

Wintersemester 2018/19

Fachbereich 4

Angewandte Informatik

MM Entwicklung von Multimediasystemen

Prof. Dr.-Ing. Johann Habakuk Israel

Sebastian Keppler

Interaktives Modellhaus

Vorgelegt von:

Gruppe Y

|  |  |
| --- | --- |
| Rico Eisenberg | s0561734 |
| Moritz Fabian Engers | s0559037 |
| Johann Jakob Gillhoff | S0559832 |
| Tony Nguyen Tien | S0559503 |

Berlin, den 28.01.2019

Inhalt

[Interaktives Modellhaus – Idee 2](#_Toc536213573)

[Einsatzmöglichkeiten 2](#_Toc536213574)

[Systembild 2](#_Toc536213575)

[Handgesten 3](#_Toc536213576)

[Kamera Positionen ändern 3](#_Toc536213577)

[Regen aktivieren 3](#_Toc536213578)

[Serielle Schnittstelle 3](#_Toc536213579)

[Sendende und Empfangende Struktur vom Arduino 4](#_Toc536213580)

[Sendet Arduino zu Unity 4](#_Toc536213581)

[Empfängt Arduino von Unity 4](#_Toc536213582)

[Umsetzung 4](#_Toc536213583)

[Erstellung des Realen Modellhauses 4](#_Toc536213584)

[Aufbau des Hauses 4](#_Toc536213585)

[Programmierung des Arduinos 4](#_Toc536213586)

[Empfang und Verarbeitung der Daten in Unity 5](#_Toc536213587)

[Erstellung des virtuellen Modellhauses in Unity 5](#_Toc536213588)

[Erstellung eines 3D Modells 5](#_Toc536213589)

[Auswirkungen auf das 3D Modell erstellen 6](#_Toc536213590)

[Veränderndes Wetter 6](#_Toc536213591)

[Erfassung der Leap Motion Gesten 6](#_Toc536213592)

[Controller: Der alles Verbindet 7](#_Toc536213593)

[Aufgetretene Probleme 7](#_Toc536213594)

[Gyroskop 7](#_Toc536213595)

[Git 7](#_Toc536213596)

[Probleme 8](#_Toc536213597)

[Quellen 8](#_Toc536213598)

# Interaktives Modellhaus – Idee

Wir haben uns überlegt, dass wir ein Modellhaus aus einem Werkstoff anfertigen und ein Abbild davon digital erstellen. Zudem sollte unser Modellhaus mit allerlei Sensoren versehen werden, so dass es über eine Schnittstelle mit unserem digitalen Abbild kommunizieren kann. So soll das Model mit seinem Abbild gegenseitig Interagieren können.

## Einsatzmöglichkeiten

Wir hatten angedacht dieses Projekt als ein früher Prototyp zu verwenden und diesen als Grundlage nehmen zu können um folgendes realisieren zu können:

1. Das Modellhaus kann man zentral in einem Smarthome einrichten und von dem Modellhaus aus das Smarthome steuern indem man z.B. durch eine Regen-Geste über dem aus allem Fenster und Türen schließt. Dabei stellt unser digitales Abbild das Smarthome da.
2. Ein Architekt könnte das Modellhaus als Modell eines reellen Smarthomes verwenden, um so das Haus zu planen oder eventuellen Kunden das noch nicht fertige Smarthome schonmal in Modellform greifbar zu präsentieren.
3. Kindern und jungen Jugendlichen könnte so spielerisch der Umgang mit Technik gezeigt und erklärt werden und Grundlegende Prinzipien könnten spielerisch vermittelt werden.



