PLOOM

PLAN YOUR ROOM

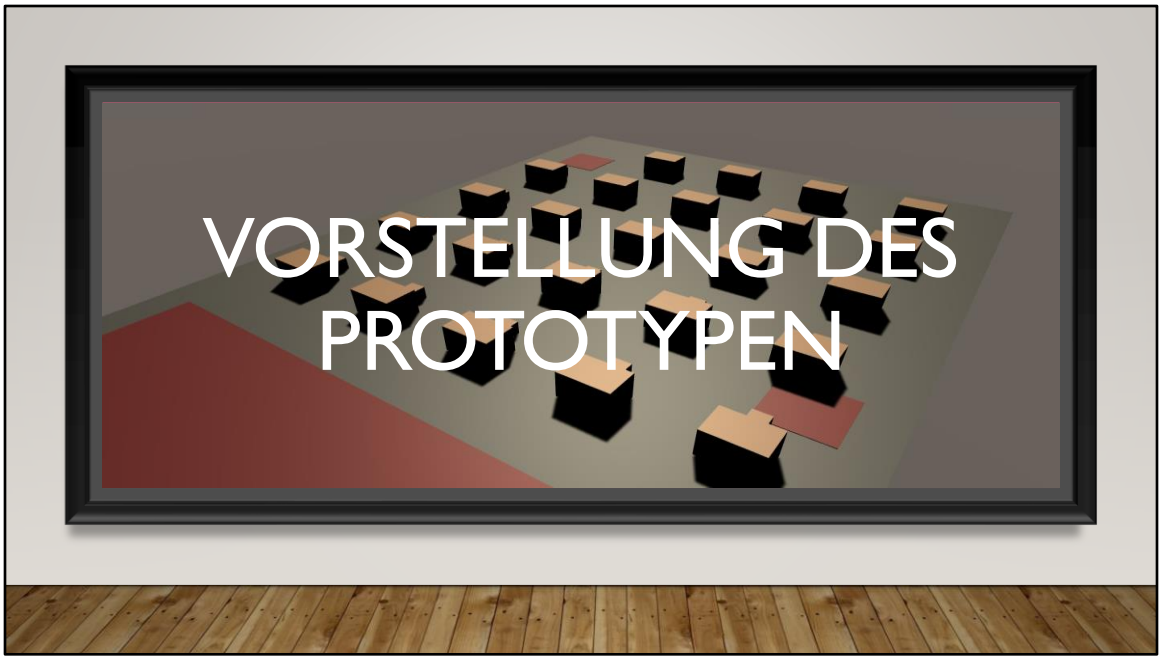
DER MOBILE RAUMPLANER

ENTWICKLUNGSPROJEKT TH KÖLN

RISIKEN

Risikoereignis	Risikoart	Mögliche Ursache
Eingabe der Daten nicht möglich	Architekturell	Seitens des Nutzers könnte der Fehler auftreten, dass die Eingabe der Daten (Länge, Breite, Abstand) nicht zulässig ist.
Fehlende Funktionalitäten	Architekturell	Die Abdeckung aller möglichen Funktionalitäten des Systems kann in vorgegebener Zeit nicht garantiert werden.
Anwendungsfälle sind unklar kommuniziert	Kommunikativ	Durch Darstellung jeglicher Objekte in Form eines sog. „Cubes“, also eines Würfels, könnte dies Unklarheit zwischen Absicht des Entwicklers und dem Nutzer erzeugen.
Applikation ist inkompatibel mit bestimmten Geräten	Technisch	Durch die Entwicklung des Systems für ausgewählte Endgeräte könnte das Risiko bestehen, dass die Applikation für Nutzer mit anderen Geräte unzugänglich ist.
Ansichten in 2D & 3D sind nicht nutzerfreundlich	Architekturell	Umsetzung der Ansichten könnten grob und technisch gestaltet sein.

Hier werden noch einmal Risiken aufgelistet und auf einzelne Risikoarten aufgeteilt, um dem Feedback des letzten Audits gerecht zu werden.



