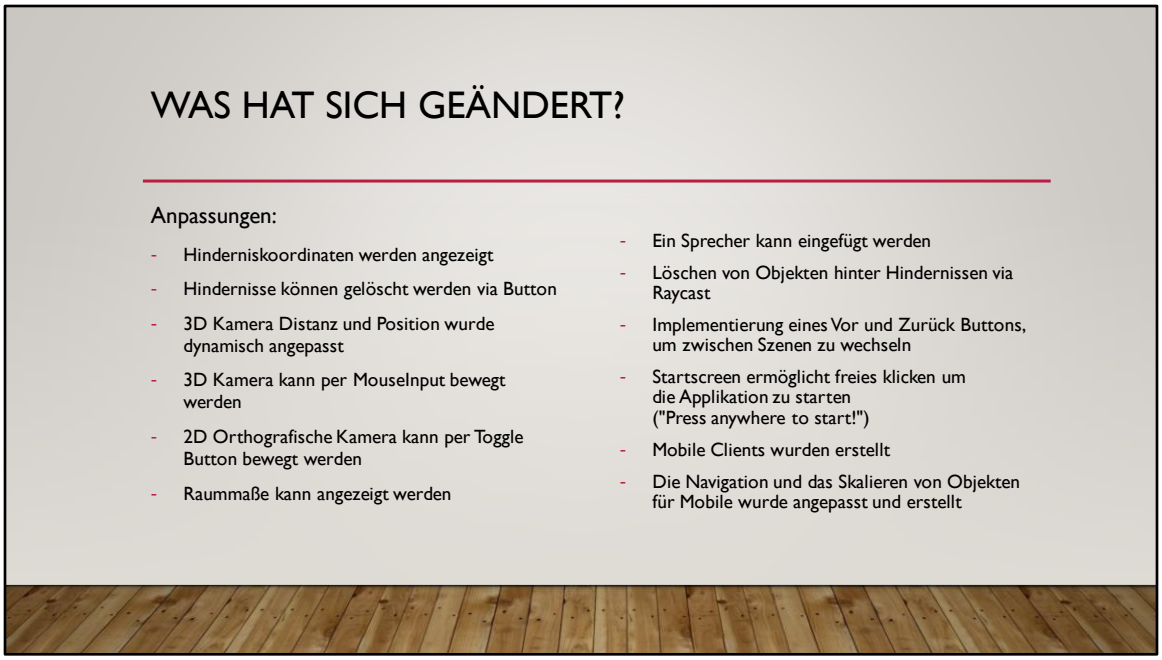
PLOOM

PLAN YOUR ROOM

DER MOBILE RAUMPLANER

ENTWICKLUNGSPROJEKT TH KÖLN



Diese Funktionen und viel weitere, an denen im Projekt gearbeitet wurde, sind in unserer Arbeitsmatrix gekennzeichnet. Die Arbeitsmatrix ist in unserem GitHub-Repository zu finden.

https://github.com/Spirit344/EPWS2020AnspachHeynckes/blob/main/EPWS2020AnspachHeynckes_Arbeitsmatrix.xlsx

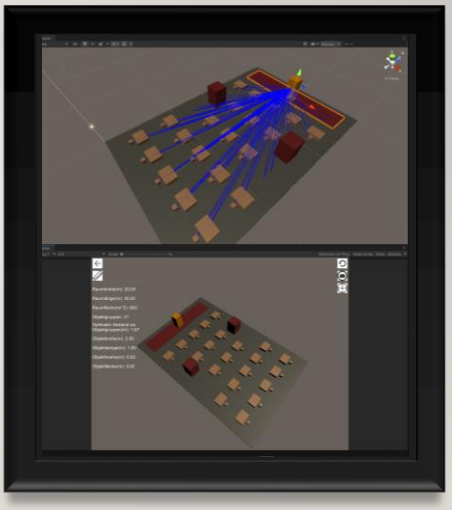
Sie finden Prototypen für die verschiedenen Betriebssystem in unserem GitHub

https://github.com/Spirit344/EPWS2020AnspachHeynckes/releases

RAYCAST

Die Funktionalität einen Sprecher in den von Ploom zu organisierenden Raum, hinzuzufügen, erweitert die Software um einen wichtigen Anwendungsfall.

