Copy&Motion

SOFTWARE ENGINEERING PROJEKT

Alina Sakowski	937961
Fabian Gusek	937820
Alexander Munkelt	937624
Felix Seeburg	933730

PROJEKTVERANTWORTLICHE: ALINA SAKOWSKI

SOFTWARE ENGINEERING | 20.06.2022

INHALTSVERZEICHNIS

PHASE 1 – ANALYSE

VERANTWORTLICHE: Alina Sakowski

EXPOSÉ

Die Semesteraufgabe ist die Erstellung einer vermarktbaren Software, die ein Eingabegerät, ein Ausgabegerät und eine Steuerung verwendet, wobei Ein- und Ausgabegerät aus dem 'Media IP Lab' der Fachhochschule Kiel stammen. Die Geräte sollen über das Protokoll Open Sound Control (OSC) miteinander kommunizieren.

Die Software Copy&Motion soll die Darstellung von verschiedenen Mustern mittels LEDs per Finger- und Handgesten ermöglichen.

Als Eingabegerät dient ein Leap Motion Controller, welcher ein Infrarot-Sensor ist, der Objekte und Bewegungen erfasst. Hier werden die Gesten angenommen.

Die Verarbeitung der Gesten und die Steuerung der Netzwerkkommunikation werden durch die geplante Software in Form von Parser und Controller übernommen. Zur Übertragung über das Netzwerk wird Open Sound Control (OSC) genutzt. OSC ist ein nachrichtenbasiertes Netzwerkprotokoll, welches die Daten über das lokale Netzwerk der Fachhochschule an Ausgabegeräte sendet.

Das Ausgabegerät ist das “KIELerLEUCHTEN” Projekt (LightHouse) der Creative Technologies AG der FH Kiel. Hier werden Scheinwerfer, die sich in den Fenstern des Hochhauses des Fachbereichs Wirtschaft befinden, angesteuert. Zur Vereinfachung beschränkt sich das Projekt auf ein gedrucktes Modell bzw. auf die App des Hauses, welche nach derselben Struktur wie das große Haus funktionieren.

Ergänzend dazu wird mit den anderen Laborgruppen ein Gesamtszenario dargestellt. Da alle Gruppen das LightHouse als Ausgabegerät gewählt haben, beschränkt sich das Gesamtszenario auf eine Schnittstelle, die Überschneidungen beim Senden ans LightHouse verhindern soll.

Es wurden folgende Teilprobleme und Lösungsvorschläge ermittelt:

1) Der Leap Motion Controller muss die ausgeführten Gesten erkennen können.
    - Dafür werden Gesten aus der Standardbibliothek des Leap Motion Controllers verwendet.
2) Das System muss OSC Daten empfangen, senden und verarbeiten können.
    - Um dies zu gewährleisten, werden OSC-Bibliotheken in die Programme eingebunden.
3) Es gibt Überschneidung beim Senden an das LightHouse mit den anderen Gruppen.
    - Um Überschneidung zu vermeiden, wird zwischen den Gruppen eine Schnittstelle implementiert, welche prüft, ob der Übertragungskanal frei ist.
4) Das LightHouse muss das richtige Muster darstellen.
    - Dafür wird im Controller jedem Muster eine Geste zugeordnet.

ANFORDERUNGSANALYSE

LASTENHEFT

1. ZIELBESTIMMUNG

Die Fachhochschule Kiel soll durch die Software Copy&Motion die Möglichkeit erhalten, verschiedene Muster am LightHouse per Fingergesten darzustellen.

2. PRODUKTEINSATZ

Die Software Copy&Motion dient der Fachhochschule Kiel als Werbetool, da die Lichter des Hochhauses Aufmerksamkeit erzeugen. Die Software kann in Kombination mit der im Media IP Lab befindlichen Nachbildung des Hochhauses (LightHouse) verwendet werden. Darüber hinaus kann die Software auch mit dem echten Hochhaus interagieren.
Zielgruppe des Produktes sind Studierende und Studieninteressierte.

3. PRODUKTÜBERSICHT

Das Use-Case-Diagramm visualisiert die Interaktion von Akteuren mit dem zu entwickelnden System.

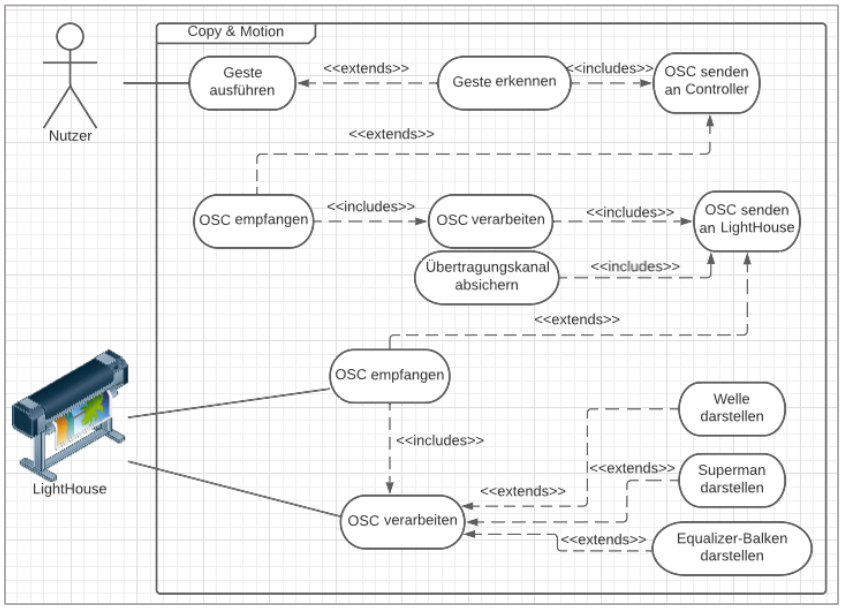


Abbildung 1 - Use-Case-Diagramm

4. PRODUKTFUNKTION

/LF10/	*Geste ausführen* **Akteur:** Nutzer **Kurzbeschreibung:** Eine Geste wird über dem Leap Motion Controller ausgeführt.
/LF20/	*Geste erkennen* **Akteur:** Nutzer **Kurzbeschreibung:** Der Parser erkennt die Geste und stellt eine OSC-Nachricht zusammen.
/LF30/	*OSC senden an Controller* **Akteur:** Nutzer **Kurzbeschreibung:** Parser sendet eine OSC-Nachricht an den Controller
/LF40/	*OSC senden an LightHouse* **Akteur:** Nutzer

	Kurzbeschreibung: Controller sendet eine OSC-Nachricht an das LightHouse
/LF50/	*OSC verarbeiten* **Akteur:** Nutzer **Kurzbeschreibung:** Eine empfangene OSC-Nachricht wird auf seinen Inhalt analysiert, um folgende Handlungen zu evaluieren.
/LF60/	*OSC empfangen* **Akteur:** Nutzer, LightHouse **Kurzbeschreibung:** Es wird nach OSC-Nachrichten gelauscht und diese werden empfangen, um weitere Handlungen durchführen zu können
/LF70/	*Welle darstellen* **Akteur:** LightHouse **Kurzbeschreibung:** Das LightHouse bildet eine Welle ab.
/LF80/	*Superman darstellen* **Akteur:** LightHouse **Kurzbeschreibung:** Das LightHouse spielt eine Superman Animation ab.
/LF90/	*Equalizer-Balken darstellen* **Akteur:** LightHouse **Kurzbeschreibung:** Das LightHouse bildet einen Equalizer ab.
/LF100/	*Übertragungskanal absichern* **Akteur:** Nutzer **Kurzbeschreibung:** Der Übertragungskanal wird abgesichert, um Überschneidungen beim Senden ans LightHouse zu verhindern.

5. PRODUKTDATEN

Da die Animationen fest im Code implementiert sind und keine Daten persistent gespeichert werden, sind keine Produktdaten erforderlich. Nach jeder Ausführung des Programms wird das Programm in seinen Urzustand zurückgesetzt.

6. PRODUKTLEISTUNGEN

/LL10/	LF20 darf nicht mehr als zwei Versuche brauchen, um eine korrekt ausgeführte Geste zu erkennen.
/LL20/	Sowohl Parser als auch Controller darf keine falschen OSC-Nachrichten senden (andere OSC-Nachricht als gewollt)
/LL30/	LF60/LF70/LF80 dürfen nicht länger als 10 Sekunden dauern, um komplett durchlaufen.
/LL40/	Das Programm muss reibungslos auf Windows 10 laufen.