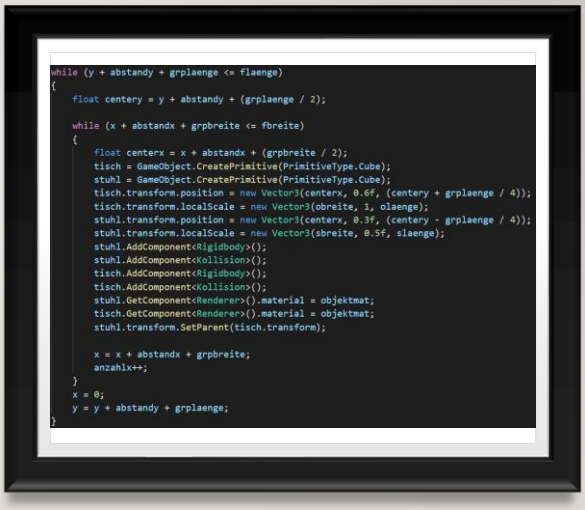
Der Prototyp ist als erster Release in unserem GitHub zu finden (https://github.com/Spirit344/EPWS2020AnspachHeynckes/releases/tag/100121). Der momentane Prototyp sollte auf Windows, MacOS und Linux laufen, allerdings wurde dieser bisher nur auf einem Windows Gerät getestet.

Einige Quality of Life Funktionen sind in dem Prototypen noch nicht beinhaltet. Dies liegt hauptsächlich daran, dass dieser Prototyp erst einmal für PC Plattformen generiert wurde, um ohne einen Emulator die Grundfunktionen und Algorithmen zu testen. Das geplante Endprodukt soll aber hauptsächlich auf mobilen Endgeräten genutzt werden, was einige Design- und Codeänderungen voraussetzt.

Ein gutes Beispiel hierzu ist die momentane Steuerung der einzelnen Hindernisse, die hinzugefügt werden können. Durch Links-Klick Drag kann ein Hindernis frei bewegt werden und via Rechts-Klick Drag skaliert werden. Die einzelnen Wischbewegungen müssten auf einem mobilen Endgerät natürlich anders umgesetzt werden, z.B. skalieren via Auseinander ziehen von zwei Fingern. Dementsprechend ist die momentane Steuerung noch nicht optimiert worden, um dort unnötige Arbeitsressourcen einzusparen.

STÜHLE HINZUFÜGEN

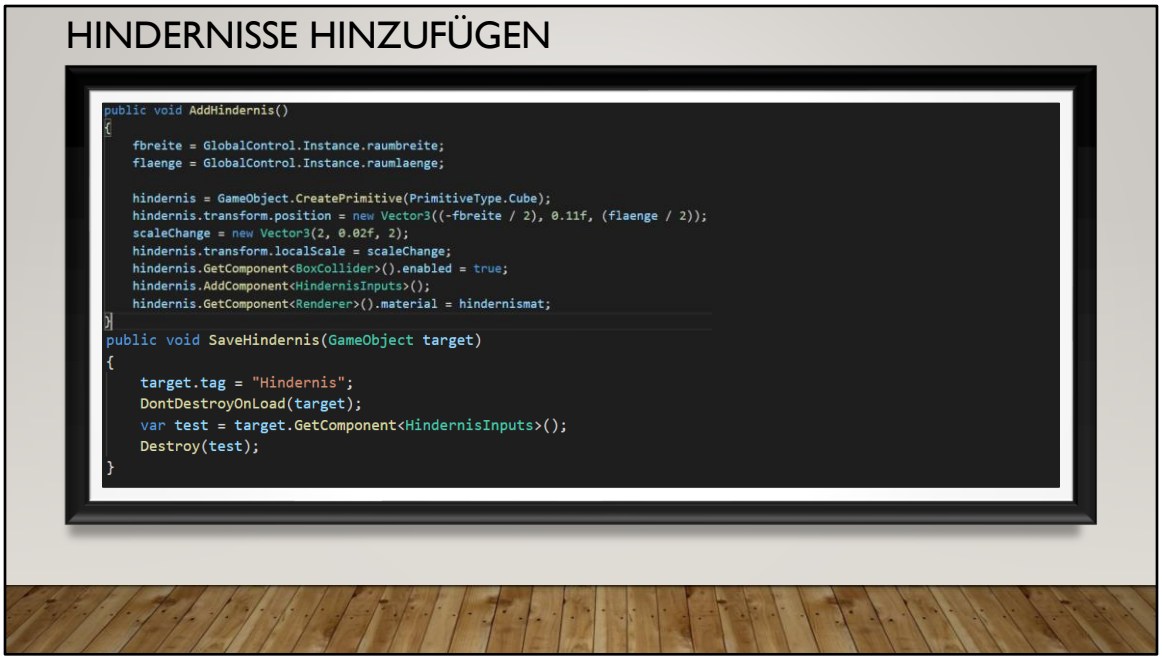
Einzelne Objekte in einem Raum hinzuzufügen ist häufig ausreichend um einen Raum planen zu können. Was ist wenn man aber auch ein weiteres Objekt wie einen Stuhl hinzufügen möchte? Wie fügt man also Objektgruppen ein, wie eine Tischgruppe, bestehend aus Tisch und Stuhl?



```
while (y + abstandy + grplaenge <= flaenge)
{
    float centery = y + abstandy + (grplaenge / 2);

    while (x + abstandx + grpbreite <= fbreite)
    {
        float centerx = x + abstandx + (grpbreite / 2);
        tisch = GameObject.CreatePrimitive(PrimitiveType.Cube);
        stuhl = GameObject.CreatePrimitive(PrimitiveType.Cube);
        tisch.transform.position = new Vector3(centerx, 0.6f, (centery + grplaenge / 4));
        tisch.transform.localScale = new Vector3(obreite, 1, olaenge);
        stuhl.transform.position = new Vector3(centerx, 0.3f, (centery - grplaenge / 4));
        stuhl.transform.localScale = new Vector3(sbreite, 0.5f, slaenge);
        stuhl.AddComponent<Rigidbody>();
        stuhl.AddComponent<Kollision>();
        tisch.AddComponent<Rigidbody>();
        tisch.AddComponent<Kollision>();
        stuhl.GetComponent<Renderer>().material = objektmat;
        tisch.GetComponent<Renderer>().material = objektmat;
        stuhl.transform.SetParent(tisch.transform);

        x = x + abstandx + grpbreite;
        anzahlx++;
    }
    x = 0;
    y = y + abstandy + grplaenge;
}
```

