Bachelor-Projekt “Mixed Reality”

Internationaler Studiengang Medieninformatik,
Hochschule Bremen

Fachsemester: 4. Semester
Zeitraum: 09.03.2020 - 26.07.2020

Betreuer: Prof. Dr. Volker Paelke,
Jendrik Bulk, Andreas Lochwitz

Teilnehmer*innen:
Moumita Ahmad, Annie Berend, Moritz Hodler, Robin Jacobse, Reda Khalife, Bjarne Lehmkuhl, Philipp Moritzer, Joscha Sattler, Tobi Schmitt, Alisa Schumann, Leonard Tuturea

Link zum Dokufilm:
https://www.dropbox.com/s/gx22rao4r3p2uxn/DokufilmSmartfactory.mp4?dl=0

Link zur Doku 3. Semester:
https://docs.google.com/document/d/1Ca-IgeTRptr8FOXyu0pBXXS-wv2hg-fTIf8nNiWEL0g/edit#

Link zum git-Repository:
https://github.com/RolimJ/MixedReality

Inhaltsverzeichnis

Abstract

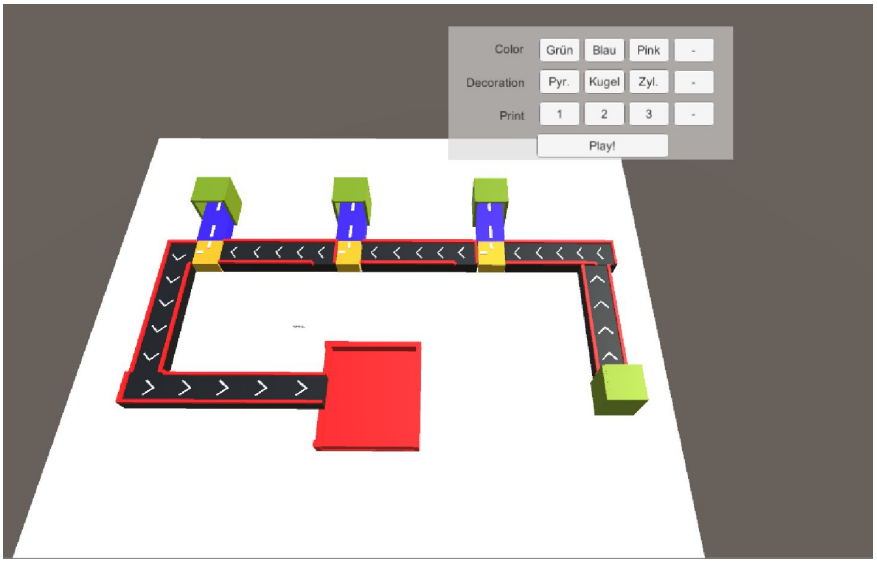
Heutzutage werden nicht mehr Massen des gleichen Produktes hergestellt, sodass der Bedarf an Fabriken, die auf eine schnell wechselnde Nachfrage reagieren, deutlich gewachsen ist. Diesem Problem nimmt sich die vom Bachelorprojekt “Mixed Reality” entwickelte SmartFactory an. Über eine Projektion auf einem Tisch kann über Bewegung oder Sprache mit der Applikation interagiert werden. Die SmartFactory ist eine digitale Simulation von modularen Fabriken. An der Applikation wird die Chance geboten zu schauen, wie eine modulare Fabrik funktionieren kann.

In unserer Simulation wird ein Auto produziert, dessen Arbeitsstationen beliebig kombiniert werden können. Das Programm ermöglicht es eine Fabrik von Grund auf neu aufzubauen, dafür stehen einige Module zur Verfügung, die miteinander kombiniert werden können. Somit kann die Fabrik an verschiedene Räumlichkeiten angepasst werden und je nach Komplexität des Produktes größer oder kleiner gebaut werden.

Planung

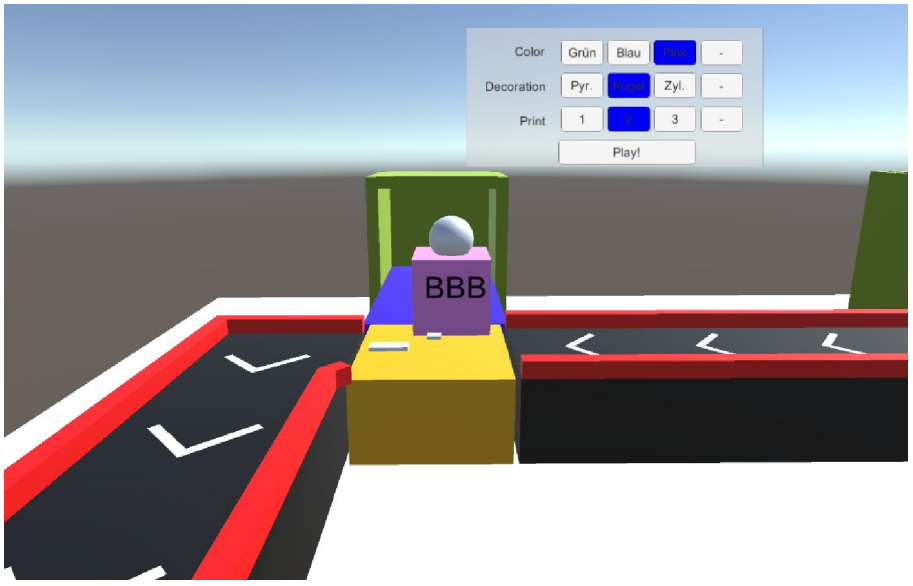
Prototyp

Für den Start in das Projekt wurde für den Beginn des Semesters ein erster Prototyp entwickelt, der als Diskussionsgrundlage für das Projekt dienen sollte.
Der Prototyp umfasste erste Grundbausteine der SmartFactory. Zentral dabei war das Konzept der Workstations, welche jeweils eine bestimmte Funktion bedienen und jeweils parametrisiert werden können. Ein Beispiel aus dem Prototyp ist eine Einfärb-Workstation, die Parameter sind hierbei die Farben Grün, Blau und Pink.



Ein weiterer wichtiger Bestandteil war das Auftrags-Interface in dem Produkte mit verschiedenen Parametern in Auftrag gegeben werden können, die im Anschluss darauf von der SmartFactory produziert werden.

Die Produktionsprozess einen Produktes läuft im Prototyp ab wie folgt: Zunächst werden im Auftrags-Interface die gewünschten Parameter ausgewählt und der Prozess mit dem “Play!”-Button gestartet. Dann wird ein Rohling aus dem Lager, im Screenshot unten links, ausgegeben und durchläuft die 3 Workstations “Color”, “Decoration” und “Print”. Das fertige Produkt fällt dann in das Abglagebecken und die SmartFactory ist wieder einsatzbereit für ein neues Produkt.



Anhand des Prototyps konnten verschiedene konzeptionelle Themen der Smartfactory gut illustriert werden. In der Gruppe wurde dann über diese diskutiert und es wurden Entscheidungen über das Design, die Anforderungen und die Funktionsweise der SmartFactory getroffen.

Softwarearchitektur

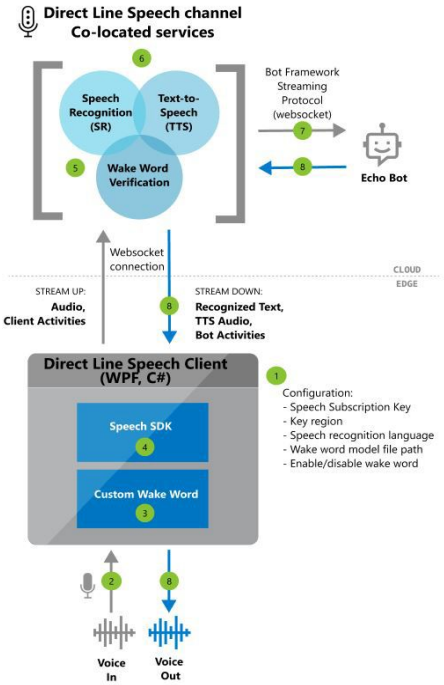
Nach Ausarbeitung der Anforderungen für das Projekt wurde eine Softwarearchitektur erstellt, an der sich bei der Implementierung des Projektes orientiert wurde. Die Architektur geht nach dem Schichtenmodell. Über die Benutzeroberfläche wird über sinnvolle Interfaces auf die Controller und die Models des Projektes zugegriffen. Da die Architektur zu komplex ist, um sie hier zu zeigen, kann sie als VPP-Projekt über folgenden Link abgerufen werden:
https://www.dropbox.com/s/r2z6ycnkg536p1c/SmartFactoryv2.vpp?dl=0

Komponenten

Controller

Damit mehrere unterschiedliche Formen von Controllern im Zusammenspiel an der SmartFactory verwendet werden können, wurde eine abstrakte Klasse SharedDataController erstellt. Von dieser Klasse kann abgeleitet werden, sodass mehrere Controller im gleichen Kontext arbeiten. Zum jetzigen Stand wird dabei jedoch nicht weiter unterschieden als vollkommen eigenständiger Controller und Controller der sich Daten mit allen anderen SharedDataControllern teilt.
Alle Controller sprechen das SmartFactory Interface an und können somit unabhängig von anderen Controllern erweitert/abgeändert werden, sodass auch hier die Erstellung von weiteren von ein unabhängigen Kontexten denkbar ist.