7. QUALITÄTSANFORDERUNGEN

Für dieses Projekt sind die nach ISO/IEC 9126 definierten Qualitätsmerkmale folgendermaßen relevant:

Änderbarkeit:
Da den Gesten ein festes Muster für die Ausgabe zugeordnet wird, setzt dies die Änderbarkeit auf einen normalen Wert.

Benutzbarkeit:
Die Nutzung sollte in einem Readme leicht verständlich dokumentiert werden.

Effizienz:
Die OSC-Signale sollten zeitnah verarbeitet werden können und die Darstellung sollte möglichst Ressourcenschonend sein.
Funktionalität:
Auf dem LightHouse sollten die richtigen Muster ausgegeben werden.

Übertragbarkeit:
Die Übertragbarkeit ist nicht relevant, da die Software explizit für das LightHouse in dem Media IP Lab der FH Kiel entwickelt wird.

Zuverlässigkeit:
Eine geringe Fehlertoleranz bei Erkennung der Gesten sollte im Code implementiert werden.

PRODUKTQUALITÄT	SEHR GUT	GUT	NORMAL	NICHT RELEVANT
ÄNDERBARKEIT			x	
BENUTZBARKEIT		x		
EFFIZIENZ		x		
FUNKTIONALITÄT	x			
ÜBERTRAGBARKEIT				x
ZUVERLÄSSIGKEIT		x		

TESTFÄLLE

Nr.	Beschreibung	Use-Case(s)
/TF1/	Unbekannte Geste	Geste erkennen (LF20)
/TF2/	Bekannte Geste	Geste erkennen (LF20)
/TF3/	Unbekannter OSC Pfad	OSC empfangen (LF50)
/TF4/	Bekannter OSC Pfad	OSC empfangen (LF50)
/TF5/	Richtiger Output am LightHouse	Welle darstellen (LF60), Superman darstellen (LF70), Equalizer darstellen (LF80)
/TF6/	Schnelle Codeausführung und Anzeige auf dem LightHouse	Welle darstellen (LF60), Superman darstellen (LF70), Equalizer darstellen (LF80)

TESTPLAN

Alle oben genannten Test müssen zur Abnahme das erwartete Ergebnis geben. Alle Tests, die automatisiert werden können, sollen das auch sein.

AUFWANDSABSCHÄTZUNG & FAZIT

Die Dauer der Analyse Phase wurde auf 3 Mann-Tage geschätzt. Tatsächlich wurden 4 Mann-Tage benötigt. Die Abweichung lässt sich auf fehlende Erfahrung in der Planung zurückführen. Es ist davon auszugehen, dass die Planung in den weiteren Phasen besser funktionieren wird, da mehr Puffer für die einzelnen Aufgaben eingeplant wird, um Zwischenschritte zu berücksichtigen. Des Weiteren stand eine Umplanung auf Grund unklarer Wünsche des Kunden an. Daraus wurde gelernt, dass es unerlässlich ist, eine klare Definition der Arbeitsaufträge einzuholen.

PHASE 2 – MODELLIERUNG

VERANTWORTLICHER: Fabian Gusek

Zur Modellierung der inneren Struktur des Systems wurden verschiedene Diagramme herangezogen, um eine präzise Beschreibung der Systemfunktion zu erhalten.

GROBENTWURF

Zuerst wurde ein Grobentwurf entstellt, um eine generelle Struktur der Software festzulegen. Es wurde sich dafür entschieden den Grobentwurf als Zustandsdiagramm (State-Machine-Diagram) darzustellen, da dies das Verhalten der Software übersichtlich darstellen kann.

In den Kästen sind festen Zustände definiert. Die Pfeile beschreiben Aktivitäten oder Prüffälle.

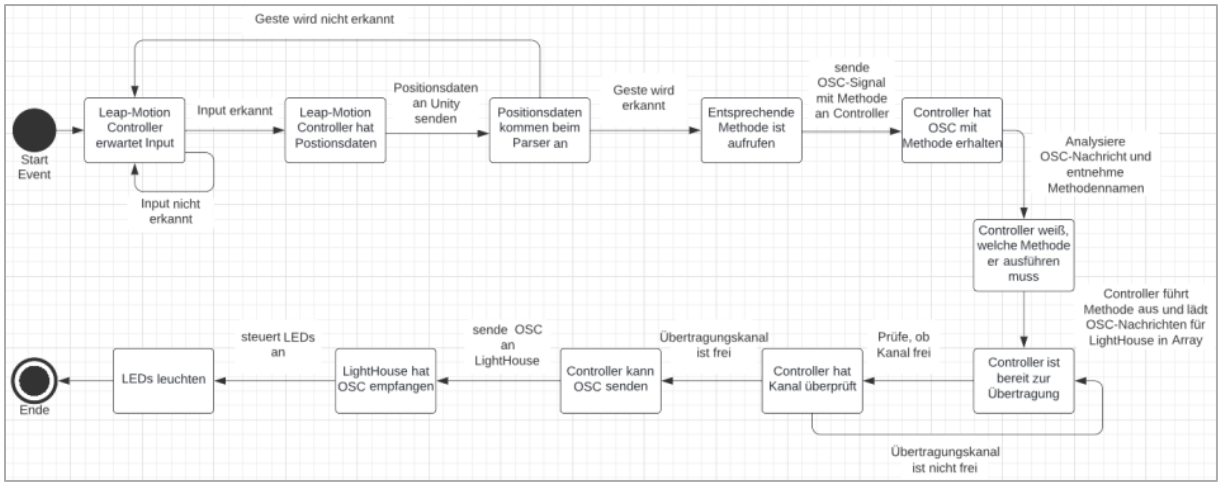
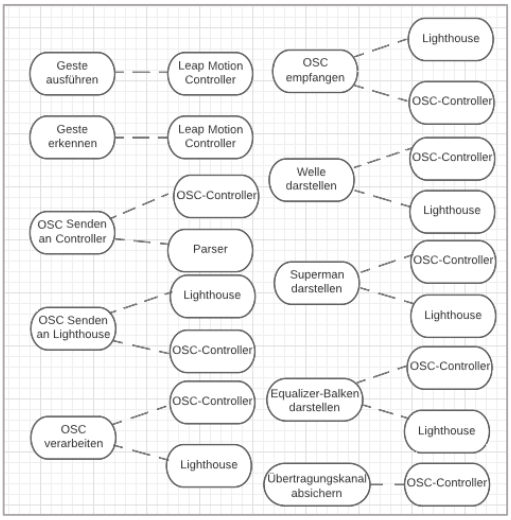


Abbildung 2 – State-Machine-Diagram

OOA

Anschließend wurde eine Objektorientierte-Analyse (OOA) durchgeführt, um den Übergang zur Implementierung zu erleichtern. Es wurde sich für eine Verhaltensorientierte Vorgehensweise, das Objektorientierte Software-Engineering (OOSE) entschieden, da hierbei zuerst Abläufe und im Anschluss die Klassen erstellt werden. Dadurch entsteht die Möglichkeit zuerst die kleinschrittigen Abläufe zu ordnen.



Im ersten Schritt wurden Szenarien anhand der Use-Cases ausgewählt. Anschließend wurde überlegt, welche Kollaboratoren benötigt werden. Dazu wurde geprüft, welche System-Objekte mitwirken sollen. Wie in Abbildung 3 ersichtlich, wurde für jeden Anwendungsfall eine Kollaboration zwischen den System-Objekten realisiert.

Abbildung 3 – Kollaborationsdiagramm

Aus den System-Objekten sind Klassenrollen entstanden, welche sich im untenstehenden Sequenzdiagramm wiederfinden. Hier wurde das Szenario „Geste ausführen“ (vgl. /LF10/) dargestellt. Als Ausgabegerät fungiert das LightHouse.