Handgesten

Serielle Schnittstelle

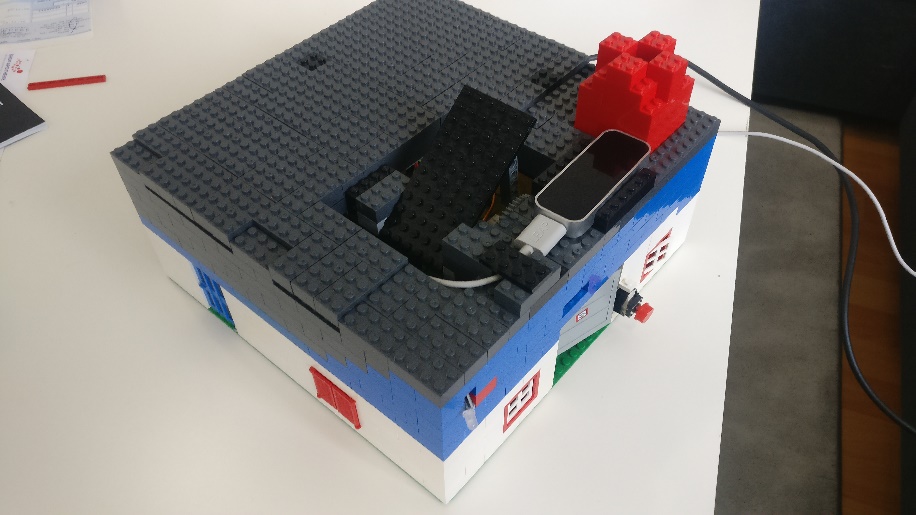


Abbildung 1 Systembild

# Systembild

## Handgesten

Es gibt zwei Möglichkeiten mit der Leap Motion zu interagieren:

### Kamera Positionen ändern

Die erste Möglichkeit dient zum Wechseln der aktiven Kamera in der Szene. Dies ermöglicht einem das Haus von verschiedenen Punkten aus zu beobachten. In der Szene befinden sich insgesamt 3 Kamera. Der Wechsel geschieht durch das Ausstrecken des Daumens, Zeigefingers und Mittelfingers der rechten Hand. Abhängig davon ob nur der Daumen, der Daumen und der Zeigefinger oder alle drei Finger ausgetreckt ist, wird entschieden welche Kamerasicht angezeigt wird.

### Regen aktivieren

Die zweite Interaktionsmöglichkeit dient zum Starten der Regenanimation im logischen Modell und das davon abhängige Schließen/Öffnen der Dachluke im physischen Modell. Falls die linke Hand mit der Handfläche nach unten zeigt und man seine Finger bewegt, startet die Animation. Damit der User nicht dauerhaft die Motion ausführen muss wie anfänglich geplant, haben wir uns dafür entschieden, dass man diese Bewegung nur einmal machen muss und die Animation gestartet wird. Dies bedingte eine weitere Motion zum Abbrechen der Regenanimation. Wir entschieden uns dabei für die linke Hand, bei der alle Finger ausgestreckt sind.

## Serielle Schnittstelle

* Tür → „potiTuer“
* Klingel → „klingelState“
* Kellertür Sonar → „sonarCM“
  + Kellertür Alarm → „isAlarm“
  + Blinklicht activ with isAlarm
* Magnet → „magnetState“
* Fenster → „lichtsensorState“
* Gyroskop → „x, y, z“
* Servo ← incomingByte

### Sendende und Empfangende Struktur vom Arduino

#### Sendet Arduino zu Unity

„potiTuer, klingelState, sonarCM, isAlarm, magnetState, lichtsensorState, x, y, z“

Die Werte werden als ganzer String übersendet an Unity.

#### Empfängt Arduino von Unity

incommingByte

Das Byte wird dabei als ASCII Zeichen interpretiert und daher ist eine übersendete „0“ eine 48 und wird so interpretiert. Daher muss von Unity einen „1“ oder „0“ übersendetet werden.

# Umsetzung

Um unser Projekt zu verwirklichen haben wir es untereilt umso produktiv in der Gruppe daran Arbeiten zu können. Dazu haben wir das Projekt folgender Maßen geteilt:

## Erstellung des Realen Modellhauses

### Aufbau des Hauses

Nachdem wir uns klar gemacht haben, welche Funktionen das Haus haben soll, planten wir wie wir das Umsetzten können und welche Sensoren und Motoren wir dafür bräuchten. Außerdem mussten wir uns noch entscheiden, aus welchem Material wir unser Haus fertigen wollen. Zur Diskussion standen Holz, Pape und Lego. Wir haben uns dann für Lego entschieden, da wir das Material zum einen Vorrätig hatten, außerdem zeichnet es sich durch seine leichte Verarbeitbarkeit und Vielseitigkeit aus.