Um einen Stuhl hinzufügen zu können, muss der Algorithmus noch einmal iteriert werden. Der Benutzer hat jetzt die Auswahl einen Stuhl hinzuzufügen in dem er Maße für einen Stuhl in der entsprechenden Szene eingibt. Wenn er dies getan hat wird automatisch diese modifizierte While-Schleife ausgeführt. Wie auch schon in dem einfachen Algorithmus wird erstmal der Mindestabstand in einer separaten Funktion berechnet, um diesen bei der Platzierung der Elemente nutzen zu können. Jetzt werden allerdings zwei Objekte erstellt. Ein Objekt für den Tisch und einen für den Stuhl. Beide werden entsprechend auf die optimale Position im Raum transformiert und dann wie vom Nutzer eingegeben skaliert. Des weiteren wird jedem Objekt ein Rigidbody und eine Kollisions-Komponente hinzugefügt, um mit vorher erstellten Hindernisobjekten interagieren zu können. Jedem Objekt wird auch ein Material zugewiesen, damit es in einem fertigen Build korrekt gerendert werden kann.

HINDERNISSE HINZUFÜGEN



```
public void AddHindernis()
{
    fbreite = GlobalControl.Instance.raumbreite;
    flaenge = GlobalControl.Instance.raumlaenge;

    hindernis = GameObject.CreatePrimitive(PrimitiveType.Cube);
    hindernis.transform.position = new Vector3((-fbreite / 2), 0.11f, (flaenge / 2));
    scaleChange = new Vector3(2, 0.02f, 2);
    hindernis.transform.localScale = scaleChange;
    hindernis.GetComponent<BoxCollider>().enabled = true;
    hindernis.AddComponent<HindernisInputs>();
    hindernis.GetComponent<Renderer>().material = hindernismat;
}
public void SaveHindernis(GameObject target)
{
    target.tag = "Hindernis";
    DontDestroyOnLoad(target);
    var test = target.GetComponent<HindernisInputs>();
    Destroy(test);
}
```

Eine zentrale Funktion unserer Anwendung ist das Hinzufügen von Hindernissen. Über einen Button in der entsprechenden Szene kann ein Hindernis hinzugefügt werden, was einen kleinen Würfel neben die Raumfläche setzt. Dieser kann dann später verschoben und skaliert werden um eine Säule oder eine nicht-belegbare-Zone festzulegen.