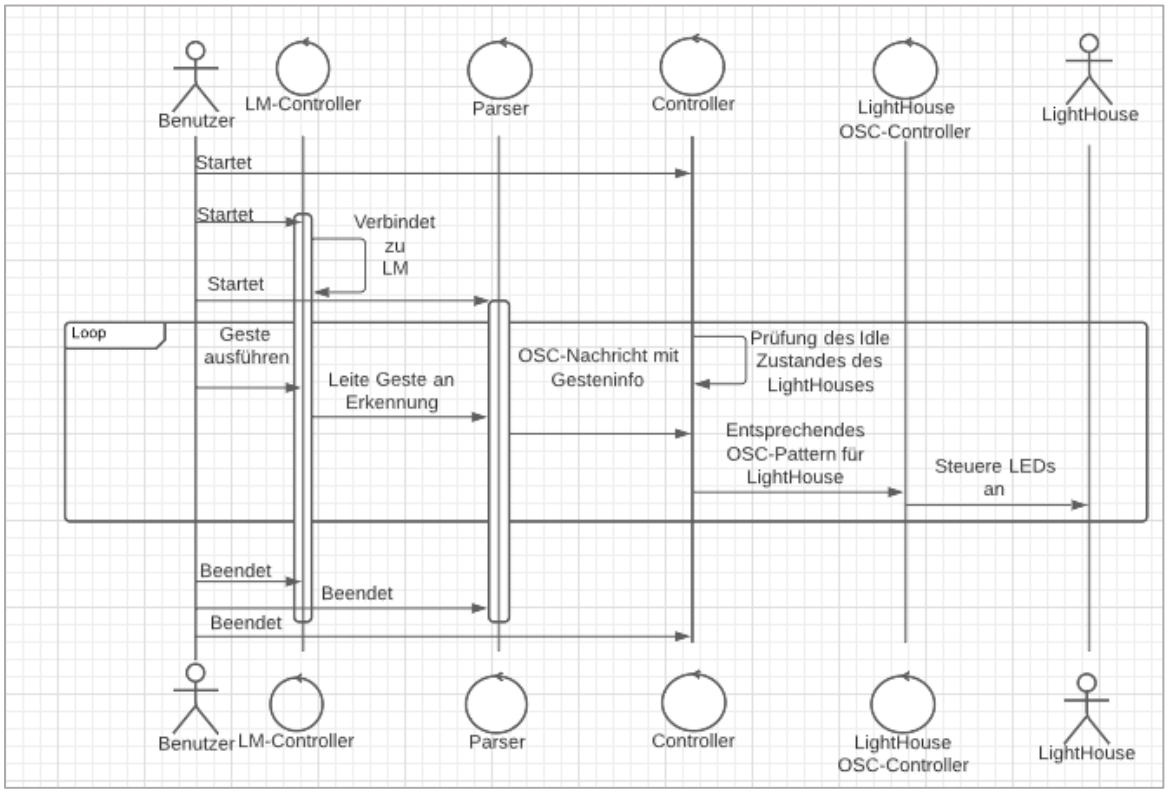
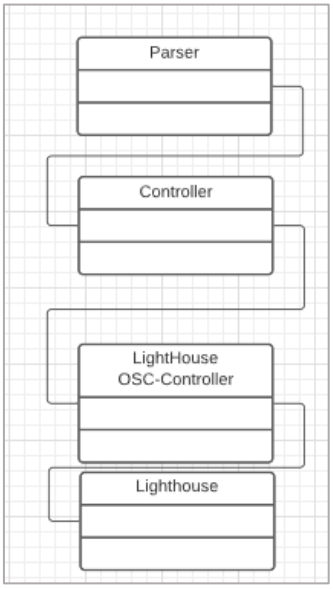


Abbildung 4 – Sequenzdiagramm



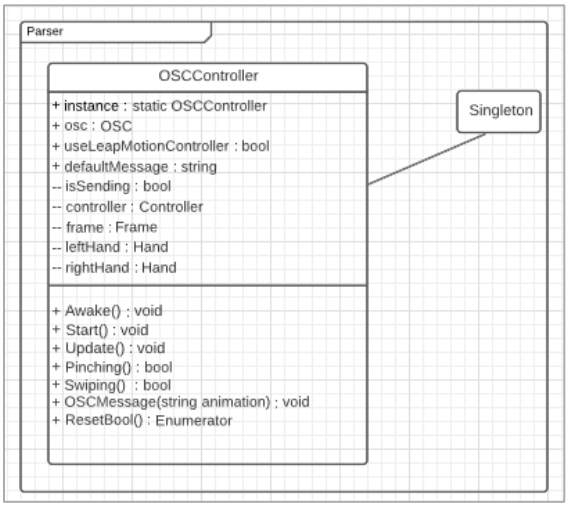
Anschließend ließ sich ausgehend vom Sequenzdiagramm ein grobes Teilklassen-Diagramm erstellen, da die Verbindungen nun deutlich wurden.

Im Anschluss wurden diese verfeinert werden. Das heißt Klassen werden durch Zusammenfassen optimiert, das Modell wird durch Operationen und Assoziationen erweitert, Attribute und Methoden werden gefunden und Strukturen und Abhängigkeiten können vervollständigt werden.

Abbildung 5 - grobes Teilklassen-Diagramm

FEINENTWURF

Durch die OOA ließ sich der Feinentwurf ableiten. Dieser legt die detaillierte Struktur der Software fest.



Das erste UML Diagramm zeigt den Parser, welcher in der Lage ist, Gesten zu erkennen. Anhand dieser Gesten wird eine OSC-Nachricht für den Controller generiert und versendet.

Abbildung 6 – Parser UML

Bei der Erstellung des Feinentwurfs der Steuerung stellte sich heraus, dass die Verwendung des Design-Patterns Singleton sinnvoll ist. Pattern helfen den Entwicklern bei Lösung von Problemen, da die Grundidee einer Lösung wiederverwendet werden kann. Ein implementierter Singleton lässt nur eine Instanz einer Klasse zu. Da Klassen wie „AnimationHandler“ oder „LEDHandler“ nur eine einzige Instanz benötigen, um die Funktionalität zu erreichen, welche gefordert ist, ist Singleton geeignet.

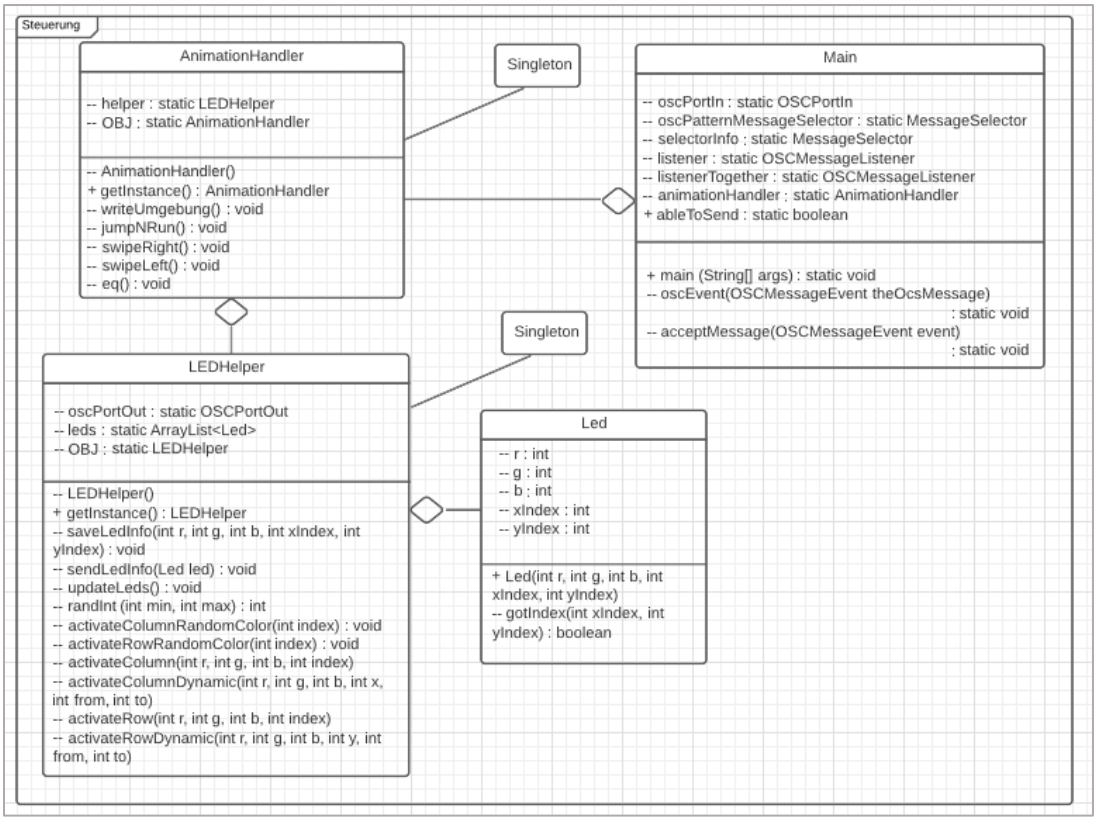


Abbildung 7 - Steuerung UML

Die Steuerung hört nun das Netzwerk nach OSC-Nachrichten des Parsers ab und lässt den AnimationHandler je nach Inhalt der Nachtrichten Methoden auslösen. Diese Methoden enthalten Muster aus OSC-Nachrichten für das LightHouse, welche mithilfe von LEDHelper und LED zusammengebaut und versendet werden.