Voice Controller - Spracherkennung



Schon seit Jahren erleichtert die Spracherkennung den Alltag in verschiedenen Bereichen. Tatsache ist, dass eine Menge Aufgaben gelöst werden müssen, bevor eine Spracherkennung gut funktionieren kann: Übersetzen der Sprachanweisungen in Text (Speech to Text, STT), Analysieren des Texts mit Methoden des Natural Language Processing (NLP) wie Named Entity Recognition (NER) zum Erkennen von Personen, Unternehmen, Orten, Zeitangaben und ähnlichem oder Part-of-Speech-Tagging (POS) zum Erkennen der Satzzusammenhänge (Subjekt-Prädikat-Objekt), sowie Erkennen des Benutzerwunsches (Intent) aus den so

gewonnenen Daten. Es gilt nur noch mittels Inferenz die richtigen Fakten aus der Wissensdatenbank zu Filter , in normale Sprache zu überreichen und mittels Sprachausgabe (Text to Speech, TTS) an den Benutzer zu übermitteln.
Durch erhebliche Fortschritte in mehreren dieser Technologie, die insbesondere auf der Grundlage tiefer neuronaler Netze möglich wurden, gibt es heute schon Lösungen. Doch was genau ist denn eine Spracherkennung?

Eine Spracherkennung (auch als Sprache-zu-Text bezeichnet) ermöglicht die Echtzeit- und Batch-Transkription von Audiostreams in Text.
In unserer Anwendung kommt die Speech SDK von Microsoft zum Einsatz die eine Bing-API verwendet. Die bei zusätzlicher Referenztexteingabe auch eine Echtzeitbewertung der Aussprache und Feedback zur Richtigkeit und Flüssigkeit des gesprochenen Texts zurückgibt.

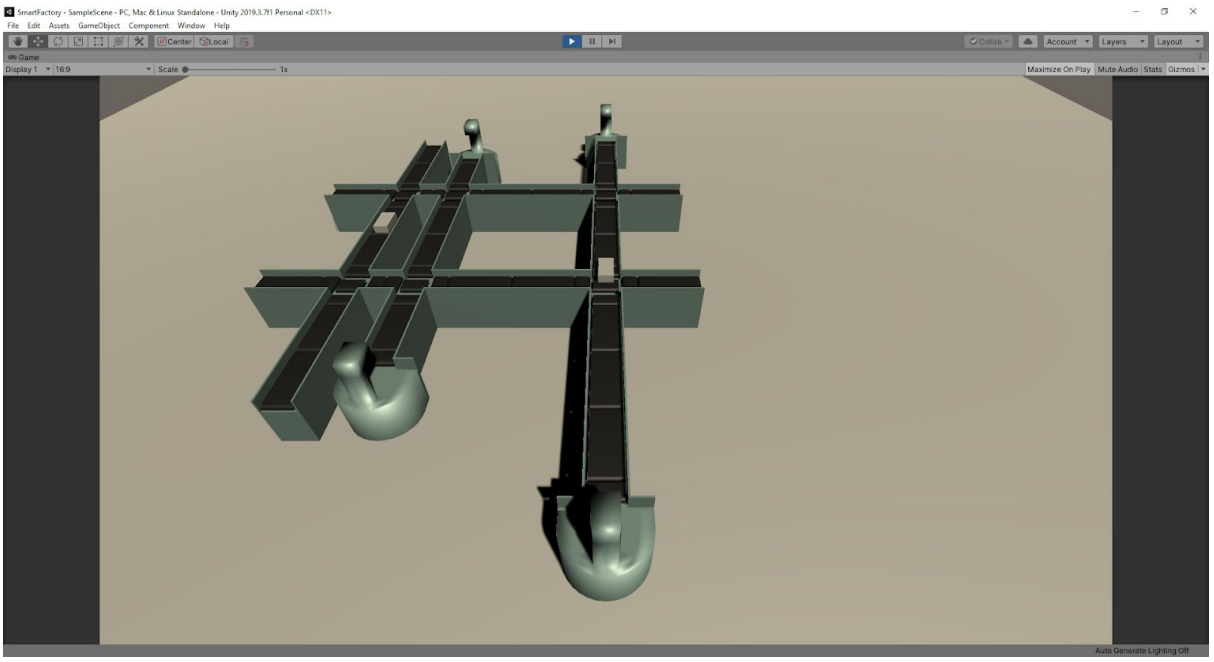
Bei der Spracherkennung werden Sie Audiodatenströme oder lokale Dateien in Echtzeit in Text umgewandelt oder übersetzt, der von der Anwendungen, Tools oder Geräten genutzt oder angezeigt werden kann. In Kombination mit Language Understanding (LUIS) können die Benutzerabsichten aus transkribierter Sprache ableiten und auf Sprachbefehle reagieren. Dabei wird ebenfalls wie bei den restlichen Controllern über die Superklasse SharedDatacontroller zugegriffen um anhand der Sprachoutputs die Aktionen auszuführen.

Produktionsaufträge

In der SmartFactory können Aufträge erstellt werden um bestimmte Items zu produzieren. Um diese Aufträge zu bearbeiten werden freie Carrier aus dem System gesucht und den Aufträgen zugeordnet. Aus den spezifischen Manuals für zu fertigende Produkte werden die benötigten Ressourcen und an zu fahrende Workstations ausgelesen. Die Positionen der Ressourcen und Stationen werden rausgesucht und mithilfe des Pathfindings in den NavigationServices zu geeigneten Instructions umgewandelt die im Carrier gespeichert werden.
Stationen sind nun in der Lage ankommende Carrier aufzunehmen, ihre Instructions auszulesen und auszuführen. Hierzu gehören ua. Aufnehmen/Ablegen eines Items, und Durchführen eines Arbeitsschrittes. Außerdem können alle Stationen über die EventServices eine Richtungsänderung der Förderbänder anfordern um den Carrier an die gewünschte Position zu befördern.

Pathfinding

Für das Pathfinding wurde der A*-Algorithmus nach Amit Patel https://www.redblobgames.com/pathfinding/a-star/introduction.html implementiert. Das hexagonale Gitter wurde hierbei auf quadratische Gitterzellen vereinfacht. Durch die feste Anordnung im Gitter mit fortlaufenden Koordinaten muss kein gesamter Graph für Nachbarschaftsbeziehungen erstellt und bei Veränderung der Fabrik-konstellation aktualisiert werden. Stattdessen werden einzelne Zweige während der Pfadfindung rekursiv über die Ein- und Ausgänge der Stationen und somit ihrer benachbarten Zellen überprüft.



Manager

Für die Verwaltung der unterschiedlichen Objekttypen die in der Szene platziert werden können wurde ein generisches Interface erstellt, welche das Factory Pattern implementiert. Für das Erzeugen der Objekte zieht das Interface hierbei eine Liste von Unity-Prefabs heran. Über diese Liste wurde einfaches Austauschen der zu erzeugten Objekte realisiert.
Von dem Interface abgeleitet wurden anschließend die typspezifischen Managerkassen für Buildings, Carrier und Items erstellt. Über eindeutige Indizes sorgen diese Klassen insbesondere beim Pathfinding und der Persistenz für eindeutige Zuordnung von Objekten und Objektbeziehungen.

Persistenz

Für die Persistenz wurde Serialisierung als Ansatz gewählt. Dieser vereinfacht das Speichern und Laden von komplexeren Klassen und Strukturen. Da die Smart Factory generell ein Editor ist in dem verschiedene Konstellationen erstellt werden, entfällt auch der Nachteil bei der Serialisierung nur schwer Testdaten erzeugen zu können.
Um jedoch nicht alles serialisieren zu müssen wurden eigene Klassen für Speicherdaten erstellt. In diesen wird zB. nur die prefabId des zu erzeugenden Objekts gespeichert, sodass Prefabs auch nachträglich noch ausgetauscht und geladen werden können. Weiterhin werden Koordinaten und Ids von einander in Beziehung stehenden Objekten gespeichert.
Um Beziehungen beim Laden wieder herstellen zu können, wurden Speicherdaten von den verschiedenen Typen Building, Item und Carrier aufgeteilt. Beim Laden werden diese in einer bestimmten Reihenfolge rekonstruiert, sodass zuerst Buildings platziert werden, anschließend Carrier einem Building, und zuletzt Items einem Carrier oder einem Building zugeordnet werden.