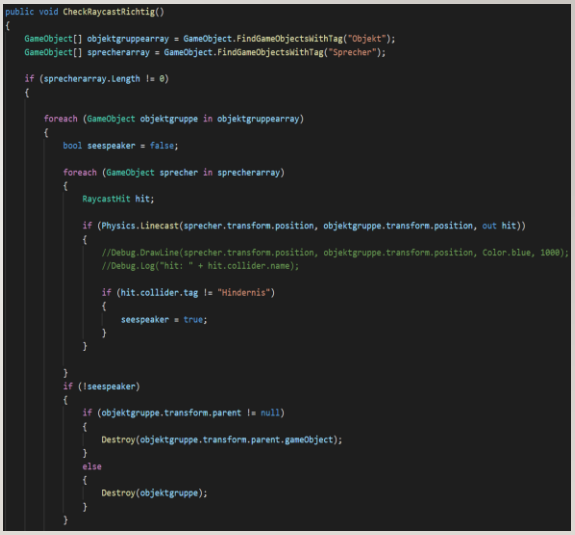
Wie entfernt man aber am besten die einzelnen Objekte, die der Sprecher nicht sehen kann, falls ein Hindernis die Sicht blockiert?



Ploom bietet nun die Möglichkeit einen Sprecher via Button hinzuzufügen. Des Weiteren können jetzt auch Höhenwerte angegeben werden, nicht nur für den Sprecher, sondern auch für alle anderen Objekte, wie Hindernisse und Tische. Dies bildet eine völlig neue Problemstellung, da nun auch im dreidimensionalen Raum beobachtet werden muss, welche Hindernisse eventuell die Sicht blockieren. Die beste Methode zur Detektion, ob die Sicht zwischen Objekten und dem Sprecher gebrochen wird, ist ein sogenannter Raycast.

Der Raycast besitzt einen Startpunkt und eine Richtung. Der Startpunkt, ist in diesem Falle der Sprecher und die Richtung, ist der Vektor, an dem das Objekt (z.B. Tisch und Stuhl) steht. Es wird dann ein Strahl der Richtung entlang geschossen, und wenn ein Hindernis getroffen wird, bevor das Objekt erreicht wurde, wird das Objekt aus der Szene entfernt. Die Alternative für einen Raycast wäre ein Linecast. Dieser hat einen Startpunkt und einen festen Endpunkt, anstatt einer Richtung. Beide Optionen würden in diesem Zusammenhang funktionieren.

RAYCAST CODE



```
public void CheckRaycastRichtig()
{
    GameObject[] objektgruppearray = GameObject.FindGameObjectsWithTag("Objekt");
    GameObject[] sprecherarray = GameObject.FindGameObjectsWithTag("Sprecher");

    if (sprecherarray.Length != 0)
    {

        foreach (GameObject objektgruppe in objektgruppearray)
        {
            bool seespeaker = false;

            foreach (GameObject sprecher in sprecherarray)
            {
                RaycastHit hit;

                if (Physics.Linecast(sprecher.transform.position, objektgruppe.transform.position, out hit))
                {
                    //Debug.DrawLine(sprecher.transform.position, objektgruppe.transform.position, Color.blue, 1000);
                    //Debug.Log("hit: " + hit.collider.name);

                    if (hit.collider.tag != "Hindernis")
                    {
                        seespeaker = true;
                    }
                }

            }
            if (!seespeaker)
            {
                if (objektgruppe.transform.parent != null)
                {
                    Destroy(objektgruppe.transform.parent.gameObject);
                }
                else
                {
                    Destroy(objektgruppe);
                }
            }
```

Die Umsetzung des Raycasts, hat sich ursprünglich als extrem schwierig und aufwendig entpuppt. Sprecher und Hindernis Objekte befinden sich in einer „DontDestroyOnLoad“ Szene, um diese aus der vorherigen Szene übertragen zu können. Die einzelnen zu platzierenden Objekte werden aber in der Ergebnis Szene generiert und sind somit nicht in der gleichen Szene enthalten.