CODING-STYLE

Als Orientierung für den Stil des Codes der Steuerung (Abbildung 7) wird der offizielle Java Coding Style von Google verwendet [1]. Der Parser ist in Unity / C# geschrieben (Abbildung 6). Dafür wird der Coding Stil von Avangarde [2] genutzt.

Die wichtigsten Namenskonventionen in Java sind:

- Attribute: name
- Klasse: Klasse
- Methode: eineFunktion()
- Konstante: KONSTANTE

Die wichtigsten Punkte in Unity / C# sind:

- Namespaces in PascalCase: Avangarde.CurrentGame.UI.MainMenu
- Klassen & Methoden in PascalCase: public class MyClass, public void SomeMethod{}
- Felder in camelCase: int someValue = 10;
- Statische Felder in PascalCase: public static int SomeStaticValue = 10

Für eine bessere Lesbarkeit werden die Namen in Camel-Case geschrieben (Beispiel: istMethode()). Außerdem wurde als Sprache Englisch gewählt.

AUFWANDSABSCHÄTZUNG & FAZIT

Die Dauer der Phase Modellierung wurde 3 Mann-Tage geschätzt. Tatsächlich wurden 4 Mann-Tage benötigt. Der Hauptgrund für die Abweichung ist, dass die OOA bei der Planung nicht berücksichtigt wurde. Nach der Zwischenpräsentation hat das Team das Feedback erhalten, dass es die OOSE falsch verstanden hat. Daraufhin wurde diese überarbeitet. Daher empfanden das Team die Zwischenpräsentation als sehr hilfreich.

PHASE 3 – ENTWICKLUNG

VERANTWORTLICHER: Alexander Munkelt

AUFGABEN

Zu Beginn der Entwicklungsphase wurden, wie in den vorherigen Phasen auch, zuerst Aufgaben festgelegt und eine Aufwandsabschätzung durchgeführt. Die Aufgaben wurden in Jira eingepflegt und unter den Teammitglieder aufgeteilt.

Nach Abschluss der Phase wurden die tatsächlich benötigen Stunden in der Übersicht ergänzt. Für eine gute Übersicht der Abweichung wurden die echten Ergebnisse farblich hinterlegt. Eine grüne Hintergrundfarbe bedeutet, dass tatsächlich weniger Zeit benötigt wurde als ursprünglich geplant. Die gelbe Hintergrundfarbe wurde verwendet, wenn bis zu zwei Stunden länger gebraucht wurde. Wenn mehr als zwei Stunden länger benötigt wurde als geplant, wurde die rote Hintergrundfarbe gewählt. Bei den Aufgaben, wo die Abweichung größer als 2 Stunden war, beeinflusste es den geplanten Ablauf stark. Dies wird auch später in Gantt-Diagramm (Abbildung 8) ersichtlich.

Aufgaben		**geschätzte Stunden Aufwand**	**tatsächliche Stunden Aufwand**
Phase 3 - Entwicklung			
Implementier-ung	Unit-Tests planen und implementieren	5	5
	Methoden für spätere Gestenerkennung schreiben	2	3
	OSC-Kommunikation von Unity zu Controller implementieren	6	5
	OSC-Kommunikation vom Controller ans LightHouse implementieren	6	12
	LED-Patterns planen und implementieren	6	9
	Gestenerkennung Leap-Motion einbinden	6	10
	Software von localhost ins FH Netz migrieren	2	1
	"Schnittstelle" zu anderen Gruppen implementieren	4	2
Doku	Readme erstellen	1	1,5
	JavaDoc erstellen	2	2
	Code in Unity kommentieren	0,5	0,5
	Gantt-Diagramm	1	2
	Doku schreiben	5	5
Organisation	Jira Task überlegen und einpflegen	1	1,5
Gesamtaufwand		**47,5**	**59,5**

Für die Phase Entwicklung wurden 6 Mann-Tage Aufwand geschätzt. Tatsächlich benötigt wurde 7,5 Mann-Tage.

URSACHEN FÜR ABWEICHUNGEN

Die Methoden für die spätere Gestenerkennung wurden in dem Programm Unity programmiert. Unity scheint nicht mit Git kompatible zu sein, daher mussten die Entwickler andere Wege finden, um gemeinsam an dem Code zu arbeiten. Weitere Ausführung dazu sind unter dem Punkt *Git und Mergestrategien* zu finden.

Die OSC-Kommunikation zwischen Controller und LightHouse zu implementieren, hat doppelt so lange gedauert, wie geplant. Grund hierfür ist, dass der Code zuerst in Processing geschrieben wurde und das Team sich zu einem späteren Zeitpunkt dazu entschieden hat den Code nach Java zu migrieren. Daher musste der Code neu geschrieben werden. Der Entschluss wurde gefasst, da Java weiterverbreitet ist und somit die Erweiterbarkeit und Benutzbarkeit verbessert wird.

Die Planung und Implementation der LED-Patterns hat drei Stunden länger gedauert als angenommen, da die Umsetzung der Idee der Animation in tatsächlichen Code, komplexer war als gedacht. Jede LED muss mit einem eigenen Methodenaufruf angesteuert werden und bei langen und komplexen Animationen werden viele LEDs angesprochen. Als Hilfsmittel wurden Methoden geschrieben, die zum Beispiel eine ganze Spalte des LightHouses ansteuern.

Die Entwicklung des Programms für die Gestenerkennung, welche den Leap-Motion Controller einbindet, hat drei Stunden länger gedauert als erwartet. Grund hierfür ist, dass die Entwickler sich erst in die Dokumentation vom Leap-Motion Unity Plugin einlesen mussten.

Die Software in das Netz der Fachhochschule Kiel zu implementieren, ging schneller als geplant, da bei der Schätzung davon ausgegangen wurde, dass Fehler auftreten werden. Tatsächlich hat dieser Vorgang reibungslos funktioniert.

Die Implementation der Schnittstelle war leicht zu implementieren. Außerdem wurden vorab mit den anderen Gruppen schon Absprachen getroffen, was die Entwicklungszeit halbiert hat.

GANTT-DIAGRAMM

Die Aufwandsabschätzung wurde dem tatsächlichen Arbeitsaufwände in einem Gantt-Diagramm gegenübergestellt. An der Y-Achse sind die Aufgaben aufgelistet und an der X-Achse ist die Zeit in Stunden angegeben. Die orangenen Balken stellen den geschätzten Aufwand da. Der tatsächliche Aufwand wird durch die grünen Balken repräsentiert.

Alle Aufgaben sind voneinander abhängig, da sie aufeinander aufbauen. Die letzten drei Aufgaben (*Readme erstellen*, *JavaDoc*, *Code in Unity kommentieren*) laufen parallel und starten, nachdem die Schnittstelle implementiert wurde.

Bei einem Gantt-Diagramm lassen sich Abhängigkeiten der einzelnen Aufgaben gut visuell darstellen. Daher ist auch deutlich ersichtlich, dass vor allem die Aufgaben *OSC-Kommunikation vom Controller ans LightHouse implementieren*, *LED-Patterns planen und implementieren* und *Gestenerkennung Leap-Motion einbinden* zu Verzögerung von allen anschließenden Aufgaben beigetragen haben. Genau bei diesen drei Aufgaben ist der tatsächliche Aufwand in der Planungstabelle auch rot markiert.