Dem Hindernis wird also eine Position zugeteilt, die Links von dem vorher festgelegten Raum liegt. Dann wird das Hindernis zu einer 2x2 großen Fläche skaliert und letztendlich werden einzelnen Komponenten dieses Objektes editiert. Die Komponente BoxCollider wird aktiviert, damit das Hindernis als Kollisionsobjekt nutzbar ist. Des Weiteren wird das HindernisInput Skript hinzugefügt, welches für das Speichern der Position, sowie für das Bewegen des Hindernisses zuständig ist. Außerdem benötigt jedes Hindernis noch ein Material, was die Oberfläche und Farbe des Objektes angibt. Ohne ein Material wüsste der Renderer nicht wie er das Objekt rendern soll.

Die Funktion SaveHindernis wird jeweils aufgerufen wenn das Hindernis bewegt oder skaliert wird, um Änderungen der Vektoren sofort abspeichern zu können, für das weitere Nutzen bei der Ergebnisausgabe. Jedem Hindernis wird hier noch zusätzlich

ein Tag hinzugefügt, was für die Kollisionsberechnung später eine zentrale Rolle spielt. Des weiteren wird das Objekt als DontDestroyOnLoad instanziiert, da beim wechseln der einzelnen Szenen sonst das Hindernisobjekt wieder zerstört wird. Danach wird das HindernisInput Skript wieder entfernt, da Hindernisse nicht beweglich sein sollen in anderen Szenen.

HINDERNISSE SKALIEREN

```
GameObject GetClickedObject(out RaycastHit hit)
{
    GameObject target = null;
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray.origin, ray.direction * 10, out hit))
    {
        target = hit.collider.gameObject;
    }

    return target;
}
```

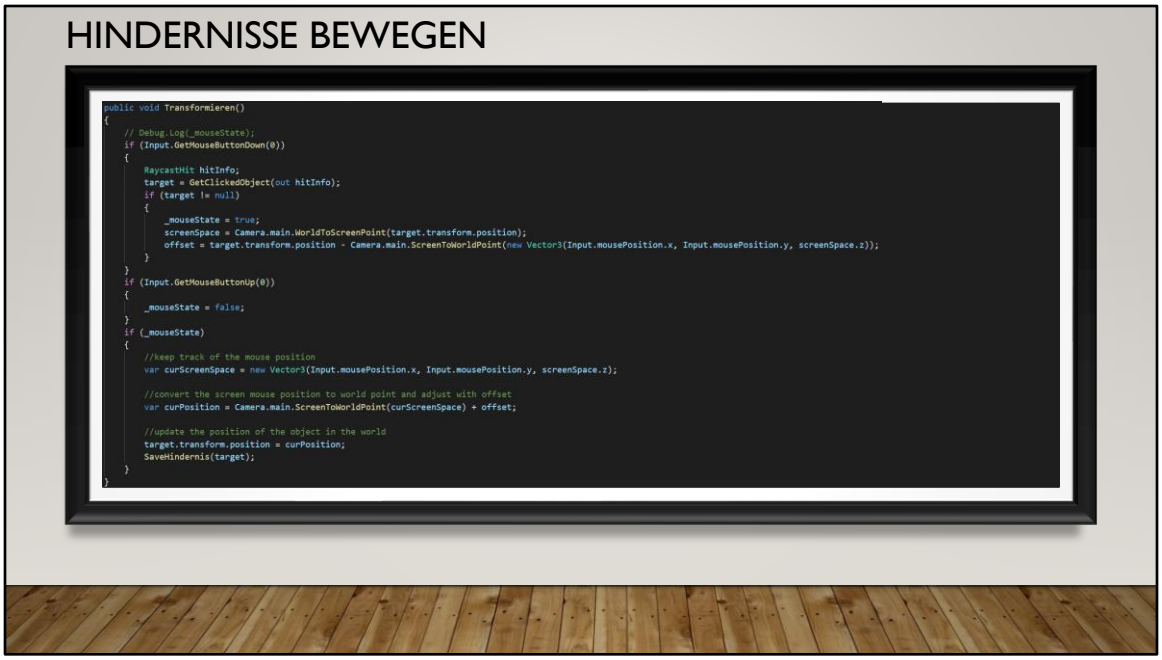
```
public void Skalieren()
{
    if (Input.GetMouseButtonDown(1))
    {
        if (target != null)
        {
            float positionZ = 10.0f;
            Vector3 position = new Vector3(Input.mousePosition.x, Input.mousePosit
            startX = position.x;
            startZ = position.z;
            position = Camera.main.ScreenToWorldPoint(position);
            startSizeX = target.transform.localScale.z;
            startSizeZ = target.transform.localScale.x;
        }
    }

    if (Input.GetMouseButton(1))
    {
        Vector3 size = target.transform.localScale;
        size.x = startSizeX + (Input.mousePosition.x - startX) * sizingFactor;
        size.z = startSizeZ + (Input.mousePosition.y - startZ) * sizingFactor;
        target.transform.localScale = size;
        SaveHindernis(target);
    }
}
```