Um also die einzelnen Objekte mit einander vergleichen zu können, muss erst einmal ein Array erstellt werden, was in allen Szenen suchen kann und dann die entsprechenden Spielobjekte beinhaltet. Mit der Funktion „FindGameObjectsWithTag“ kann ein Array unkompliziert aufgefüllt werden, wenn alle Objekte vorher mit dem entsprechenden Tag versehen wurden.

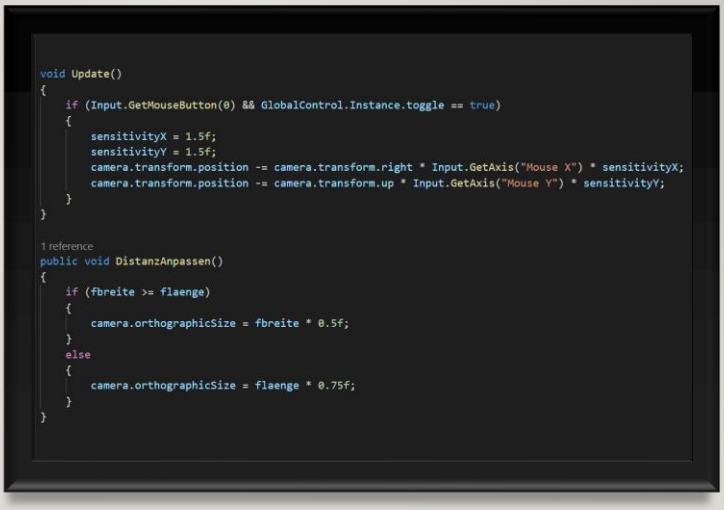
Jetzt werden einige Checks benötigt, um die Funktion in verschiedenen Anwendungsfällen benutzbar zu machen. Erstmal wird überprüft, ob es überhaupt einen Sprecher im vorher angefertigten Array gibt. Wenn einer oder mehrere Sprecher existieren wird dann mit der doppelten ForEach-Schleife fortgefahren.

In dem „objektgruppearray“ sind Tische und Stühle bereits abgespeichert. Jetzt muss geprüft werden, ob jeder Tisch und jeder Stuhl einen Sichtkontakt mit mindestens

einem Sprecher aufbauen kann. Daher geht die innere Schleife auch alle Sprecher im „sprecherarray“ durch.

Jetzt wird die Position des Sprechers als Startposition des Raycasts festgelegt. Die Richtung des Raycasts, ist die jeweilige Position des Objektes, wie zum Beispiel der Tisch. Wenn der Raycast nun mit einem Objekt kollidiert, wird erst einmal überprüft, ob das Objekt den Hindernis Tag besitzt. Wenn das nicht der Fall ist, wird eine boolean variable belegt, die für die Berechnung der Anzahl der bestehenden Objekte in einer anderen Funktion benötigt wird. Wenn ein Hindernis getroffen wird, wird das Objekt gelöscht. Wird ein Tisch getroffen, wird auch der Stuhl als „Child-Objekt“ des Tisches gelöscht.