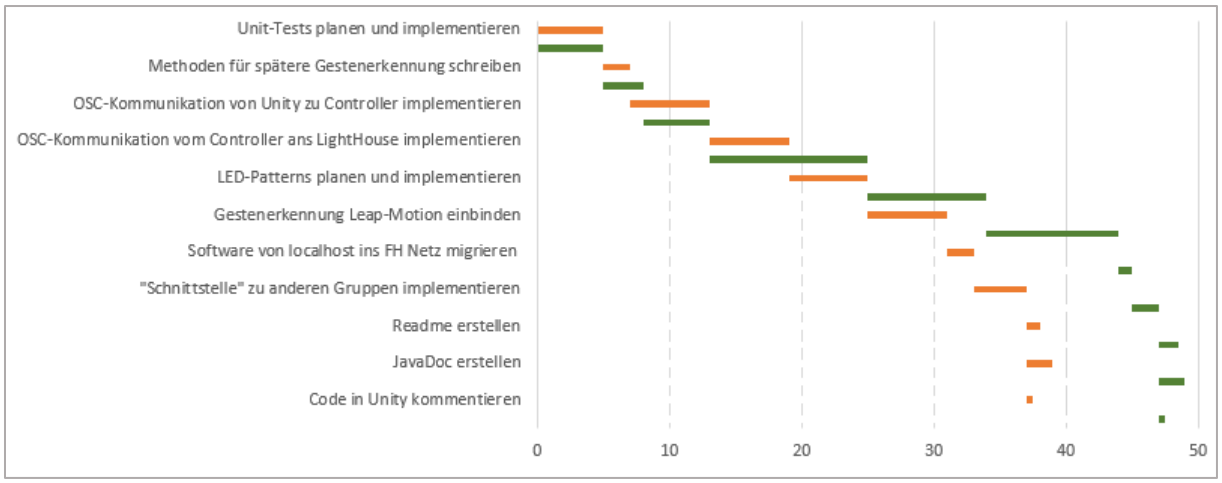


Abbildung 8 - Gantt-Diagramm mit geschätztem und tatsächlichem Aufwand

TEST-DRIVEN-DEVELOPMENT (TDD)

Durch die Verwendung von Test-Driven-Development wird ein permanentes Testen ermöglicht. Der Test prüft auch nach Umbau oder Neu-Implementierung einer Methode, ob diese noch funktionsfähig ist.

Die Implementierung des Codes lief nach dem Prinzip des Test-Driven-Development. Hierfür wurden Tests geschrieben, die sich an den Testfällen der Analyse-Phase orientieren. Der Code wurde iterativ anhand der Tests entwickelt. Nach Implementierung einer Methode wurde diese und der gesamte bisherige Code getestet.

Beispiel für Vorgehen bei Implementierung der Gestenerkennung:

- Das Programm löst eine Methode aus, wenn eine Geste erkannt wird. Also prüft der Test, ob eine Methode ausgeführt wurde.
- Dies wurde für die erste Geste durchgeführt und anschließend iterativ für die restlichen Gesten. Dabei waren die Tests für die vorangegangenen Gesten Teil des neuen Tests.

CODIERUNGSRICHTLINIEN

Bei der Implementierung wurde sich an dem in Modellierungs-Phase festgelegten Codierungs-Style orientiert. Ergänzend dazu wurde in einem *Readme* festgehalten, wie die Software zu bedienen ist. In einem *JavaDoc* wurden die Funktionalitäten der Methoden beschrieben. Dies führt zu einer hohen Wartbarkeit und Erweiterbarkeit.

GIT UND MERGESTRATEGIEN

Ein Problem, welches direkt am Anfang aufgetreten ist, ist die Nutzung von Git mit dem Programm Unity, da dieses sich nicht über Git teilen ließ. Dies erschwerte den Entwicklungsprozess, da nun der Code zwischen den Entwicklern hin- und hergeschickt wurde. Dies erschien dem Team nach ersten Versuchen zu unstrukturiert, da die Übersicht, welcher

der aktuelle Stand ist, verloren ging. Deshalb wurde auf die Methode des Pair-Programming gewechselt, was sehr gut funktionierte.

Bei dem Controller-Programm, welches bei einer Migration von Processing auf Java umgestellt wurde, traten keine Probleme auf und Git konnte genutzt werden. Es wurde sich für die Mergestrategie “Fast-Forward” entschieden. Hierbei wurde, wie es für diese Strategie üblich ist, für jegliche Features und Änderungen Branches erstellt und nicht auf dem main-Branch selbst gearbeitet. Auf diesen wurde lediglich zum mergen der Feature-Branches gewechselt.

FAZIT

Durch die Entwicklungs-Phase wurde verdeutlicht, wie wichtig es ist mit einem Tool zur Versionsverwaltung in der Entwicklung, wie zum Beispiel Git, zu arbeiten. Außerdem wurde gelernt, dass es effektiv ist Pair-Programming zu betreiben, da zwei Personen am selben Problem arbeiten und somit schneller eine Lösung gefunden werden kann.

PHASE 4 – TEST

VERANTWORTLICHER: Felix Seeburg

Für die Test wurde sich an dem *IEEE 829 Standard for Software Test Documentation* orientiert [3]. Der Standard teilt sich in drei wesentliche Punkte – die Teststrategie, die Testspezifikationen und den Testbericht.

TESTSTRATEGIE

Es ist ein Performancetest notwendig, um die Code Geschwindigkeit und Effizienz zu testen. Dieser ist besonders bei dem OCS-Controller zum Einsatz gekommen. Bei dem Performancetest muss ein beliebiges LED-Pattern aktivieren werden und auf das LightHouse projizieren werden. Im Anschluss wird die Geschwindigkeit des LED-Pattern mit der festgelegten Geschwindigkeit verglichen. Je nachdem, ob die Geschwindigkeit geringer oder gleich ist, ist der Test gescheitert oder wurde erfolgreich abgeschlossen.

Ein funktionaler Test, der durchgeführt wurde, ist der Black-Box-Test. Dabei wird ausschließlich die Funktion betrachtet und nicht der Code selber. Daher sind, wie in der untenstehenden Tabelle ersichtlich, zusätzliche Werkzeuge nicht erlaubt. Der Black-Box-Test wurde genutzt, um die Verbindung zwischen dem OCS-Controller und dem LightHouse zu testen. Dazu wurde ein beliebiges LED-Pattern auf dem LightHouse projizieren und geschaut, ob es korrekt angezeigt wird.

Zusätzlich wurde ein Strukturtest, ein White-Box-Test, durchgeführt. Bei diesem Test darf der Tester in den Code sehen. Daher sind, wie in der untenstehenden Tabelle ersichtlich, zusätzliche Werkzeuge erlaubt. Beim White Box-Test wurde mithilfe des Leap Motion-Controllers einige programmierte Gesten ausgeführt und geprüft, ob diese von Unity korrekt erkannt werden.

	Verantwortlicher	**Anzahl Testzyklen**	**Zusätzliche Werkzeuge**
Performancetest	Felix Seeburg	Bei jeder Änderung	Unity, Visual Studio, IntelliJ
BlackBox-Test	Alina Sakowski	Jedes neue Feature	/
WhiteBox-Test	Alexander Munkelt	Jedes neue Feature	Unity, Visual Studio

Außerdem sollen Code Reviews durchgeführt werden, um Schwachstellen und Verbesserungspotenzial zu entdecken.

In der nachfolgenden Tabelle befindet sich die Planung für die Code Reviews.

Datum	**Code**	**Reviewer**	**Verantwortlicher**
28.05.2022	Code in Unity	Paul von der Wehl, Alexander Neumann	Alexander Munkelt
13.06.2022	Code in Java	Felix Seeburg, Alina Sakowski	Alexander Munkelt

TESTSPEZIFIKATIONEN

Es wurden für jeden Test wurde eine Testspezifikation erstellt, die das Testziel, die Vorbedingungen, die Testbeschreibung und das erwartete Ergebnis festlegen. Außerdem wurden eine Identifizierung und Kategorisierung der Risikobereiche der Testfälle durchgeführt.

Für jeden Testfall aus der Analyse-Phase wurde eine solche Testspezifizierung und Risikoabschätzung durchgeführt, welche sich im Anhang befindet.

TESTBERICHT

Für die in den Testspezifikation beschrieben Test wurde die Testdurchführung dokumentiert. Diese befinden sich im Anhang.

Beispiel *Testdurchführung Schnelle Codeausführung und Anzeige auf LightHouse*:

- 04.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist nicht eingetroffen
    - Aufgrund eines unperformanten Codes wurden mehrere tausend OSC Messages an das LightHouse geschickt. Dies resultierte in einer Überlastung, welcher zu einer langsamen Übertragung am LightHouse führte.

REVIEW

Am 13.06.2022 haben die Reviewer Felix Seeburg und Alina Sakowski zusammen mit dem Programmierer Alexander Munkelt ein Code Review in Java durchgeführt. Dabei ist dem Reviewer aufgefallen, dass eine Methode, die dafür sorgt, dass das LightHouse beschrieben wird, hochgradig ineffizient ist und deshalb die Ausgabe auf dem LightHouse so langsam erfolgt. Mit der Hilfe des Programmierers konnte dieses Problem allerdings schnell behoben werden. Die Review für das Testen des Unity Codes befindet sich im Anhang.