Interaktionen

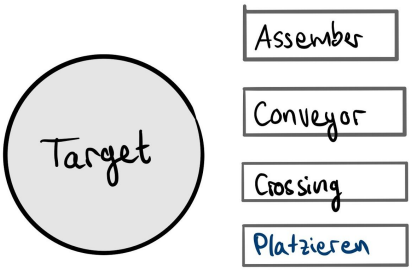
Planung

Zuerst haben wir mögliche Interaktionsweisen und User Interfaces konzipiert.
Dafür haben wir eine Liste an Anforderungen aufgestellt:

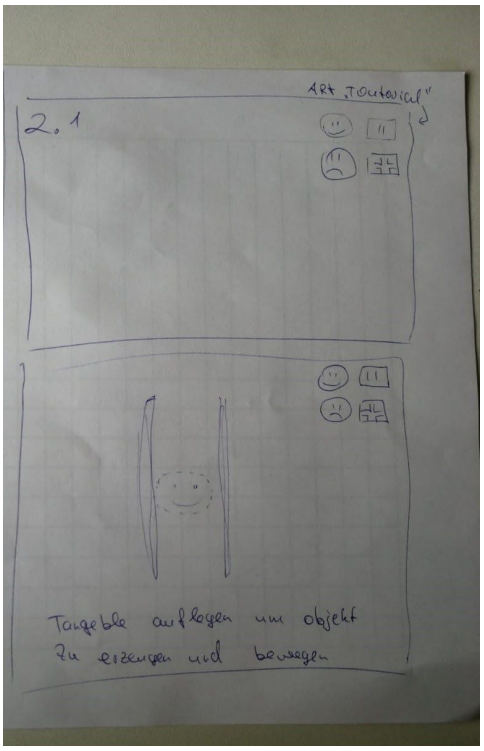
- Objekte müssen positioniert, verschoben und rotiert werden können
- Komponenten (Fabrik Module) sollen Konfiguriert werden können
    - Parameter einstellen
- Systemsteuerungen:
    - Laden, speichern
    - Simulation starten, anhalten
    - Aktueller Zustand speichern
    - Events erzeugen

Anschließend haben wir ein paar Skizzen angefertigt, die eine mögliche Menüführung darstellen können:

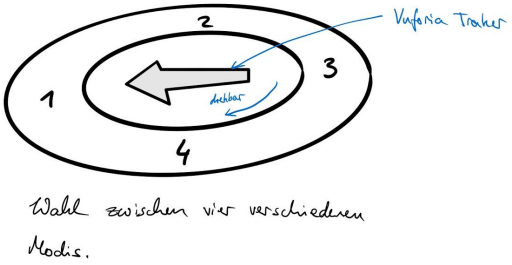
- Menü hat einen festen Platz auf dem Tisch gebunden
- Nur ein Tangible wird zum Auswählen aller Buildings genutzt
- zum Auswählen wird das Tangibles auf Buttons platziert



- weitere Menü Variante ist an Target/Marker



- Weitere Variante: Verschiedenen Tangibles werden Objekte zugewiesen.



- Menü Variante für Parametereinstellungen

Aus diesem Brainstorming haben wir geschlossen, dass es sinnvoll ist, zwei getrennte Menüs zu entwickeln.
Ein Hauptmenü zu Beginn der Anwendung zum Starten und Laden einer Fabrik, als auch für komfort Möglichkeiten wie Auflösung, Grafik, etc. und ein ingame-Menü für Fabrik-bezogene Einstellungen, während der Laufzeit der Anwendung.
Dies sollte unter anderem zur Auswahl des Endprodukts, Starten der Simulation und Beeinflussung der Simulation dienen.

Umsetzung

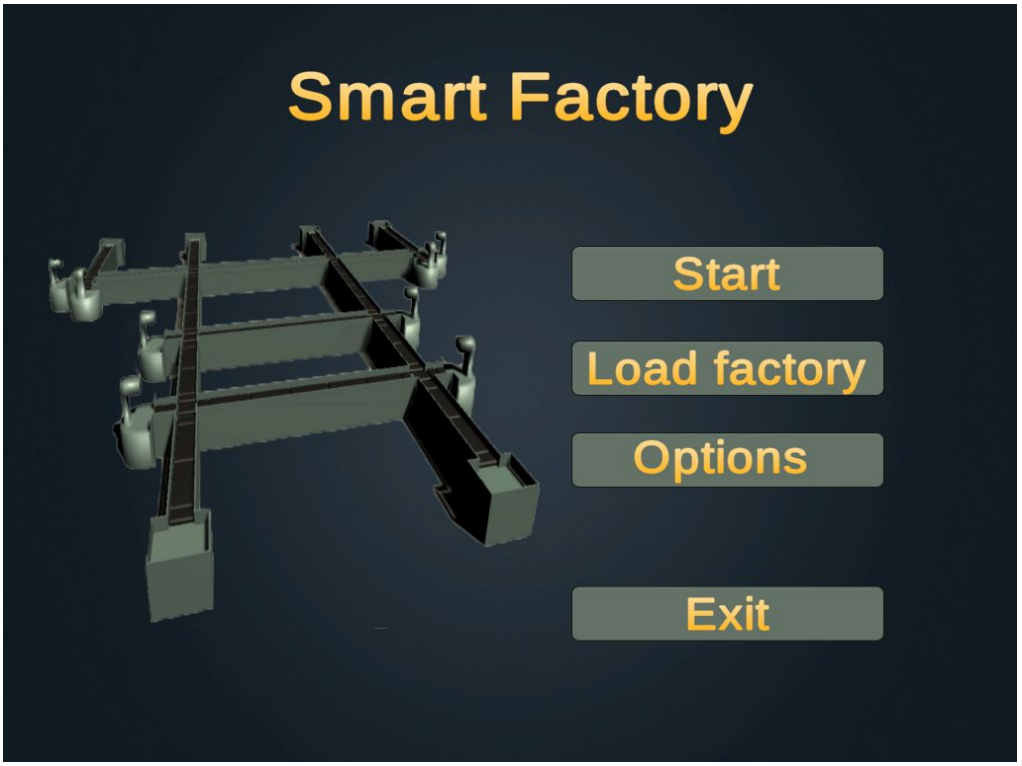
Nachdem eine grobe Planung vorhanden war, fingen wir damit an einzelne Aspekte des User Interfaces auszutesten und ein simples Menü mit allen Basisfunktionen zu erschaffen.

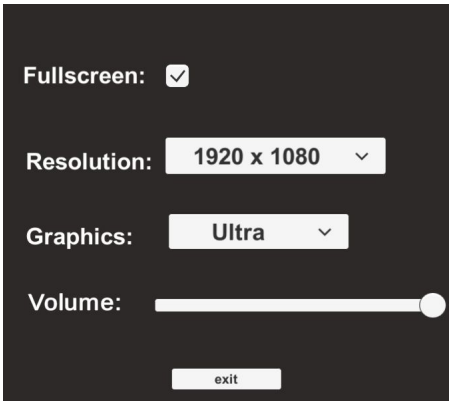
So entwickelten wir kleine Anwendungen um Teilaspekte zu testen, zum Beispiel Möglichkeiten für Parametereinstellungen oder eine Anwendung um den Farbtracker von Vexpot, den wir dieses Semester das erste Mal nutzen, zu testen.

Außerdem entwickelten wir eine erste Version eines User Interfaces mit der Anforderung sowohl auf dem Laptop, als auch auch dem Tisch zu funktionieren. Grund dafür war die lange Home-Office Zeit, bei der viel Zuhause getestet werden musste. Die Bedienbarkeit wurde simpel und selbsterklärend gehalten.

Für das *Startmenü* wollten wir folgende Funktionen umsetzen:

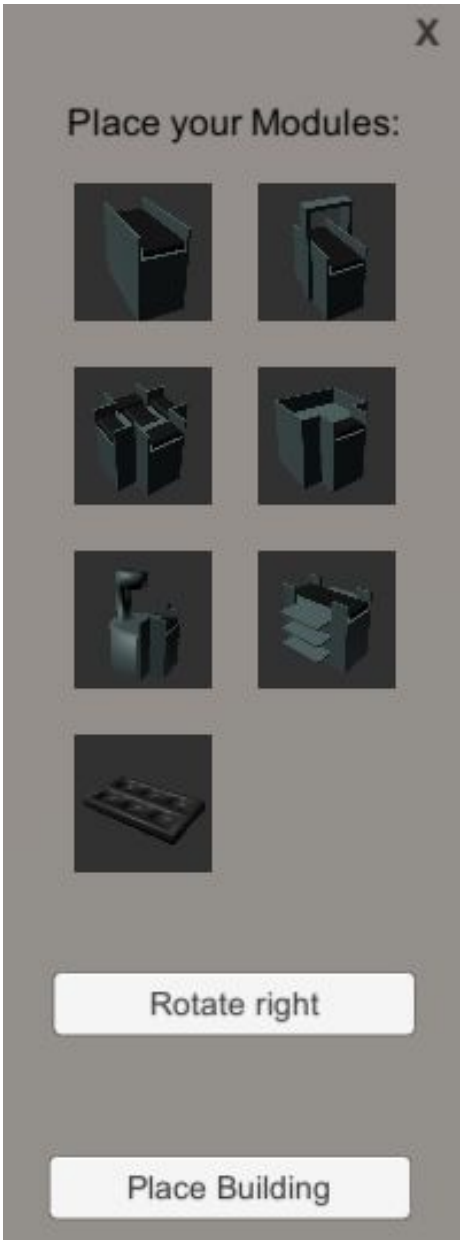
- Neue Fabrik starten
- Bestehende Fabrik laden
- Allgemeine Einstellungen wie Lautstärke und Auflösung anpassen





Dagegen sollte das *Ingame Menü* folgende Funktionen besitzen:

- Aktuellen Stand speichern
- Neue Fabrik aufbauen → Buildings (Fabrik Module) platzieren
- Einstellungen an den Buildings vornehmen, Bsp.: Anzahl der zu bohrenden Löcher festlegen
- Storage Items und Carriers in der Fabrik platzieren
- Simulation starten



Während die Bedienung am Laptop einfach mit der Maus geschehen kann, sollte für den Tische eine intuitive Lösung gefunden werden.
Im Laufe der Zeit haben wir die folgende Möglichkeiten von Tangible User Interfaces ausgetestet:

Realsense

Eine weitere Idee war es die über das auslesen der Entfernung mit der Realsense, verschiedene Aktionen auf dem Tisch auszulösen. Dafür wurde die Intel Realsense SDK als Basis genommen. In der Szene „PointCloudProcessingBlocks“ der SDK wird mit einer Punktwolke gearbeitet, was soviel heißt, dass die durch die Realsense aufgenommenen Bilder als Punkte mit x, y, z Koordinaten gespeichert werden, die wir für das auslesen der Tiefe benötigen. Diese Tiefendaten werden im RsDevice Objekt unter dem Pointcloud-Objekt bereitgestellt. Leider ließen sich weder konkrete Beispiele für C# und das Realsense Model D435 finden, noch ist es gelungen an die Tiefendaten durch ausprobieren zu gelangen.