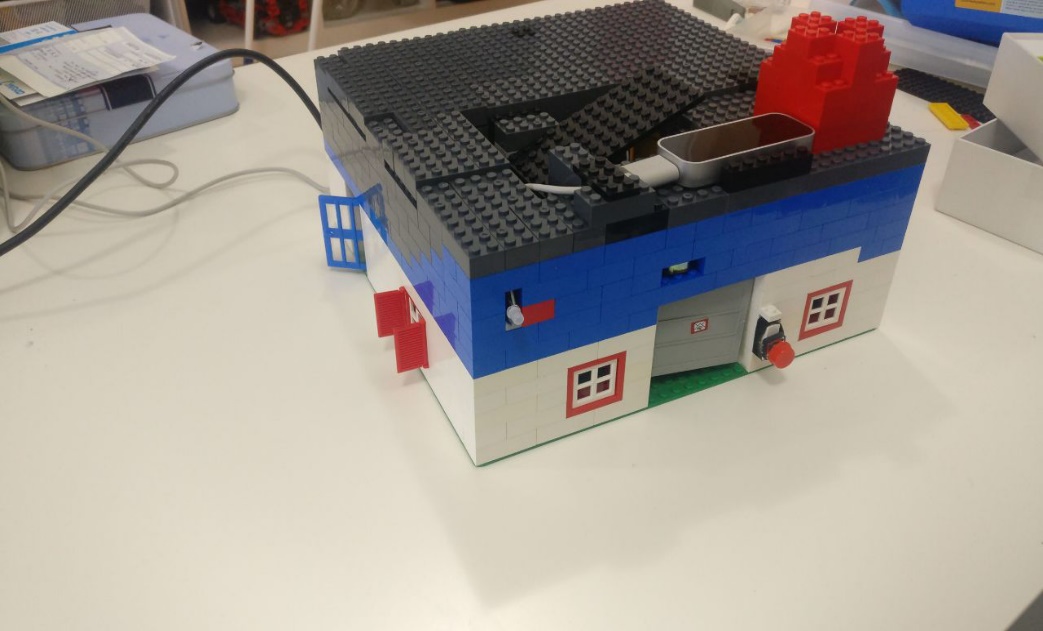


Abbildung 2 Reales Modellhaus

Nachdem alles Benötigte organisiert war, begannen wir das Haus aus Lego zu bauen und dabei versuchten wir bestmöglich die Sensoren, den Servo und die Leap einzubauen. Nachdem wir dann auch noch alle Sensoren und den Servo and den Arduino angeschlossen hatten und die Verkabelung dokumentiert hatten, begannen wir den Arduino zu programmieren.

### Programmierung des Arduinos

Die Programmierung des Arduinos verlief relativ reibungslos und schneller als erwartet. Dies lag unter anderem daran, dass in dem von uns bestelltem Sensor-Kit bereits Beispielcode zu jedem Bauteil beilag. Nach einiger Anpassung lief das meiste recht schnell und wir konnten wie in dem SL gelernt über die serielle Schnittstelle die Sensordaten über die Serielle Konsole senden. Unity empfängt und verarbeitet diese dann um die Änderungen am Modellhaus in Unity darzustellen.

