugt worden

- Diesem muss nun eine *Input Action Manager Component* hinzugefügt werden (Deprecated in neuen Versionen)

- In dieser Component werden nun als Action Assets die XRI Default Input Actions zugewiesen, die Teil der Starter Assets sind

Bewegungstypen in VR / Locomotion

- Um Bewegungen grundlegend an das *GameObject* zu übergeben, dass den Spieler darstellt, wird ein *Locomotion System (Action-Based)* benötigt, dass ebenfalls unter ***GameObject -> XR*** zu finden ist (Umbenannter XR-Origin muss zugewiesen werden)

- Nun muss noch eine begehbare Fläche definiert werden, um z.B. Teleportation auf dieser zu erlauben. Dazu wird an das gewünschte GameObject (z.B. Plane) die *Teleportation Area* oder *Teleportation Anchor* Skript *Component* angehängt

Bewegungstypen in VR / Locomotion

- Die Zuweisung der Interactions ist im *XR Origin* im *Left* oder *Right Controller GameObject* einstellbar

- Ob ein Bereich anvisiert wird auf den teleportiert werden kann, wird durch Laser an den Controllern angezeigt. (Rot = Kein Teleport / Weiß = Teleport möglich) Die Farben könne in den Controller *GameObjects* eingestellt werden

- Machen Sie sich am besten etwas mit den Settings in den XR Objekten vertraut. Hier kann viel eingestellt werden (auch per Skript)

Bewegungstypen in VR / Locomotion

- Standard Button für *Teleport* ist der *Grip*
- Standard Button für *Snap Turn* sind die *Thumbsticks*
- *Continuous Movement* (Motion Sickness!) kann optional als Skripte (*Continuous Move/Turn Provider*) an das *Locomotion System* angehängt werden (Eine Hand muss in jeder Skript Component deaktiviert werden)
- Die Sample Presets haben ebenfalls einige Einstellungen zu Dreh- & Bewegungsgeschwindigkeit etc. der Skripte

Interaktionen in VR / Grabbing

- Neben Kollisionsdetektion mit den Controllern kann eine sehr einfache Greif-Interaktion nützlich sein

- Dazu muss einem GameObject lediglich das *XR Grab Interactable* Skript hinzugefügt werden. Dies fügt auch automatisch einen *Collider* und *Rigidbody* hinzu

- Um die Interaktion flüssiger zu gestalten kann im *Rigidbody* die *Collision Detection* auf *Continuous Dynamic* und im *Grab Interactable* Skript *Smooth Position* und *Smooth Rotation* gewählt werden (Ist jedoch Performance-intensiver)

Interaktionen in VR / Grabbing

- Die *Movement Type Property* legt fest wie sich das Objekt verhält, während es sich im Griff des Spielers befindet

- Hier kann entweder *Instantaneous* gewählt werden, was das Objekt von Frame zu Frame teleportiert, *Kinematic* bei dem es sich weiter an physikalische Gegebenheiten (Beschleunigung) hält, oder *Velocity Tracking* bei dem zusätzlich Kollisionen mit anderen Objekten wie Wänden weiter beachtet werden

- Für mehr Info: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html

Pico Neo Setup / Import

- Um mit der Pico Neo 3 in Unity zu arbeiten wird das Pico Unity Integration SDK benötigt

- Nach erfolgreichem Download einfach als lokales Package mit *Window -> Package Manager* über das Plus-Symbol importieren

- Das folgende Pop-Up erlaubt den Wechsel zu einem Android Build und das Festlegen einer ID für den Pico Store (Optional / Benötigt Pico Account)

https://developer-global.pico-interactive.com/sdk/

Pico Neo Setup / Voreinstellungen

- Um spezifisch für die Pico zu entwickeln wird in *Edit -> Project Settings -> XR Plug-In Management* das Pico Plug-In im Android Tab aktiviert

- In *Edit -> Project Settings -> Player* im Android Tab:
  - Minimum API Level: Android 10.0 (API Level 29)
  - Scripting Backend: IL2CPP
  - Target Architectures: ARM64 (ARMv7 abschalten)

- Abschließend muss dem *XR Origin / Rig* GameObject das *PXR_Manager* Skript angefügt werden