Um ein Hindernis skalieren und bewegen zu können, muss erstmal eines der hinzugefügten Hindernisse ausgewählt werden. Die Funktion GetClickedObject gibt durch einen Raycast an, welches Objekt als Target gespeichert werden soll. So kann man durch einen Mouseclick das Objekt auswählen, was als nächstes editiert werden soll.

Wenn man jetzt Rechts-klickt, kann das Objekt skaliert werden. Die einzelnen Daten des Skalierungsvektors werden über die Mausposition eingetragen. Wichtig dabei ist, dass die Position der Maus auf einem Bildschirm in 2D Ansicht, nicht der Situation der 3D Programmwelt entspricht. Dementsprechend muss die tatsächliche Position via ScreenToWorldPoint übersetzt werden.

Danach wird auch noch einmal die Funktion SaveHindernis benutzt um die Daten des Hindernisses zu speichern für die weitere Bearbeitung in der Ergebnisszene.

HINDERNISSE BEWEGEN



```
public void Transformieren()
{
    // Debug.Log(_mouseState);
    if (Input.GetMouseButtonDown(0))
    {
        RaycastHit hitInfo;
        target = GetClickedObject(out hitInfo);
        if (target != null)
        {
            _mouseState = true;
            screenSpace = Camera.main.WorldToScreenPoint(target.transform.position);
            offset = target.transform.position - Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenSpace.z));
        }
    }
    if (Input.GetMouseButtonUp(0))
    {
        _mouseState = false;
    }
    if (_mouseState)
    {
        //keep track of the mouse position
        var curScreenSpace = new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenSpace.z);

        //convert the screen mouse position to world point and adjust with offset
        var curPosition = Camera.main.ScreenToWorldPoint(curScreenSpace) + offset;

        //update the position of the object in the world
        target.transform.position = curPosition;
        SaveHindernis(target);
    }
}
```

Das Transformieren von einzelnen Hindernissen funktioniert beinahe genauso wie das Skalieren. Auch hier wird erstmal überprüft ob ein Objekt auch angeklickt wurde und dieses wird dann, je nach MausPosition bewegt. Auch hier wird durch Screen und World View unterschieden, und die einzelnen Vektoren werden dann zu einem offset subtrahiert. Das offset wird benutzt um die Mausposition in die 3D Position des Objektes zu konvertieren. Letztendlich wird dann auch hier das Ergebnis gespeichert.

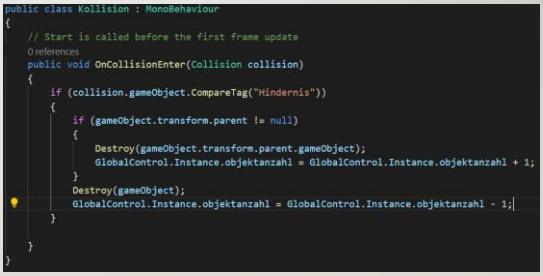
Da man in dieser Szene auch die Kamera von 2D orthographisch auf 3D wechseln kann, ist die Steuerung im 3D Modus natürlich ganz anders, da dementsprechend die genaue Übersetzung von Screen zu World Space anders funktioniert. In unserem Prototypen haben wir deswegen zusätzlich noch eine Skalierung im 3D Raum umgesetzt. Da, wie auf vorigen Folien bereits erwähnt, aber eine Touchsteuerung umgesetzt werden soll, haben wir uns entschieden keine weiteren Ressourcen, auf das Transformieren im 3D Raum via Computermaus, zu allokieren.