Vuforia

Diese Variante wirkte anfangs sehr vielversprechend und Zuhause am Laptop funktioniert sie auch nur mit kleinen Ungenauigkeiten. Doch nach mehrmaligen testen am Tisch mussten wir diese Variante aufgrund seiner Unzuverlässigkeit leider aufgeben.
Die Unzuverlässigkeit während des Objekt trackings könnte daran liegen, dass Vuforia darauf ausgelegt ist in 3D, auf kurzer Entfernung und am besten durch ein Smartphone zu funktionieren. Das Smartphone würde dabei mehr mobilität und bewegungsfreiraum bieten.
Doch der große Abstand zwischen Tischplatte und Kamera sorgte für viel ungenauigkeit. Auch wäre eine Anwendung in der der User die Targets nur langsam bewegen kann und oft näher zur Kamera bringen muss, so dass diese wieder gefunden zu werden, nicht sehr Userfreundlich.
Aus diesen Gründen entschieden wir uns dagegen Vuforia zu nutzen.

reacTIVision

“reacTIVision” ist ein recht altes Framework, allerdings schien es einen interessanten Ersatz zu Vuforia zu bilden.
Wichtig zu wissen ist, dass die Anwendung - im Gegensatz zur Vuforia - nur einen 2D-Raum trackt, dies war allerdings für unser Szenario komplett ausreichend und könnte zusätzlich vielleicht auch die Performance verbessern.

reacTIVision wird viel für Soundtables genutzt, welche später im interaktiven Prototyp der Sound-Gruppe auch kurz erwähnt werden, und arbeitet mit Fiducials.
Nach ein paar Tests wurde schnell klar, dass sich unsere Vermutung bestätigte und das Tracking um einiges zuverlässiger als bei Vuforia ist.
Bei dem Versuch, das Framework in die SmartFactory zu integrieren, fiel dann aber aber auf, dass Reactivision neben einem notwendigen, außenstehenden Tool, mit einigem Overhead verbunden war. Mit den uns zur Verfügung stehenden Mitteln war nur über sehr fehleranfällige Übergangslösungen eine Integration in das Projekt gelungen. Da an diesem Punkt, dass parallel entwickelte Farb-Tracking schon sehr gut funktionierte, hat sich die Interaktionsgruppe entschieden reacTIVision fallen zu lassen.

Farbtracker

Als weitere Variante gibt es das Tracken von Farben, um dann mit bunten Fingerhüten oder anderen, farbigen Steuerungselementen die Anwendung bedienen zu können.
Während wir im letzten Semester bereits einen Farbtracker selber entwickelt haben, haben wir uns in diesem Semester für ein vielversprechendes Framework von Vexpot aus dem Unity Asset Store entschieden.
Das Framework hat sich als voller Erfolg herausgestellt. Es gab bereits sehr hilfreiche scripts zur Auswahl der zu trackenden Farbe etc. Außerdem gab es gute Demo-Szenen, die einen den Einstieg und das Verständnis zur eigenen Nutzung sehr vereinfacht haben.

Will man den Farbtracker nutzen ist es wichtig als erstes die *Main Camera & Farbtracker* und *EventSystemColorTracker* in die Szene zu integrieren.
In der *Main Camera & Farbtracker* ist es möglich Einstellungen zur Kamera, als auch zu den Scripts ‘ColorTrackerPanel’ und ‘ColorTrackerRenderer’ vorzunehmen. Die wichtigste Einstellung ist dabei die zu trackenden Farben zu konfigurieren.
Das *EventSystemColorTracker* mit dem Script ‘TouchlessInputModule’ haben wir dazu genutzt, um mit den getrackten Farben Events, wie das Klicken eines Buttons, auslösen zu können. Dabei konnte bestimmt werden wie lange die Farbe beispielsweise über einem Button sein muss, damit dieser ausgelöst wird.

Um das Framework auf unsere Anwendung anzupassen mussten wir allerdings noch ein paar EInstellungen vornehmen.
Da die Anderen Interaktionsmöglichkeiten nicht den gewünschten Erfolg aufwiesen, entschieden wir uns dafür alles mit dem Farbtracker umzusetzen.

Dafür brauchten wir ingame zwei Farben die unterschiedliche Aufgaben übernehmen konnten.

Ingame ist es notwendig eine Farbe zu haben, die Gebäude positionieren kann, während eine weitere Farben zusätzliche Einstellungen & Events auslösen kann.
Wir haben uns dafür das Array der Farben zu nutze gemacht und bestimmt, dass die Farbe mit dem geringsten Index, die sein soll, welche als Cursor zum Positionieren der Gebäude dienen soll. Gleichzeitig können jedoch weiterhin alle Farben Events auslösen.
Der Cursor zur Positionierung wird visuell auf der Bodenfläche der Fabrik farblich kenntlich gemacht, damit gesehen werden kann, welche Farbe für die Positionierung zuständig ist.

Insgesamt funktioniert der Farbracker sehr gut und wir sind mit dem Resultat zufrieden.
Gegebenenfalls können noch kleine Anpassungen in der Kalibrierung vorgenommen werden, damit es noch smoother ist und ein leichtes wackeln verhindert wird.

Abschließend können wir sagen, dass der Farbtracker und Reactivision wohl die vielversprechendsten Lösungen zu sein scheinen. Die Einbindung in unser bestehendes Projekt sich mit dem Farbtracker, aber als einfacher herausstellte und mit ein paar Anpassungen alle benötigten Funktionen umsetzbar waren.
So haben wir uns letztendlich dafür entschieden nur den Farbtracker umzusetzen.

Sonification

Planung

Soundfactory-Prototyp

Es wurde zuerst ein Prototyp erstellt, um die Übertragung der Daten wie in der Smartfactory aus Unity nach Pure Data zu simulieren. Zudem wurde der Prototyp angelegt um spielerischer mit Sound umgehen zu können und dabei herauszufinden welche Sounds zu der Smartfactory passen. Der Prototype wurde so aufgebaut, dass viele Methoden aus C# und Patches aus Pure Data direkt für die Smartfactory übernommen werden konnten. Es gibt beispielsweise auch in diesem Prototype einen Soundservice. Auf diesen wird unten eingegangen.
Eine weitere Idee für den Soundfactory-Prototyp war, für den Tisch eine ähnliche Interaktivität wie bei der Smartfactory zu implementieren. Es sollte zudem auch die Interaktivität der Tangibles mit der echtzeit Soundgestaltung in Pure Data getestet werden. Es wurde jedoch im Laufe des Semesters deutlich, dass das regelmäßige und freie Arbeiten am Tisch aufgrund von Covid-19 nicht möglich wird. Deshalb wurde eine andere Möglichkeit gesucht, um interaktive Inputs zu testen.