ORTHOGRAPHISCHE-KAMERA



```
void Update()
{
    if (Input.GetMouseButton(0) && GlobalControl.Instance.toggle == true)
    {
        sensitivityX = 1.5f;
        sensitivityY = 1.5f;
        camera.transform.position -= camera.transform.right * Input.GetAxis("Mouse X") * sensitivityX;
        camera.transform.position -= camera.transform.up * Input.GetAxis("Mouse Y") * sensitivityY;
    }
}

1 reference
public void DistanzAnpassen()
{
    if (fbreite >= flaenge)
    {
        camera.orthographicSize = fbreite * 0.5f;
    }
    else
    {
        camera.orthographicSize = flaenge * 0.75f;
    }
}
```

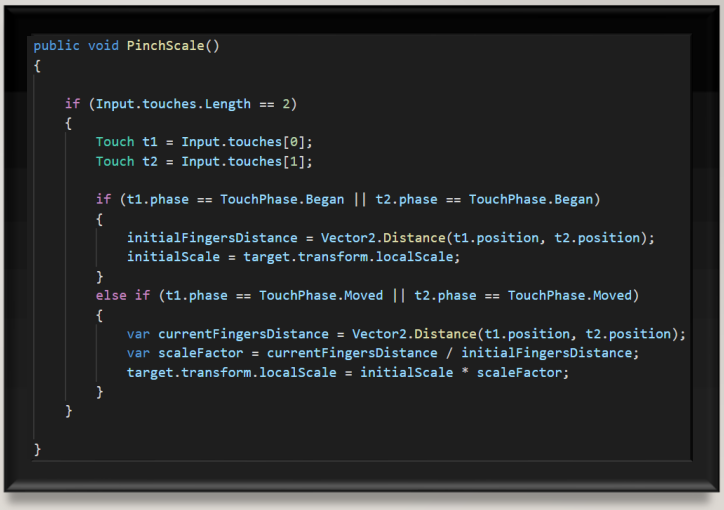
Die Darstellung eines Raumplanes, ist am übersichtlichsten in einer orthographischen Sicht. So werden keine weiteren Schatten und Kanten generiert, und wie auf einem zwei dimensional erstellten Bauplan, kann man die genaue Größe und Position der Objekte erkennen.

Die Orthographische Kamera muss allerdings jeweils an einer anderen Stelle generiert werden, je nachdem wie groß der vorgegebene Raum ist, muss die Kamera ja über dem Mittelpunkt des Raumes schweben. Wenn man die Kamera über den Mittelpunkt des Raumes bewegt hat, muss allerdings noch die Größe des Sichtfeldes angepasst werden. In einem 4/3 Seitenverhältnis ergeben sich dann bestimmte Werte, die in der Methode „DistanzAnpassen“ berechnet werden. Je nachdem wie der Raum geformt ist, muss ein anderer Wert als Größe eingestellt werden. In einem 20x30 Meter Raum liegt die Optimale Kameragröße zum Beispiel bei 25.

Wenn jetzt aber ein Raum angegeben wird, der sehr lang ist, aber gleichzeitig sehr schmal, wie zum Beispiel ein langer Gang, dann passt der ganze Raum nicht auf den Bildschirm. Um aber trotzdem den ganzen Raum betrachten zu können, muss die Möglichkeit geboten sein, den Kamera per Mouse/Fingerdrag zu verschieben. In der Update Funktion wird überprüft, ob der zur Bewegung benötigte Knopf gedrückt

wurde. Wenn ja wird die Kamera zur Mausposition verschoben.

SKALIEREN FÜR MOBILE



```
public void PinchScale()
{

    if (Input.touches.Length == 2)
    {
        Touch t1 = Input.touches[0];
        Touch t2 = Input.touches[1];

        if (t1.phase == TouchPhase.Began || t2.phase == TouchPhase.Began)
        {
            initialFingersDistance = Vector2.Distance(t1.position, t2.position);
            initialScale = target.transform.localScale;
        }
        else if (t1.phase == TouchPhase.Moved || t2.phase == TouchPhase.Moved)
        {
            var currentFingersDistance = Vector2.Distance(t1.position, t2.position);
            var scaleFactor = currentFingersDistance / initialFingersDistance;
            target.transform.localScale = initialScale * scaleFactor;
        }
    }

}
```

Das Skalieren von Hindernissen ist auf dem Windows Client durch das Rechtsklicken eines Objektes möglich. Auf einem mobilen Client muss nun also die Möglichkeit geboten werden, das Objekt anderweitig zu skalieren.