Pico Neo Setup / Streaming

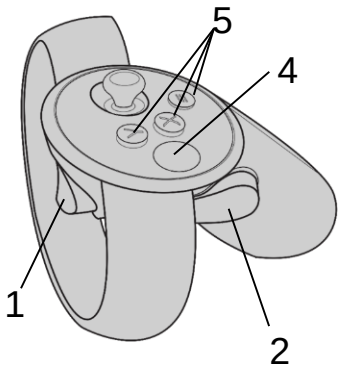
- Durch das Link-Kabel, Link Software & SteamVR wird die Live Preview in Unity genutzt

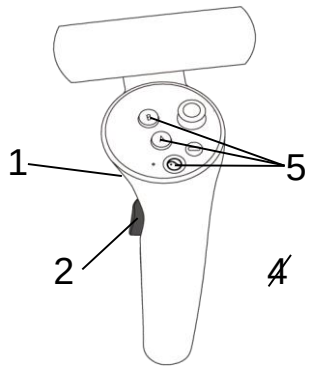
- Dazu muss in *Edit -> Project Settings -> XR Plug-In Management* das OpenXR Plug-In geladen werden
  - Nur für das Debugging nötig. Build ist weiterhin Android!
- Nun in den zugehörigen OpenXR Einstellungen die *Play Mode OpenXR Runtime* auf SteamVR setzen

- Als Interaction Profile wird *Oculus Touch* genutzt
  - Pico hat noch kein eigenes Default Profile

Development Tipps – Button Priority

- Da manche Buttons auf den VR Controllern leichter zu erreichen/bedienen sind, sollten Sie bei der Zuweisung einer Prioritätsliste folgen
  1. Trigger
  2. Grip Button
  3. 3D-Interactables (Knöpfe, Hebel etc. in der 3D Umgebung)
  4. Trackpad (Wenn vorhanden)
  5. Menu Buttons (A, B, X, Y, Start etc.)





- Die Thumbsticks sollten Bewegungsaktionen (Drehen, Bewegen, Teleport) vorbehalten sein

Bildquellen: https://www.pngfind.com/mpng/hiwJwob_oculus-touch-controller-line-png-download-oculus-touch/ ,https://hilfe.r3dt.com/space/KB/1560936577

Development Tipps – Immersion

- Es sollte jederzeit eine Framerate von mindestens 60 bis 90 Frames pro Sekunde gehalten werden

- Vermeiden von regulären zweidimensionalen UIs
  - In die Umgebung verbaute Ansätze vermeiden einen Simulationseffekt
  - Statische Menüs sollten maximal zum Starten/Beenden des Spiels verwendet werden

- Post-Processing Effekte wie Anti-Aliasing, Color-Correction und Bloom können in VR aufdringlich und unrealistisch wirken

Development Tipps – Immersion

- Jegliche Objekte sollten eine Minimaldistanz zur Kamera halten müssen
  - Das Eintauchen der Kamera in eigentlich feste Objekte führt schnell zu einem Verlust jeglicher Immersion
  - Einzige Ausnahme dieser Regelung sind Gegenstände die der Nutzer aufnimmt/hält
- Nutzung von Audiosignalen als Feedback
- Schwarzblende bei Szenenwechsel oder nicht durch den Spieler initiierten Teleport
  - Ein sofortiger Ortswechsel kann zu Immersionsbruch führen

Development Tipps – Immersion

- Trotz der hohen Auflösung moderner HMDs sollten Texte groß gehalten werden, da die Entfernung zum Nutzer (in einer immersiven Umgebung) nicht statisch ist.
  - Die einfachste Lösung für kleinere Texte ist eine Bindung an den Controller durch z.B. Grab-Funktion

- Achten Sie darauf bei Bewegungen möglichst einige statische Objekte darzustellen (z.B. das Cockpit bei einem Flugzeug) um Motion Sickness zu vermeiden
  - Abrupte Bewegungen der Kamera sollten immer vermieden werden, auch bei Bewegungen die der Spieler selbst startet

Tutorials & Online Quellen

- Unity Learn Basics Kurs: https://learn.unity.com/ (VR Development – Sehr lang – 8-10h)

- XR Toolkit Docs: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html

- Viele gute Einsteiger Tutorials auf Youtube (Auch für andere Packages wie Steam-VR oder dem Mockup Plug-In)

Showcase