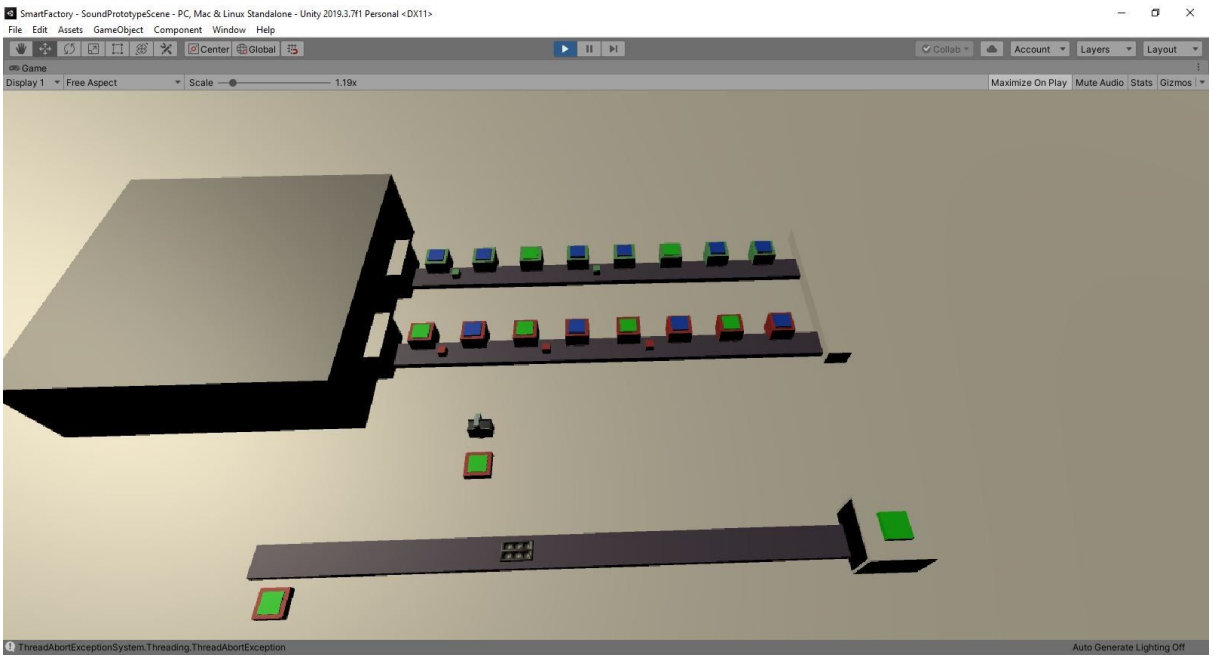


Abbildung: Soundfactory-Prototyp

Interaktiver Prototype

Der interaktive Prototype wurde mithilfe von Reactivision, Unity, dem OSC-Protokoll und Pure Data realisiert. Dafür wurde ein Whiteboard im Homeoffice genutzt, auf dem mit Magneten befestigte Fiducials als interaktive Tangibles dienen. Diese werden von einer Realsense mittels Reactivision getrackt und an Unity weitergegeben und von dort, wie auch in der Smartfactory, mittels OSC-Protokoll an Pure Data gesendet. Das Grundlage für das Unity-Projekt kam von einem Open-Source-Projekt das aktualisiert und verändert wurde.

Abbildung: Fiducials auf einem Whiteboard

Der Interaktive Prototyp ist sehr musikalisch und kann gut für einen Soundtable genutzt werden. Insgesamt konnte dadurch viel freier gearbeitet werden, wobei viele Patches die in der Smartfactory genutzt werden hierbei entstanden sind und generell mehr Ideen für die Sonification kamen.
Außerdem demonstriert der Prototyp sehr anschaulich Funktionalität und die Möglichkeiten von Reactivision.

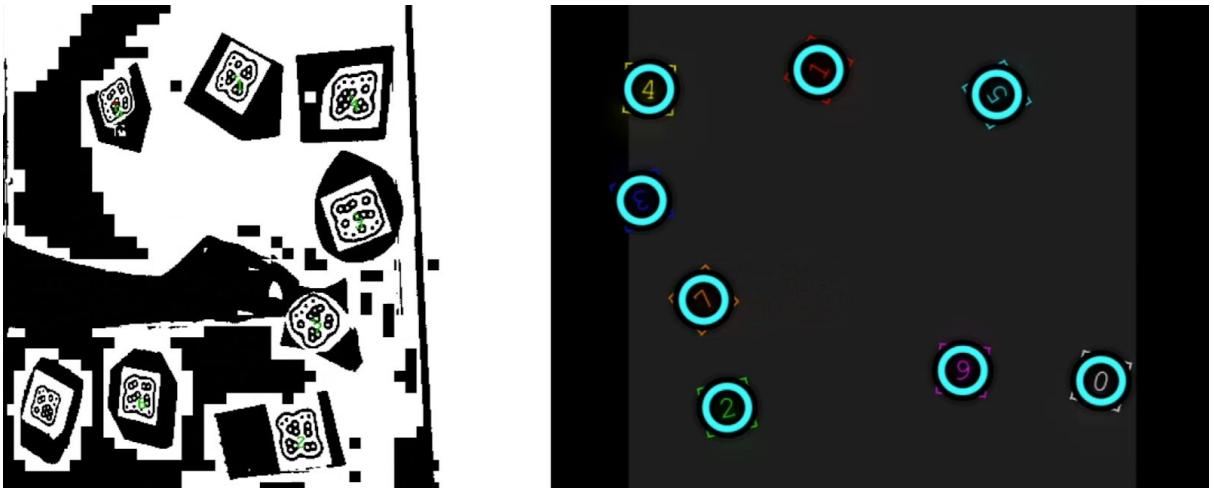


Abbildung: Das in Reactivision getrackte Bild (Links) und die Darstellung in Unity (Rechts)

Umsetzung

Im Vorhinein kann man sagen, dass eine Sonification eigentlich nur im kleinen Maßstab passiert und eine umfangreiche Sonificationen ohne direktes Arbeiten an dem Tisch mit funktionierender Smartfactory nicht realisierbar war. Jedoch werden Daten aus der Smartfactory an Pure Data gesendet und dort genutzt, um in Echtzeit Sound auszulösen oder zu modulieren, was an sich bereits als Sonification definiert wird.

Implementierung im Projekt

Damit die Daten aus Unity mit Pure Data weiterverarbeitet werden können, wurde ein eigener Soundservice in dem Projekt angelegt. Dieser verwendet mit der Methode „PlaySound“ das Open Sound Control Protocol, das zur Kommunikation mit Pure Data dient. In der Methode wird ein String übergeben, mit dem in Pure Data interpretiert werden kann, welches Ereignis aktuell verarbeitet werden muss. Zusätzlich wird ein Integer übermittelt, der die Möglichkeit gibt, das Ereignis weiter zu spezifizieren.

```
12 references
private void PlaySound(string msg, float index){
    oscSender.PlaySoundOSC("/" + msg + index);
}
```

So werden beispielsweise beim Erstellen verschiedener Module, verschiedene Integer übergeben, sodass jedes Modul einen eigenen Sound erzeugt. Damit der Soundservice nicht durchgehend bei den verschiedenen Managern und Objekten nachfragen muss, ob es eine Veränderung gibt, wird er von dem Eventservice unter anderem mit Hilfe von UnityEvents benachrichtigt. Des Weiteren werden manche Daten nur bei einer Veränderung

an Pure Data übertragen. Bei dem Interaktiven Prototyp hat sich gezeigt, dass es zu einer Verzögerung in der Übertragung bei dem durchgängigen Senden von Daten kommen kann.

Datennutzung in Pure Data

Pure Data nutzt das *netreceive* Objekt um BLOB's auf einem ausgewählten Port zu empfangen, kann daraufhin mit *oscparse* die mittels OSC gesendeten Daten direkt verarbeiten und leitet diese in Pure Data an die gewünschten Stellen weiter.

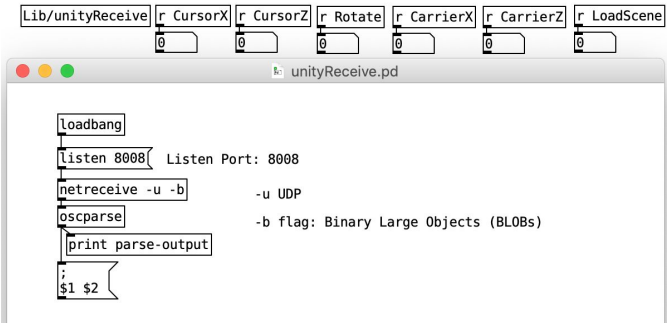


Abbildung: Patch für das Empfangen der Daten aus Unity

Soundgestaltung

Da für die Soundgestaltung in Pure Data zum erstens sehr viele Grundlagen der Akustik nötig sind und zum zweiten die Programmiersprache selbst erlernt werden musste, wurden zuerst sehr viele Methoden der Klangmodulation in Pure Data ausprobiert. Mit der Zeit sind zahlreiche Patches entstanden und für eine effektive Nutzung für die Smartfactory wurden viele mit Interface, Persistenz und Polyphie implementiert. Für die Sonification der Smartfactory wurden dann unterschiedliche Patches ausgesucht und angepasst.

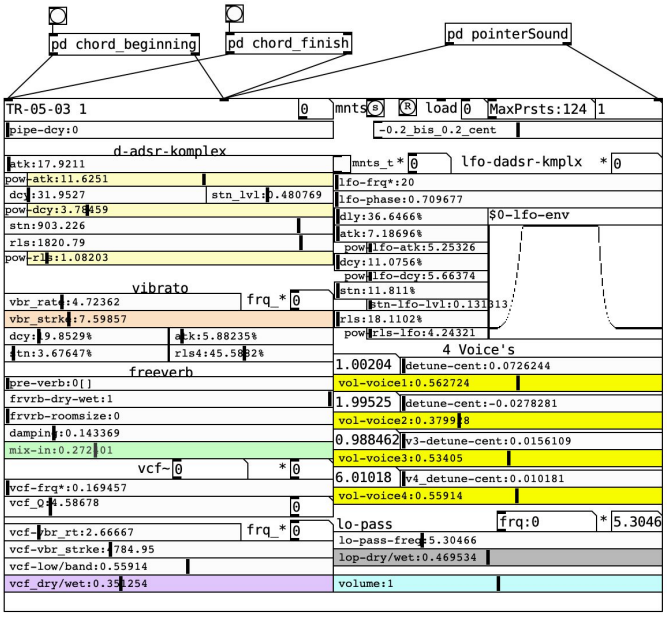


Abbildung: Interface eines Synthesizers

Auswahl-Sound

Das Wechseln der Felder mittels des Cursors in der Smartfactory wurde genutzt um einen “Auswahl-Sound” auszulösen. Für diesen wurde ein Synthesizer gepatcht, dessen Klänge durch additive- und subtraktive Klangsynthese erzeugt werden.
Das heißt insgesamt hat der Synthesizer vier Grundwellen, die mit unterschiedlichen Frequenzen schwingen. Diese haben eine gemeinsame Hüllkurve für die Lautstärke und werden außerdem durch verschiedene Effekte geleitet, die mit Hüllkurven, LFO’s und Dry/Wet-Reglern die Grundwellen modulieren können.