Eine Zusatzfunktion für die 3D Ansicht könnte so ausfallen, dass man eventuell per Swipe die Kamera um den Raum rotieren kann, anstatt das Hindernisse dort ebenfalls transformierbar sind.

KOLLISION DETEKTION

Es gibt mehrere Möglichkeiten, um auszurechnen ob eines unserer Tischobjekte nun mit einer Hinderniszone kollidiert.

1. Mathematische Berechnung der Belegten Zonen
2. Kollision Detektion vom Hindernisobjekt, mit den zu platzierenden Objekten.



```
public class Kollision : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    public void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Hindernis"))
        {
            if (gameObject.transform.parent != null)
            {
                Destroy(gameObject.transform.parent.gameObject);
                GlobalControl.Instance.objektanzahl = GlobalControl.Instance.objektanzahl + 1;
            }
            Destroy(gameObject);
            GlobalControl.Instance.objektanzahl = GlobalControl.Instance.objektanzahl - 1;
        }

    }
}
```

Verschieden Möglichkeiten kommen in Betracht, um nun auszurechnen, wo genau Objekte platziert werden dürfen und wo nicht. Die Hindernisse sind bereits angelegt und der Nutzer hat die Maße der Objekte eingegeben. Da alle Objekte die eingegeben werden rechteckig sind, sowie auch das Hindernis, ist es nun möglich einzelne Flächen auszurechnen und zu überprüfen ob diese mit Anderen überschneiden. Während diese Lösung mathematisch elegant ist und in jeder Programmierumgebung umgesetzt werden kann, haben wir uns trotzdem für eine weitere Möglichkeit entschieden. Unity als mächtige Engine bietet hier viele Möglichkeiten um eine Kollision zweier Objekte zu berechnen.

Wenn also eine dieser internen Methoden benutzt werden kann, können auch Hindernisse mit runder oder vollständig individueller Form angelegt und verarbeitet werden. Dazu benutzen wir also in Unserem Prototypen die Funktion OnCollisionEnter. Bei einer Kollision von zwei Objekten wird nun der Tag der verschiedenen Objekte verglichen. In der SaveHindernis Funktion haben wir bereits jedes Hindernis mit dem Tag „Hindernis“ abgespeichert. Wenn nun also ein Objekt nicht den „Hindernis“ Tag besitzt, wird es direkt nach der Platzierung zerstört. Die Anzahl aller Objekte im Raum wird dann um 1 verringert, um trotzdem noch die richtige Objektanzahl im gesamten Raum ausgeben zu können.

Wenn einzelne Stühle oder nur der Tisch einer Objektgruppe von einem Hindernis geblockt wird, muss dementsprechend die ganze Tischgruppe gelöscht werden, damit man nicht einzelne Stühle und Tische im Raum stehen hat. Da der Stuhl aber jeweils das Child-Objekt eines Tisches ist, wird direkt die ganze Gruppe gelöscht und so verbleiben keine einzelnen Objekte.

WEITERARBEIT

Funktionserweiterungen:

- „Neue Berechnung starten“ Button (Zerstörung von DontDestroy Objekten, z.B GlobalControl, Hindernisse)
- Hindernisse löschen via Button/MausInput
- 3D Kamera Distanz und Position dynamisch anpassen
- 3D Kamera per MouseInput bewegen anstatt der festgelegten Rotation
- Vor und Zurück Button, um zwischen Szenen zu wechseln
- Irgendwo klicken, um den Startscreen zu überspringen
- Beim Anlegen der Hindernisse die Möglichkeit bieten die Objekte zueinander snappen zu lassen
- Inkrement der Hindernisbewegung und -skalierung anzeigen in Koordinatenfeldern

Ui Anpassung:

- Änderungen vornehmen für einen Mobile Client
- Titelscreen designen
- Logo designen
- Szenen Layouts erstellen
- Designs für Buttons und andere Elemente erstellen
- Modelle, Licht und Material anpassen
- Alles in Unity umsetzen

Je nach Feedback wird die Weiterarbeit angepasst und eventuell nochmal auf bestimmte Bereiche fokussiert.

VIELEN DANK

- Weitere Informationen, sind in unserem Wiki zu finden
  https://github.com/Spirit344/EPWS2020AnspachHeynckes/wiki
- Prototype Release Windows Build zum Download
  https://github.com/Spirit344/EPWS2020AnspachHeynckes/wiki/Audit-3PDF