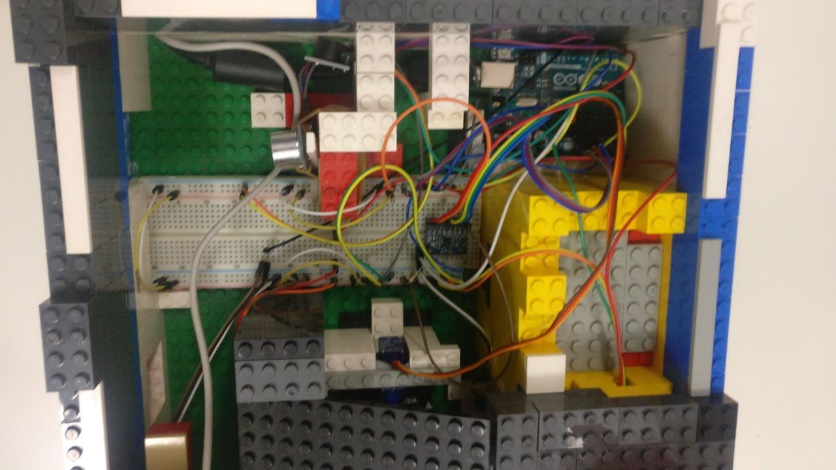


Abbildung 3 Reales Mellhaus Verkabelung

### Empfang und Verarbeitung der Daten in Unity

Da der serielle Empfang einen blockierenden Aufruf enthält, haben wir uns dazu entschlossen die serielle Kommunikation in einem eigenen Thread durchzuführen. So können wir sicherstellen, dass eine hohe FPS gewährleistet bleibt. Außerdem wird der Haupt Unity Thread nicht blockiert, wenn die Serielle Verbindung abbricht. Die gesamte serielle Kommunikation lagern wir in ein Script aus welches wir dem GameObject „SerialCommunicator“ hinzufügen. Dieses GameObject existiert nur einmal und kann von anderen Komponenten für die Kommunikation genutzt werden. Innerhalb des Scripts benutzen wir das C# volatile Keyword. Da wir auf die Attribute der Klasse von mehreren Threads aus zugreifen. Jede Zeile, welche vom Arduino über die Serielle Schnittstelle übertragen wird, liefert den gesamten State des Modellhauses. Diese Komma-Separierte Zeile wird aufgeteilt und die jeweiligen Werte werden aktualisiert.

## Erstellung des virtuellen Modellhauses in Unity

### Erstellung eines 3D Modells

Zur Erstellung eines virtuellen Abbildes des physischen Legohauses wurde die 3D-Grafiksoftware „Cinema 4D“ genutzt.

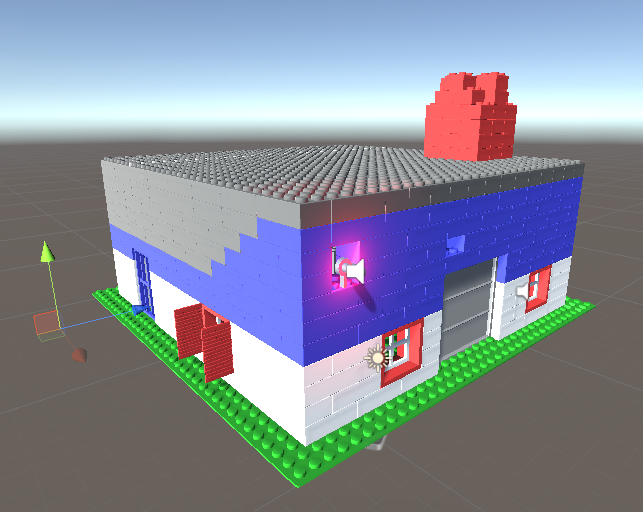


Abbildung 4 3D Modellhaus

Beim Bau des physischen Legohauses wurden im Wesentlichen 2 Arten von Legobausteinen verwendet: Bausteine mit 2x4 Noppen und Bausteine mit 2x2 Noppen auf der Oberfläche.

Diese beiden Arten von Bausteinen wurden zunächst als Polygon-Objekte in Cinema 4D nachgebildet. Die Polygon-Objekte bestanden somit abstrahiert aus einem Würfel mit 4 Zylindern bzw. einem Quader mit 8 Zylindern.

Diese Polygon-Objekte dienten im weiteren Verlauf als Referenz-Objekte für alle weitere Bausteine des virtuellen Legohauses, um ggf. Form, Polygonauflösung der Zylinder oder die Rundungen an den Kanten der Bausteine nachträglich für alle Instanzen flexibel anpassen zu können. Die Instanzen der Bausteine wurden entsprechend angeordnet, um die Mauern, das Flachdach und den Schornstein des 3D-Legohauses zu modellieren.