Des Weiteren wurde mit dem gleichen Synthesizer ein Sound gepatched, der signalisieren soll, dass ein Feld schon belegt ist. Für das passende Soundfeedback wurden die vier Grundwellen in der Frequenzhöhe so verstellt, dass der Sound dadurch unharmonisch klingt, was eine Unstimmigkeit bei dem*der Höhrer*in verursachen soll und signalisieren soll, dass man kein Building an dieser Stelle positionieren kann.

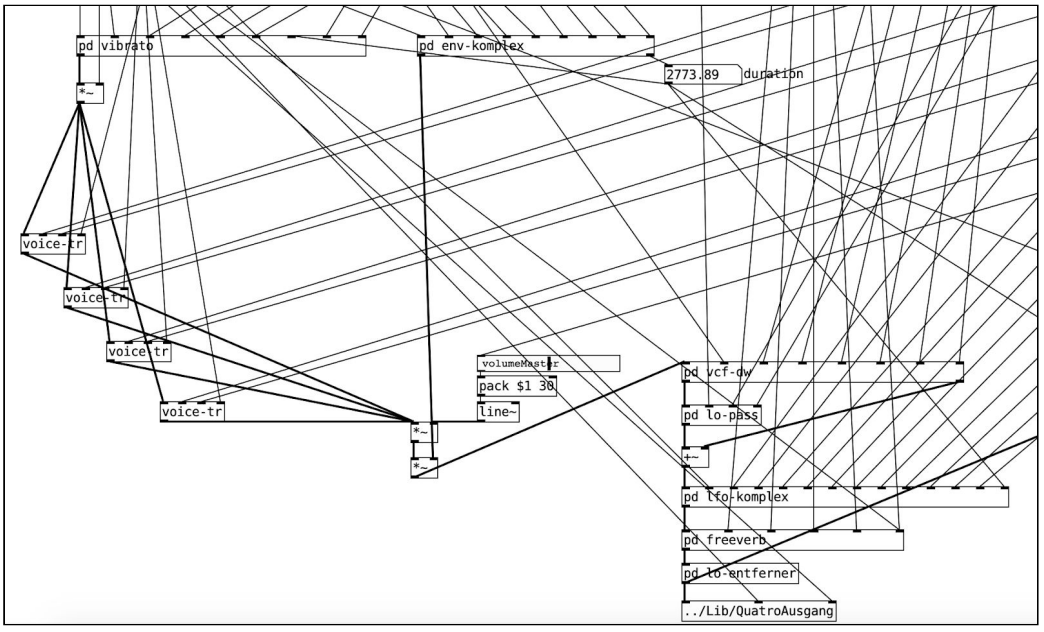


Abbildung: Signalfluss und Effektkette des Synthesizers

Cursorposition

Die genaue Position des Cursors wurde genutzt um einen Panning effekt für vier Lautsprecher auszulösen. Das bedeutet, dass sich die Lautstärkeverteilung der Lautsprecher je nach Position des Cursors verändert.

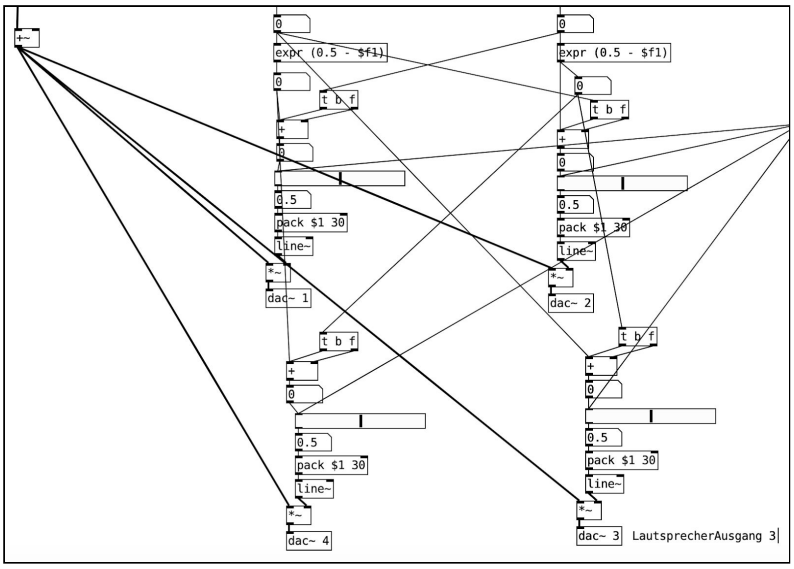


Abbildung: Lautstärkeverteilung auf vier Ausgänge

Um den Raumwirkung beim Cursor weiter zu intensivieren, verstärkt sich ein Reeverb-Effekt auf der vertikalen Ebene des Tisches. Dabei wird davon ausgegangen, dass der*die Nutzer*in an der Vorderseite des Tisches steht.

Buildings

Für den Sound beim Bauen eines Buildings in der Smartfactory wurde ein Sampler mit eingebauten Effekten programmiert. Desweiteren hat der Sampler die Möglichkeit Presets zu speichern. Es wurden verschiedene Samples zurechtgeschnitten und für jedes Building ein eigenes Preset erstellt. Beim Bauen eines Buildings wird das entsprechende Preset geladen und der Sound ausgelöst. Auch hier wird die Position von dem Cursor genutzt, um dem Sound auf die Boxen zu verteilen. Hier wurde für die Intensivierung des Raumes ein Echo-Effekt in der Vertikalen des Tisches genutzt. Je weiter entfernt ein Building von der Frontseite des Tisches platziert wird, desto größer werden die zeitlichen Abstände des Echos.

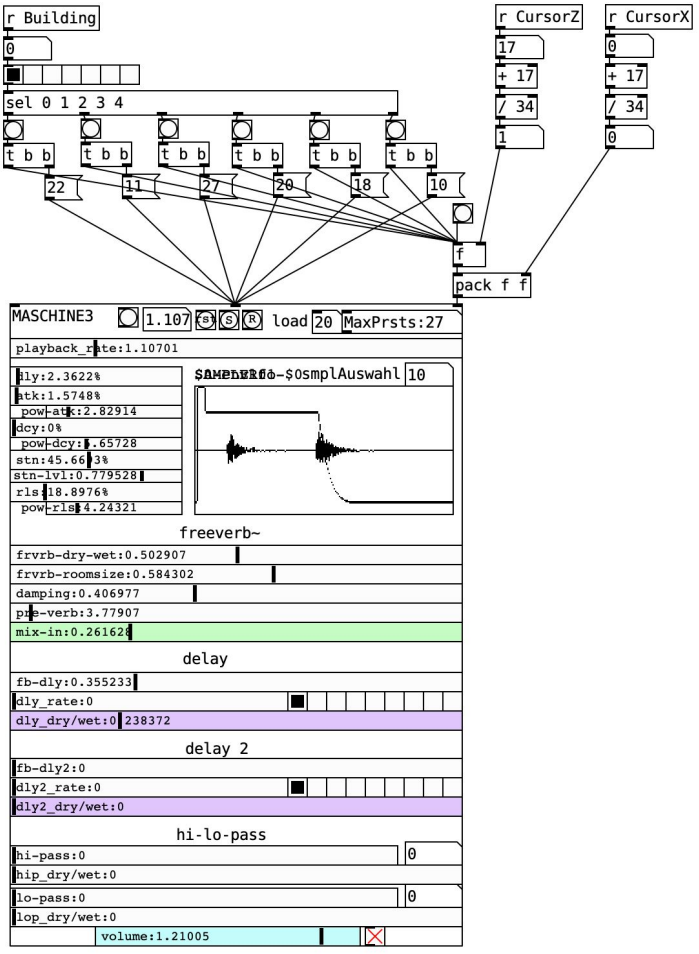
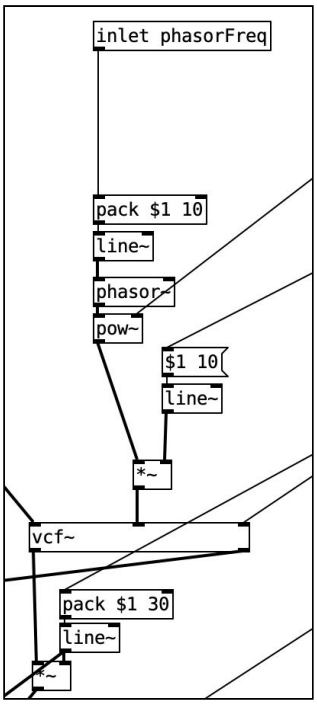


Abbildung: Samplerinterface und Inputs

Bewegung vom Carrier

Wenn sich ein Carrier auf dem Fließband in Bewegung setzt, wird ein Fließbandsound ausgelöst. Für diesen Sound wurde White-Noise genutzt, das durch einen Voltage-controlled filter geht. Die Cutoff-Frequenz des Filters wird durch einen LFO in Sägezahnform moduliert. Dadurch öffnet sich der Filter in einer gewünschten Frequenz, die für die Geschwindigkeitswahrnehmung des Fließbands genutzt werden kann. Dies wird beispielsweise beim Starten und Stoppen genutzt. Es wurde auch der Sound für die Drehung des Carriers programmiert, jedoch ist es schwierig diese Aktion aus den vorhandenen Daten in Unity zu extrahieren.