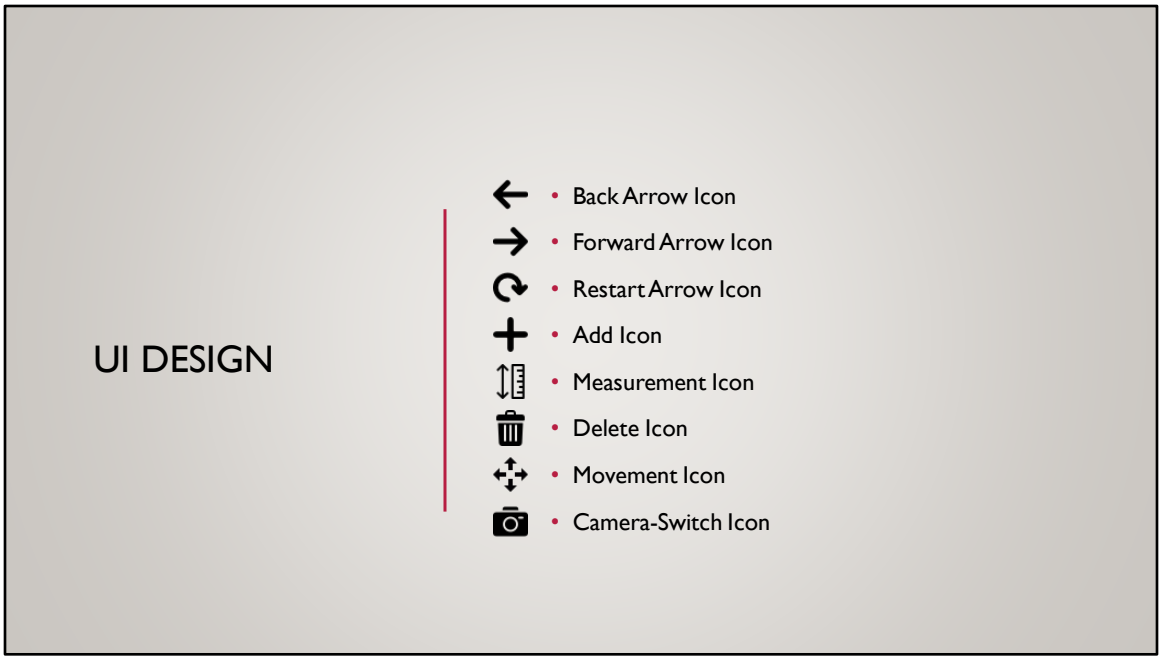
Die Methode „PinchScale“ wird nun dazu benutzt, um den sogenannten Pinch-Zoom für die Skalierung zu benutzen. Bei einem Pinch-Zoom benutzt der Anwender zwei Finger und zieht diese auseinander um das Objekt heran zu zoomen oder in unserem Falle zu skalieren. Dazu prüft die Software, ob der Anwender zwei Finger auf dem Bildschirm hat. Wenn das der fall ist, wird die Distanz zwischen den Begin des Pinch-Zooms und dem Ende gemessen. Und das Objekt wird dann entsprechend skaliert mit „target.transform.localScale“.

UI/UX ANPASSUNGEN

- Layout blautönig und minimalistisch gehalten
- Funktionalitäten für jeden Nutzer schnell erkennbar designed
- Einheitliches, zusammenhängendes Motiv

Um dem Benutzer ein gutes Nutzungserlebnis zu bieten, ist es entscheidend, das Layout der Anwendung entsprechend zu gestalten. Da es bei Ploom um die Organisation von öffentlichen und privaten Räumlichkeiten geht, haben wir uns entschieden, Symbole typischer raumbezogener Objekte einzubauen, also einen Stuhl, einen Tisch usw.