Code Review Anforderungen	**Checked**
Clean Code	✓
Kommentare	✓
Gute Variablen / Methodennamen	✓
Unterordner	✓
Auslagerung in Klassen	✓
Gute Performance	X

UNITTEST

Da das Testen in Unity sich als komplizierter gestaltete als erwartet, wurde auf Reviews ausgewichen.

ABNAHMETEST

Der Abnahmetest wird vom Kunden vorgenommen. Dazu kann der Kunde sich der *Readme* zur Installation orientieren. Nach Erfolgreicher Installation kann getestet werden, ob die ausgeführten Gesten erkannt werden (/TF1/ und /TF2/) und entsprechend auf dem LightHouse ausgegeben werden (/TF5/).

AUFWANDSABSCHÄTZUNG & FAZIT

Der geschätzte Aufwand für die Test-Phase lag bei 2 Mann-Tagen. Die tatsächlich benötigte Zeit stimmt relativ genau mit der Aufwandsabschätzung überein.

Es wurde gelernt, dass es wichtig ist für jeden Test eine Testspezifikation zu entwerfen. Da dort feste Regeln für jeden Test festgelegt wurden, ergab sich somit strukturierte Herangehensweise an das Testen. Ergänzend dazu konnte das Testergebnis entsprechend bewertet werden. Durch die Test-Phase wurde ein besseres Verständnis für die Notwendigkeit von Tests erlangt.

ZUSAMMENFASSUNG

SELBSTEINSCHÄTZUNG

Das Projekt kann insgesamt als Erfolg gesehen werden. Es wurden bis auf ein paar Ausnahmen alle Kundenanforderungen erfüllt. Außerdem wurde ein funktionierendes, rundes Produkt entwickelt, das Aufmerksamkeit generiert. Nach einigen Startschwierigkeiten ist es dem Team gelungen, einen Workflow zu etablieren und sich koordiniert an die Aufgaben zu begeben.

AUFWANDSANALYSE

Das Team hat den Arbeitsaufwand für die einzelnen Schritte deutlich unterschätzt. Bei den Aufwandsabschätzungen wurden kleine, aber aufwendige Kleinigkeiten nicht mit einbezogen, was zu einer Differenz zwischen Schätzung und der tatsächlich gebrauchten Zeit führte. Die fehlende Nutzung von Git für den Parser verlängerte die Entwicklungszeit, weil auf Pair-Programming gewechselt werden musste und noch mehr parallel gearbeitet werden konnte. Das Formatieren und Finden des passenden Ausdrucks in der Dokumentation der Software nahm ebenfalls mehr Zeit in Anspruch als gedacht.

AUSBLICK

Das System hat viele Möglichkeiten zur Erweiterung in Bezug auf Animationen und Gesten. Für das LightHouse bestehen ohne Ende Möglichkeiten Animationen und Bilder darzustellen. Der Treiber des Leap-Motion Controllers verfügt über weitere, nicht belegte Standard-Gesten, welche sich einfach implementieren und einer Animation zuweisen lassen. Eigene Gesten können mit etwas mehr Aufwand auch selbst erstellt werden.

Des Weiteren können interaktive Elemente wie Minispiele realisiert werden, bei denen zum Beispiel ein statischer oder dynamischer Hintergrund existiert und sich bei einer Geste nur ein Teil des Bildes verändert und so ein interaktives Erlebnis entsteht.

VERWEISE

[1] Google Java Style Guide [online], Available: https://google.github.io/styleguide/javaguide.html#s4-formatting [Zugriff am 13.06.2022]

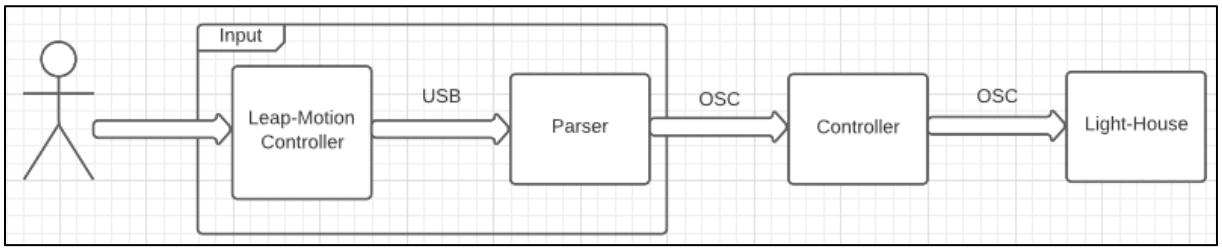
[2] [Unity] Coding guidelines & Basic Best Practices [online], Available: https://avangarde-software.com/unity-coding-guidelines-basic-best-practices/ [Zugriff am 18.06.2022]

[3] IEEE 829 Standard for Software Test Documentation [online], Available: https://standards.ieee.org/ieee/829/3787/ [Zugriff am 18.06.2022]

ANHÄNGE

I. Ergänzungen zum Bericht

Erster Grobentwurf für die Software:



User Stories:

/US1/ Als Nutzer möchte ich eine Geste ausführen, um Muster auf dem LightHouse darzustellen.

/US2/ Als Nutzer möchte ich, dass meine ausgeführte Geste erkannt wird, um Muster auf dem LightHouse darstellen zu können.

/US3/ Als Nutzer möchte ich, dass eine OSC-Nachricht an den Controller gesendet wird, um meinem Ziel Muster auf dem LightHouse darzustellen einen Schritt näher zu kommen.

/US4/ Als Nutzer möchte ich, dass eine OSC-Nachricht an den Controller gesendet wird, um meinem Ziel Muster auf dem LightHouse darzustellen einen Schritt näher zu kommen.

/US5/ Als Nutzer möchte ich, dass OSC-Nachrichten verarbeitet werden, um die LEDs von LightHouse anzusteuern.

/US6/1/ Als Nutzer möchte ich, dass die OSC-Nachrichten empfangen werden, um die richtigen Muster am LightHouse zu sehen.

/US6/2/ Als LightHouse möchte ich OSC-Nachrichten empfangen, um die richtigen Handlungen auszuführen.

/US7/ Als LightHouse möchte ich eine Welle darstellen, um den Nutzer zu erfreuen.

/US8/ Als LightHouse möchte ich eine Superman Animation abbilden, um den Nutzer zu erfreuen.

/US9/ Als LightHouse möchte ich eine Equalizer-Balken darstellen, um den Nutzer zu erfreuen.

/US10/ Als Nutzer möchte ich die Verbindung terminieren können, um anderen Nutzern die Bedienung des LightHouses zu ermöglichen.

Ausführliche Beschreibung der Anwendungsfälle

Die Anwendungsfälle leiten sich aus den oben genannten User-Stories ab.

/LF10/	**Titel:** Geste ausführen
	Kurzbeschreibung: Eine Geste wird über dem Leap Motion Controller ausgeführt.
	Akteur: Nutzer
	Vorbedingung: keine
	Beschreibung des Ablaufs:
	• Benutzer hält die Hand über den Leap-Motion Controller
	  • Er wischt nach rechts
	  • Er wischt nach links
	  • Er macht die "Mamma-Mia"-Geste (https://ergo-reiseblog.de/laendertipps/italienische-gesten) mit links
	  • Er macht die "Mamma-Mia"-Geste (https://ergo-reiseblog.de/laendertipps/italienische-gesten) mit rechts
	Auswirkungen: Das Programm zur Gestenerkennung hat einen Input.
	Anmerkungen: keine

/LF20/	**Titel:** Geste erkennen
	Kurzbeschreibung: Der Parser erkennt die Geste und stellt eine OSC-Nachricht zusammen.
	Akteur: Nutzer
	Vorbedingung: Geste wurde über dem Leap-Motion Controller ausgeführt
	Beschreibung des Ablaufs:
	• Der Parser verarbeitet die Geste
	  o Wischt er nach rechts, soll dies vom Input als das erkannt werden
	  o Wischt er nach links, soll dies vom Input als das erkannt werden
	  o Wird die "Mamma Mia"-Fingerbewegung mit der linken Hand ausgeführt, soll dies vom Input als das erkannt werden
	  o Wird die "Mamma Mia"-Fingerbewegung mit der rechten Hand ausgeführt, soll dies vom Input als das erkannt werden
	• Wird einer dieser Inputs erkannt, wird ein Alias für die Geste an eine OSC-Nachricht für den Controller angehangen
	Auswirkungen: Eine OSC-Nachricht mit einem Alias der Geste als Anhang kann gesendet werden.
	Anmerkungen: keine