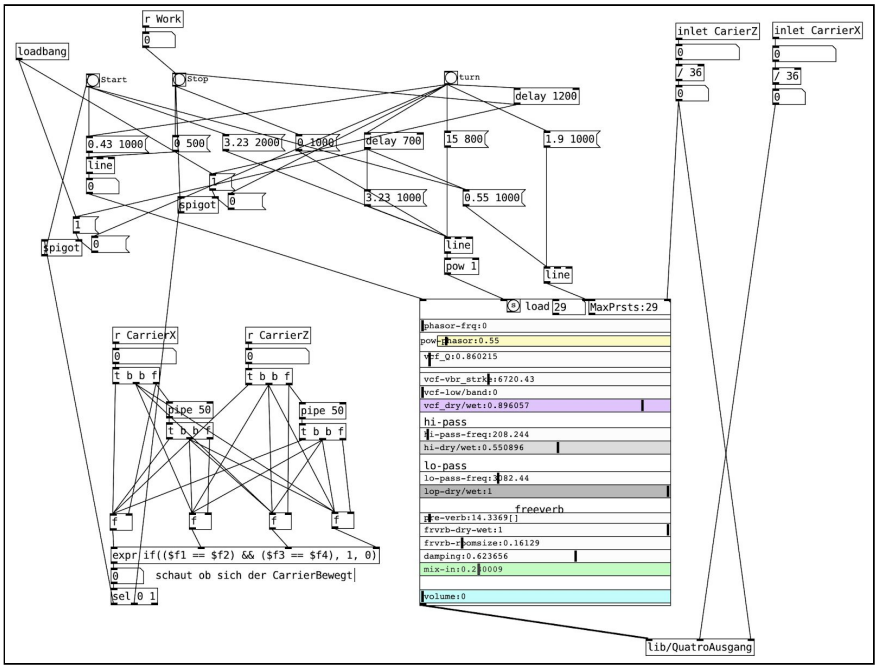


Abbildung: VCF und LFO Abbildung: Whitenoise-interface und Carrierbewegungslogik

Workstations und Weiteres

Insgesamt wurden verschiedene Sounds für die Workstations entwickelt. Da jedoch die Smartfactory auf dem Homeoffice-Computer bei fast allen Workstations abstürzt und keinerlei Daten übertragen werden, ist nur ein Sound für die Workstations und ein Sound für den Storage eingebaut. Des Weiteren wurden noch Sounds für das Laden einer Factory und das Fertigstellen eines Auftrags gepatched.
Leider konnte auch die verteilung vom Sound auf dem Tisch nicht vollständig getestet werden.

Technische Einstellungen

Pure Data muss für den Sound installiert sein. Die Patches wurden mit der Vanilla Version 0.50-2 erstellt.
Bei dem Einrichten von Pure Data am Tisch, sollte darauf geachtet werden den entsprechenden Treiber für das Audiointerface “Behringer U-Phoria UMC204HD” zu installieren. Der Treiber sollte daraufhin in den Audioeinstellungen von Pure Data bei “Ausgabegeräte” ausgewählt werden und die Kanäle von zwei auf vier erweitert werden. Dadurch können alle vier Lautsprecher von Pure Data einzeln angesprochen werden.
Die Samplingrate und Blocklänge sollten sich mit dem passenden Treiber automatisch anpassen. Sollte es dennoch zu Kratzen in der Soundwiedergabe kommen, sollten die Parameter der Samplingrate und Blocklänge angepasst werden.

Für die korrekte Darstellung von den Interfaces in Pure Data, muss die Schriftart “DejaVuSansMono” auf dem Computer installiert werden.

Starten der Sonification

Die beschriebenen Patches befinden sich auf der Stations-Branch, welches die einzige Branch war, die einigermaßen auf dem Computer im Homeoffice funktioniert hat.
Es muss der "__Main.pd" Patch gestartet werden, der im Ordnerverzeichnis "SmartFactory/PureData/" liegt. Daraufhin sind die ersten Patches aktiv. Die Patches wurden so angelegt, dass zuerst die Patches für das Bauen einer Smartfactory aktiviert werden. Hier ist die Sonification vom Cursor sowie der Drehung und der Platzierung der Buildings aktiv. Sobald eine gespeicherte Smartfactory geladen wird, werden die eben genannten Patches deaktiviert und die Patches für den Ablauf eines Auftrags aktiviert. Eine erneute Aktivierung der ersten Patches ist zurzeit nicht implementiert, da hier noch Daten von dem Menü der Smartfactory fehlen.
Beim erneuten Bauen einer Smartfactory nach dem Laden, muss für den Sound Pure Data geschlossen und der “__Main.pd” Patch neu gestartet werden. Die Gesamtlautstärke kann im “__Main.pd” Patch eingestellt werden.