Es ist wichtig, das Design minimalistisch zu halten, um zu verhindern, dass der Benutzer durch irgendwelche Designelemente abgelenkt wird. Die verschiedenen Blautöne, die auch im Icon implementiert sind, stehen für Ruhe und Verantwortung - zwei Aspekte, die als Ziele der User Experience angesehen werden können. Das gesamte Design des Systems ist einfach und leicht verständlich gehalten.



Back Arrow Icon - Ermöglicht Zugriff auf den vorherigen Slide
Forward Arrow Icon - Ermöglicht Zugriff auf den nachfolgenden Slide
Restart Arrow Icon - Neustart der gesamten Raumorganisation
Add Icon - Hindernisse oder Sprecher können hiermit hinzugefügt werden
Measurement Icon - Blendet Maße im Endergebnis ein
Delete Icon - Löscht Hindernis oder Sprecher
Movement Icon - Ermöglicht Bewegung der orthografischen Kamera
Camera-Switch Icon - Ermöglicht Wechsel zwischen orthografischer und 3-Dimensionaler Kamera

FAZIT

- Das Endprodukt von PLOOM weicht nur leicht von der ursprünglichen Vision ab (keine 3D Modelle, wesentlich detailliertere Hindernisdetektion, keine speziellen Raumformen)
- Exponentielle Zunahme verschiedener Funktionen; je größer das System wurde, desto mehr Funktionen kamen hinzu (damit kamen auch viele Bugs)
- Realisierung und Veröffentlichung einer PLOOM Applikation wäre durchaus denkbar mit Funktionserweiterungen
- Die Arbeit an PLOOM hat uns in vielen Bereichen fortgebildet

Das Entwicklungsprojekt Ploom ist für unser Team eine Erfolgsgeschichte. Viele Hürden und Hindernisse wurden während der Entwicklung überwältigt und wir haben extrem viele Dinge gelernt. Nicht nur über die Anwendung von Unity oder das Coden in C#, sondern auch über das Design und den Prozess eines Software Projektes und die Arbeit im Team.

Die Ploom App ist in den meisten Aspekten der ursprünglichen Planung treu geblieben. Der ursprüngliche Fokus, war eher auf das Design und die Methodik der App fokussiert. Zum Beispiel sollten Objekte und Hindernisse eigene 3D Modelle haben, und es sollte die Möglichkeit geboten werden L-förmige und Runde Räume zu erstellen. Durch das sinnvolle Feedback der Mentoren, hat sich das Projekt in diesen Fällen geändert und ist daher tiefer in die Algorithmik der Hindernisdetektion eingegangen.

Über den Zeitraum haben sich mehr und mehr Probleme und Hürden offenbart. Durch das entwickeln neuer Funktionen, die vorher nicht bedacht waren, wurden viele Lücken gefüllt und die Software wurde komplexer und komplexer. Während die Komplexität selber auch eine Hürde dargestellt hat, können wir uns nun die Softwareentwicklung an einem großen Projekt genauer vorstellen.

Die Applikation hat natürlich noch viele Schwächen. Einige Bugs konnten in diesem Zeitraum nicht behoben werden und einige Funktionen fehlen noch, um eine polierte App anbieten zu können. Auch das Pflegen verschiedener Betriebssysteme erweist sich als extrem umständlich. Allerdings kann man sich das Nutzen der Ploom Applikation im Alltag vieler Raumplaner vorstellen, wenn weitere Funktionalitäten hinzugefügt werden.

VIELEN DANK

- Die prozentuale, kollaborative Erarbeitung kann unter folgendem Link abgerufen werden: https://github.com/Spirit344/EPWS2020AnspachHeynckes/blob/main/EPWS2020AnspachHeynckes_Arbeitsmatrix.xlsx
- Weitere Informationen, sind in unserem Wiki zu finden: https://github.com/Spirit344/EPWS2020AnspachHeynckes/wiki
- Verfügbare Prototypen für verschiedene Plattformen, sind hier zu finden: https://github.com/Spirit344/EPWS2020AnspachHeynckes/releases