/LF30/	**Titel:** OSC senden an Controller **Kurzbeschreibung:** Parser sendet eine OSC-Nachricht an den Controller **Akteur:** Nutzer **Vorbedingung:** • Parser hat einen Input des Leap Motion Controllers erhalten und eine OSC-Nachricht gebaut. **Beschreibung des Ablaufs:** • Parser sendet eine OSC-Nachricht **Auswirkungen:** Controller erhält einen Input. **Anmerkungen:** keine
/LF40/	**Titel:** OSC senden an LightHouse **Kurzbeschreibung:** Controller sendet eine OSC-Nachricht an das LightHouse

	Akteur: Nutzer **Vorbedingung:** • Controller hat einen Input vom Parser erhalten. **Beschreibung des Ablaufs:** • Controller sendet eine OSC-Nachricht **Auswirkungen:** LightHouse erhält einen Input. **Anmerkungen:** keine
/LF50/	**Titel:** OSC verarbeiten **Kurzbeschreibung:** Eine empfangene OSC-Nachricht wird auf ihren Inhalt analysiert, um folgende Handlungen zu evaluieren. **Akteur:** Nutzer **Vorbedingung:** OSC-Nachricht wurde empfangen **Beschreibung des Ablaufs:** • Der Controller prüft die OSC-Nachricht auf seinen Inhalt o Falls ein Pattern erkannt wird, wird die entsprechende Methode mit OSC-Nachrichten für das LightHouse aufgerufen • Das LightHouse prüft die OSC-Nachricht auf seinen Inhalt o Falls ein Pattern erkannt wird, werden die LEDs aus der OSC-Nachricht zum Leuchten gebracht **Auswirkungen:** • Der Controller ruft verschiedene Methoden auf, die OSC-Nachrichten für das LightHouse enthalten. • Das LightHouse lässt die LEDs leuchten, die in der OSC-Nachricht angesprochen und vom Programm erkannt wurden. **Anmerkungen:** keine
/LF60/	**Titel:** OSC empfangen **Kurzbeschreibung:** Es wird nach OSC-Nachrichten gelauscht. Diese werden empfangen, um weitere Handlungen durchführen zu können **Akteur:** Nutzer, LightHouse **Vorbedingung:** • OSC-Nachricht wurde von Parser oder Contoller gesendet • Netzwerkschnittstelle/Listener ist vorhanden **Beschreibung des Ablaufs:** • Prüfe, ob Sender eine OSC-Nachricht mit dem gewünschten Pattern gesendet hat. • Falls ja, nehme die Nachricht an. • Falls nicht, weiter lauschen. **Auswirkungen:** Die OSC-Nachricht kann analysiert werden und entsprechende Handlungen ausgeführt werden. **Anmerkungen:** keine
/LF70/	**Titel:** Welle darstellen **Kurzbeschreibung:** Das LightHouse bildet eine Welle ab. **Akteur:** LightHouse **Vorbedingung:** • Es wurde nach links oder rechts gewischt. • OSC-Nachrichten Pattern für Superman wurde von Controller ausgelöst und wurden vom LightHouse empfangen. **Beschreibung des Ablaufs:** Das LightHouse steuert die LEDs an **Auswirkungen:** Es wird eine sich bewegende Welle auf dem LightHouse abgebildet, solange bis eine neue Geste registriert wird.

	Anmerkungen: keine
/LF80/	**Titel:** Superman darstellen **Kurzbeschreibung:** Das LightHouse spielt eine Superman Animation ab. **Akteur:** LightHouse **Vorbedingung:** • Es wurde mit der rechten Hand die „Mamma Mia“ Geste ausgeführt. • OSC-Nachrichten Pattern für Superman wurde von Controller ausgelöst und wurden vom LightHouse empfangen. **Beschreibung des Ablaufs:** • Das LightHouse steuert die LEDs in der Reihenfolge an, wie die Nachrichten reinkommen. **Auswirkungen:** Die LEDs werden angesteuert und eine Superman Animation wird auf LightHouse dargestellt. **Anmerkungen:** keine
/LF90/	**Titel:** Equalizer-Balken darstellen **Kurzbeschreibung:** Das LightHouse bildet einen Equalizer ab. **Akteur:** LightHouse **Vorbedingung:** • Es wurde mit der linken Hand die „Mamma Mia“ Geste ausgeführt. • OSC-Nachrichten Pattern für Equalizer wurde von Controller ausgelöst und wurden vom LightHouse empfangen. **Beschreibung des Ablaufs:** • Das LightHouse steuert die LEDs in der Reihenfolge an, wie die Nachrichten reinkommen. **Auswirkungen:** Die LEDs werden angesteuert und es erscheinen Equalizer-Balken auf dem LightHouse. **Anmerkungen:** keine
/LF100/	**Titel:** Übertragungskanal absichern **Kurzbeschreibung:** Der Übertragungskanal wird abgesichert, um Überschneidungen beim Senden ans LightHouse zu verhindern. **Akteur:** Nutzer **Vorbedingung:** • Das OSC Signal wurde verarbeitet. **Beschreibung des Ablaufs:** • Bevor das OSC ans Light House gesendet wird, wird geprüft, ob der Übertragungskanal frei ist. **Auswirkungen:** Durch die Absicherung des Übertragungskanal kann es nicht zu Überschneidung beim Senden ans LightHouse kommen. Somit kann die Verbindung terminiert werden, wenn der Sendevorgang abgeschlossen ist, um andern Gruppen das Senden zu ermöglichen. **Anmerkungen:** keine

Testspezifikationen

- Testziel
  - Unbekannte Geste
- Vorbedingung
  - Leap-Motion-Controller muss angeschlossen sein
  - Unity läuft
- Beschreibung
  - Man führt eine Geste, welche noch nicht programmiert ist, über den Leap Motion Controller aus.
- Erwartetes Ergebnis
  - Leap Motion erkennt die Geste, aber Unity verarbeitet die Geste nicht und sendet somit auch keine OSC-Nachricht.

- Testziel
  - Bekannte Geste
- Vorbedingung
  - Leap Motion controller muss angeschlossen sein
  - Unity läuft
- Beschreibung
  - Man führt eine Geste, welche schon programmiert ist, über den Leap Motion Controller aus.
- Erwartetes Ergebnis
  - Leap Motion erkennt die Geste, Unity verarbeitet die Geste und sendet somit auch eine OSCI-Nachricht.

- Testziel
  - Bekannter OSC Pfad
- Vorbedingung
  - Unity läuft
  - OSC-Controller läuft
  - Lighthouse
- Beschreibung
  - Da wir zwei fälle von OSC-Nachrichten haben müssen auch beide getestet werden
    - 1.Fall
      - OSC Controller starten, um auf OSC Nachrichten zu warten.
      - Unity starten und eine OSC Nachricht an den OSC Controller senden.
    - 2.Fall
      - Lighthouse starten, um auf OSC Nachrichten zu warten.
      - OSC-Controller starten und eine OSC Nachricht an das Lighthouse senden.
- Erwartetes Ergebnis
  - 1. Fall: Nachricht kommt am OSC-Controller korrekt an
  - 2. Fall: das Lighthouse gibt das LED-Pattern aus welches erwartet wird

- Testziel
  - Unbekannter OSC Pfad
- Vorbedingung
  - Unity läuft
  - OSC-Controller läuft
- Beschreibung
  - Da wir zwei Fälle von OSC-Nachrichten haben müssen auch beide getestet werden
    - 1.Fall
      - OSC Controller starten, um auf OSC Nachrichten zu warten.
      - Unity starten und eine OSC Nachricht an den OSC Controller senden.
    - 2.Fall
      - LightHouse starten, um auf OSC Nachrichten zu warten.
      - OSC-Controller starten und eine OSC Nachricht an das LightHouse senden.
- Erwartetes Ergebnis
  - 1.Fall: OSC-Controller empfängt keine Nachricht trotz korrekter Absendung der OSC-Nachricht von Unity.
  - 2.Fall: das LightHouse gibt kein Led-Pattern aus

- Testziel
  - Richtiger Output am LightHouse
- Vorbedingung
  - Unity läuft
  - Leap Motion controller ist angeschlossen
  - OSC-Controller läuft
  - LightHouse läuft
- Beschreibung
  - Eine Geste über dem Leap Motion controller ausführen