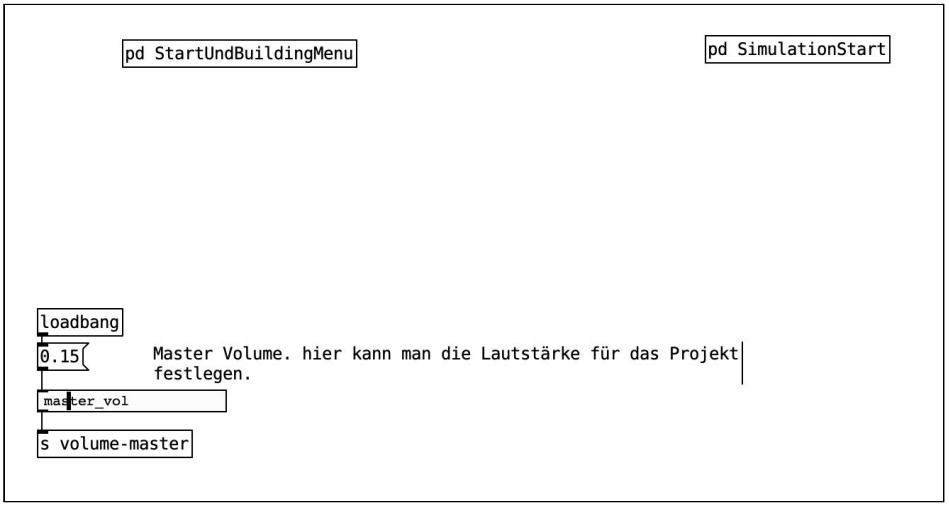


Abbildung: Ausschnitt aus dem __Main.pd Patch

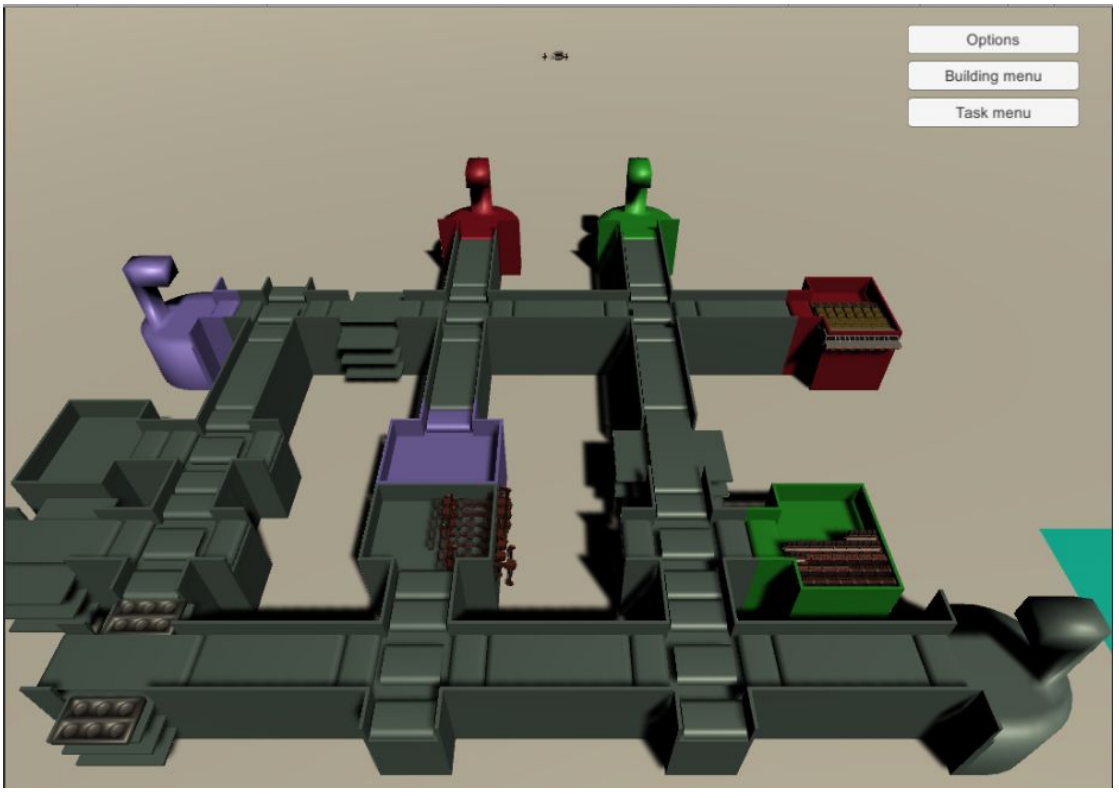
Produktionsimplementierung

Planung

Nach der Entscheidung modular (Spielzeug-)Autos zu produzieren, wurde geschaut, wie man diese am besten durch 3D-Modelle modular darstellen kann. Hierfür wurde sich der Einfachheit halber ein Asset-Pack im Unity Asset-Store gekauft. Diese Pack beinhaltet einfache Bauteile, wie Unterbauten, mit denen die Autos modular aufgebaut werden können und sinnvoll zusammengestellt werden können.

Umsetzung

Für die Umsetzung wurde beispielhafter Aufbau erzeugt:



Dieser besteht aus vier Fertigungsstationen und fünf Lagerorten. Standardmäßig wurden die Teile, die für die nächste Fertigung benötigt werden in die Lagerstation vor der Fertigungsstelle gelegt. So sind im ersten Schritt alle Chassis-Arten und Reifen im ersten Lager vorhanden. Startet man nun einen Auftrag, fährt der carrier zunächst zur ersten Lagerstation, holt sich das über das Menü gewählte Chassis und die Refien ab und lagert die Fertigung in der nächsten Station (hier in grün). Geht der Auftrag nun weiter holt sich der Chassis das vorher gefertigte Teil und einen Motor und bringt dies zur nächsten Fertigungsstation. Am Ende wird das fertige Auto auf dem letzten Lagerort platziert.

Über das Fertigungsmenü hat man vorher die Möglichkeit verschiedene Varianten des Autos zu fertigen:

Logging

Planung

Um Feedback von Programm zu erhalten haben wir uns dazu entschieden, eine Logging Schnittstelle einzubauen. Diese soll helfen Grundlegende Informationen auszulesen und zu archivieren, gegebenenfalls sollen diese Informationen dabei helfen Fehler zu analysieren und den grundlegenden Ablauf schriftlich festzuhalten. Diese Informationen werden in einer CSV File gespeichert damit der Nutzer diese einsehen kann. Anfangs war die Idee, dass es eine Datei für die Fehleranalyse gibt und eine für Grundlegende Informationen, wie z.B. wo sich die Carrier befinden und wann sie wo welche Station besucht haben. In der Fehleranalyse sollten Daten gespeichert werden, die zu verzögerung des Prozess geführt haben. Wie z.B. ein Stau oder falsche Konfiguration der Maschinen.

Umsetzung

Da erste was wir tun mussten um Daten auszulesen, war es eine Klasse zu erstellen, die es ermöglicht zu checken ob schon eine Log Datei vorhanden ist, oder eine neue erstellt werden muss. Dafür haben wir den CSVManager erstellt. Diese Klasse hat mehrer Funktionen die uns dabei helfen Strings in Dateien zu speicher. Immer wenn das Programm gestartet wird, guckt der CSVManager zuerst ob schon ein Ordner für die Logs vorhanden ist, falls nicht erstellt er diesen.

```
static void VerifyDirectory()
{
    string dir = GetDirectoryPath();
    if (!Directory.Exists(dir))
    {
        Directory.CreateDirectory(dir);
    }

}

3 Verweise
static string GetDirectoryPath()
{
    return Application.dataPath + "/" + reportDirectoryName;
}
```

Im nächsten Schritt wird geguckt ob schon eine CSV Datei vorhanden ist um die erhaltenen Informationen darin zu speichern, falls nicht wird diese erstellt.

```
static void VerifyFile()
{
    string file = GetFilePath(
    if (!File.Exists(file))
    {
        CreateReport();
    }
}
```

Eine Logfile wird angelegt in dem die Überschriften übergeben werden und diese strukturiert archiviert werden. Dafür muss oben ein Array mit den vorgegeben Überschriften erstellt werden, dieser wird dann durch einen For Loop durchiteriert und in die Datei geschrieben. In unserem Fall gibt es als Überschriften “Building”, “status” und die aktuelle Zeit.

```
public static void CreateReport()
{
    VerifyDirectory();
    using (StreamWriter sw = File.CreateText(GetFilePath()))
    {
        string finalString = "";
        for (int i = 0; i < reportHeaders.Length; i++)
        {
            if (finalString != "")
            {
                finalString += reportSeparator;
            }
            finalString += reportHeaders[i];
        }
        finalString += reportSeparator + timeStampHeader;
        sw.WriteLine(finalString);
    }
}
```

Um etwas in die Logdatei zu schreiben muss eine Verbindung zwischen unserer CSVManager Klasse und der EventServices Klasse erstellt werden. Durch die EventServices bekommen wir einen Array mit Strings, in dem unsere Daten stehen wie z.B. wann ein Building platziert wurde. Dies wird dann an unsere Klasse weitergeben und wieder der Array wird wieder durchiteriert und in die Logdatei geschrieben. Hierbei muss darauf geachtet werden, das man die Daten an die richtige stelle weiter gibt weil es sonst nicht mehr mit den Überschriften zusammen passt und es zur verwirrung und der Verfälschung der Daten kommt

```
public static void AppendToReport(string[] strings)
{
    VerifyDirectory();
    VerifyFile();
    using (StreamWriter sw = File.AppendText(GetFilePath()))
    {
        string finalString = "";
        for (int i = 0; i < strings.Length; i++)
        {
            if (finalString != "")
            {
                finalString += reportSeparator;
            }
            finalString += strings[i];
        }
        finalString += reportSeparator + GetTimeStamp();
        sw.WriteLine(finalString);
    }

}
```

Hier haben wir einmal die Methode aus der EventServices Klasse in der wir die ID des gesetzten Carriers bekommen und diese an unsere Klasse weitergeben damit jeder plaziert

Carrier geloggt werden kann. Dazu bekommt der Nutzer Feedback in dem im UI ein Text angezeigt wird was genau gerade passiert ist.

```
public Carrier carrierSpawned(int index)
{

    string[] logging = new string[2];
    logging[0] = "Carrier-" + carrierSessionID++.ToString();
    logging[1] = "spawned";
    textupdate.changeText(logging[0] + " " + logging[1]);
    CSVManager.AppendToReport(logging);
    return carrierservices.CreateCarrier(index);
}
```

Zusammenfassung

Dieses Dokument stellt einen Überblick über das zweiten Semester des Bachelor-Projekts “Mixed Reality” des Studiengangs Medieninformatik, Hochschule Bremen, dar.

Während im Laufe des ersten Semesters drei Prototypen entwickelt wurden, wurde in diesem Semester gemeinsam an einem großen Projekt gearbeitet. Dennoch gab es mehrere Untergruppen, die auf bestimmte Themengebiete spezialisiert waren:

Die **Softwarearchitektur** wurde von einer Gruppe erstellt, anschließend auch die einzelne Bereiche umgesetzt.

Die **Interaktionsgruppe** hat verschiedene Interaktionsmöglichkeiten ausgearbeitet und anschließend, die für unsere Anwendung passenden umgesetzt und diese in die Anwendung der Softwarearchitektur Gruppe eingefügt.

Bei der **Sonification** wurden die benötigten Daten aus der Applikation mit der datenstromorientierten Programmiersprache Pure Data verarbeitet, sodass die User auch ein akustisches Feedback der Aktionen der Smartfactory bekommen.

Die **Logging Gruppe** hat eine Schnittstelle für das Loggen von Daten der Smartfactory entwickelt. Mithilfe dieser können Informationen über den Produktionsprozess in externen Logdateien festgehalten werden.

Trotz den abgetrennten Aufgabenbereichen gab es Überschneidungen, so dass zum Beispiel Teile der Softwarearchitektur auch durch Input anderen Personen entstanden.

Verantwortlichkeiten

Moumita Ahmad	Interaktion, Menü, Tracking
Annie Berend	Sonification, Filmdreh, Filmschnitt
Moritz Hodler	Logging, Filmdreh
Robin Jacobse	Controller, Produktionsaufträge, Pathfinding, Manager, Persistenz
Reda Khalife	Spracherkennung
Bjarne Lehmkuhl	Logging
Philipp Moritzer	Abstract, Softwarearchitektur, Produktionsimplementierung, Prototyp, Formatierung
Joscha Sattler	Interaktionen, Hauptmenü, Farbtracker
Tobi Schmitt	Sonification, reacTIVision
Alisa Schumann	Interaktion, Tracking mit Realsense
Leonard Tuturea	Prototyp, Interaktion, Filmdreh