Erwartetes Ergebnis
  - Anhand der Geste welche ausgeführt wurde erwartet man unterschiedliche LED patterns oder auch für den Fall einer Unbekannten Geste keine ausgabe am LightHouse

- Testziel
  - Schnelle Codeausführung und Anzeige auf LightHouse
- Vorbedingung
  - Unity läuft
  - Leap-Motion-Controller ist angeschlossen
  - OSC-Controller läuft
  - LightHouse läuft
- Beschreibung
  - Man führt eine Geste über den Leap-Motion-Controller aus.
- Erwartetes Ergebnis
  - Die korrekte Geste wird in erwarteter Geschwindigkeit korrekt ausgeführt

Identifizierung und Kategorisierung der Risikobereiche der zu testenden Software (* = Kritegrad * - ******)

Software:

- Unity
    - Kann keine Verbindung mit Leap Motion Controller herstellen.(**)
    - Kann keine Gesten erkennen. (*)
    - Kann keine OSC Nachrichten an den OSC Controller senden. (***)
- Java OSC Controller
    - Kann die empfangenen Daten nicht verarbeiten. (***)
    - Schickt falsche Daten ans LightHouse weiter. (*****)
    - Kann keine OSC Nachrichten senden. (****)
- LightHouse
    - Empfängt keine OSC Nachrichten. (****)
    - Empfängt zu viele OSC Nachrichten, um diese sinnvoll zu verarbeiten. (***)
    - Zeigt das LED-Pattern zu langsam an. (**)

Testdurchführung

Testdurchführung unbekannte Geste:

- 20.05.2022 (Unity) - Felix Seeburg - Erwartetes Ergebnis ist eingetroffen
- 13.06.2022 (Unity) - Fabian Gusek - Erwartetes Ergebnis ist eingetroffen

Testdurchführung bekannte Geste:

- 20.05.2022 (Unity) - Felix Seeburg - Erwartetes Ergebnis ist eingetroffen
- 13.06.2022 (Unity) - Alina Sakowski - Erwartetes Ergebnis ist eingetroffen

Testdurchführung bekannter OSC Pfad:

- 12.05.2022 (Processing) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen
- 04.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen

Testdurchführung unbekannter OSC Pfad:

- 12.05.2022 (Processing) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen
- 04.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen

Testdurchführung Richtiger Output am LightHouse:

- 02.05.2022 (Processing) - Fabian Gusek - Erwartetes Ergebnis ist nicht eingetroffen
    - Am Anfang haben wir versucht, mit einem in Processing geschriebenen Controller das LightHouse anzusteuern. Dies funktionierte jedoch in der ersten Iteration nicht, da wir den Aufbau der OSC Messages noch nicht verstanden hatten.
- 13.05.2022 (Processing) - Fabian Gusek - Erwartetes Ergebnis ist eingetroffen
- 04.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen

Testdurchführung Schnelle Codeausführung und Anzeige auf LightHouse:

- 13.05.2022 (Processing) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen
- 04.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist nicht eingetroffen
    - Aufgrund eines unperformanten Codes wurden mehrere tausend OSC Messages an das LightHouse geschickt. Dies resultierte in einer Überlastung, welche zu einer langsamen Übertragung am LightHouse führte.

- 13.06.2022 (Java) - Alexander Munkelt - Erwartetes Ergebnis ist eingetroffen

Review:

Am 28.05.2022 haben die Reviewer Paul von der Wehl und Alexander Neumann zusammen mit dem Programmierer Alexander Munkelt ein Code Review in Unity durchgeführt. Dabei ist den Reviewern aufgefallen, dass kaum Kommentare vorhanden waren.

Code Review Anforderungen	Checked
Clean Code	✓
Kommentare	X
Gute Variablen / Methodennamen	✓
Unterordner	✓
Auslagerung in Klassen	✓
Gute Performance	✓

II. Hinweise zu Jira

Alle Jira Task, die nicht zugewiesen sind, wurden in der Gruppe besprochen bzw. bearbeitet.

III. Aufwandsabschätzung

Tätigkeit		geschätzte Stunden Aufwand	tatsächliche Stunden Aufwand	Grund für Abweichung
Phase1 - Analyse				
Idee finden	Kennenlernen der Media IP Lab Geräte	2	4	Zutritt zum Media IP Lap erst verspätet möglich
	Definition Projektziel	1	1	
Hard- & Software kennenlernen	Leap-Motion Controller verstehen (Treiber, Funktionsweise)	2	4	hat länger gedauert bis Geräte verstanden wurden
	Software festlegen und kennenlernen	2	3	
Exposé schreiben	Lösungsidee beschreiben	0,5	0,5	
	Formulierung der Teilprobleme	0,5	0,5	
	Testfälle formulieren	1	1	

Lastenheft ausarbeiten	Use-Cases schreiben und Use-Case Diagramm erstellen	4	8	Umplanung auf Grund unklarer Wünsche des Kunden
	Anwendungsfälle ausformulieren	3	4	
	User-Stories verfassen	1	1	
	Produktdaten und Produktleistungen erkennen und formulieren	1	1	
	Zielbestimmung und Produkteinsatz formulieren	1	1	
	Produktübersicht erstellen	1	0,5	
	Qualitätanforderungen festlegen	1,5	1	
Organisation	Jira Task überlegen und einpflegen	1	1	
Gesamtaufwand		**22,5**	**31,5**	
Phase 2 - Design & Entwurf				
Grobentwurf	skizzieren	3	3	
	mit Programm ausarbeiten	2	2	
	OOSE erarbeiten und erstellen	0	8	wurde bei Planung vergessen
Feinentwurf	skizzieren, erste Überlegungen	2	2	
	Kontrolle, ob schlüssig	3	2,5	
	genaue Ausarbeitung	4	4,5	
Doku	Fein- und Grobentwurf einfügen	1	1	
	Diagramme und Abläufe beschreiben	3	4	
	Entscheidungen begründen	2	2	
Organisation	Jira Task überlegen und einpflegen	1	1,5	
	Abstimmung in Team	2	2	
Gesamtaufwand		**23**	**32,5**	
Phase 3 - Entwicklung				
Implementier-ung	Unit-Tests planen und implementieren	5	5	
	Methoden für spätere Gestenerkennung schreiben	2	3	
	OSC-Kommunikation von Unity zu Controller implementieren	6	5	
	OSC-Kommunikation vom Controller ans LightHouse implementieren	6	12	Aufwändiger da Migration zu Java, daher neu schreiben
	LED-Patterns planen und implementieren	6	9	Idee der Animation als Code auszudrücken, aufwändeigre als gedacht
	Gestenerkennung Leap-Motion einbinden	6	10	in die Dokumentation vom Leap-Motion Unity Plugin einlesen

	Software von localhost ins FH Netz migrieren	2	1	es wurde erwartet, dass Fehler auftritt, allerding ist kein Fehler aufgetreten
	"Schnittstelle" zu anderen Gruppen implementieren	4	2	leicht zu implementieren, da vorher in der Gruppe schon Absprachen getroffen wurden
Doku	Readme erstellen	1	1,5	
	JavaDoc erstellen	2	2	
	Code in Unity kommentieren	0,5	0,5	
	Gantt-Diagramm	1	2	
	Doku schreiben	5	5	
Organisation	Jira Task überlegen und einpflegen	1	1,5	
Gesamtaufwand		**47,5**	**59,5**	
Phase 4 - Test				
Test	Teststrategie überlegen	1	1	
	Checkliste für Code-Review erarbeiten	2	2	
	Testplan erstellen (wer wann mit wem welchen Code reviewed)	2	2	*2 Stunden, da im Team (4 Person erstellt je 0,5 Std)*
	Code-Review durchführen	2	2	
	notwendige Änderungen protokollieren	0,5	0,5	
	Testabdeckung protokollieren	1	1	
	Abnahmetest entwickeln	3	3	
Doku	Teststrategie ausformulieren	2	2	
	Dokumentation Code-Review	0,5	1	
	Dokumentation UnitTest	0,5	0,5	
	Dokumentation Abnahmetest	0,5	1	
Organisation	Jira Task überlegen und einpflegen	1	1	
Gesamtaufwand		**16**	**17**	

Legende:	keine Abweichung
	weniger Zeit benötigt, als geplant
	bis zu 2 Stunden länger gebraucht
	mehr als 2 Stunden länger gebraucht