Danach wurden Polygon-Objekte für Fenster, Türen und Einrichtungsgegenstände erstellt und das 3D-Legohaus weitestgehend entsprechend der realen Vorlage texturiert.

Das fertige Modell wurde schließlich in das proprietäre FBX-Dateiformat der Firma Autodesk exportiert, um es in Unity nutzen zu können.

### Auswirkungen auf das 3D Modell erstellen

Nachdem das 3D Haus in Unity importiert war mussten wir noch die Umgebung auf den Userinput reagieren lassen. Dazu haben wir zunächst die Auswirkungen erschaffen, die der User auf das 3D Modell haben kann.

#### Veränderndes Wetter

Um Niederschlag in Unity zu realisieren, wurden 3 Game-Objects für Wolken, Nebel und Regen erstellt. Alle Objekte implementieren dabei jeweils eigene Particle-System-Instanzen, bei denen entsprechende Parameter gesetzt wurden, um das gewünschte Verhalten zu erzielen. Zudem wurden in Photoshop Grafiken als PNG‘s für Wolken, Nebel und Regentropfen erstellt, um sie für die Erstellung von Materialien in Unity zu nutzen. Diese Materialien wurden den Renderern der Particle-Systeme zugewiesen, um möglichst realistische Wolkenformationen und Regentropfenformen zu visualisieren.

### Erfassung der Leap Motion Gesten

Nun mussten wir uns noch darum kümmern, dass die Hände über dem reellen Haus erfasst werden. Zunächst wurde die Methode zur Feststellung ausgestreckter Finger entwickelt. Sie soll true zurückliefern, wenn besagte Finger ausgestreckt sind. Der Methode werden die getrackten Finger sowie eine Liste von Fingertypen, die ausgestreckt sein sollen, übergeben. Es wird anschließend geschaut, ob die Finger, die getrackt wurden, einem Fingertyp aus der Liste entsprechend und ausgestreckt sind mithilfe der Finger-Eigenschaft isExtended aus der Leap Motion API.

Das Umsetzen der zweiten Interaktionsmöglichkeit gestaltete sich schwieriger, da in der Leap Motion v4 keine vorgefertigten Funktionen zum Tracking von Bewegungen sich befinden wie in den vorherigen Versionen und somit selbst implementiert werden musste. Um dies zu realisieren wurden zunächst die Positionen aller Finger gespeichert. Im nächsten Frame wird anschließend überprüft, ob sich die Position der Finger geändert hat. Dieser Ablauf des Speicherns der Positionen im ersten Frame und der Bewegungsüberprüfung im zweiten Frame ist periodisch. Problem dieser Umsetzung war jedoch, dass eine Toleranzgrenze für jeden Finger ermittelt werden musste, weil die getrackten Positionen niemals gleichbleiben, unabhängig davon, ob es an Messungsungenauigkeiten der Leap Motion oder an minimale Bewegung der Hand durch Zittern lag. Die Geste, damit die Regenanimation gestoppt wird, wurde im Allgemeinen für den Kamerawechsel bereits implementiert.

Um die die nun implementierten Gesten zu optimieren bzw. zuverlässiger zu machen, mussten Restriktionen eingeführt werden: Falls eine der Gesten für das Wetter stattfand, werden diese Gesten nicht mehr überprüft für eine bestimmte Zeit. Dasselbe, aber mit einem anderen Zeitwert, für die Kamera. Des Weiteren wird die Kamera erst gewechselt, wenn die gleiche Geste für eine der Kameras mindestens 5 Frames anhielt. Dies verhindert, dass beim Herausziehen der Hand eine Geste erkennt werden kann.

### Controller: Der alles Verbindet

Damit bei einer Eingabe auch was passiert, müssen wir nun noch die passende Eingabe mit der dazu gehörigen Auswirkung Logisch in Unity verbinden. Dazu existiert in Unity ein Controller-Gameobject, welches eingehendenden Werte, zumeist einfache true- und false-Werte, logisch mit den passenden Repräsentationen verbindet. Sodass wenn sich z.B. in dem realen Haus die Tür öffnet sich auch in dem Unity-Modell diese Tür öffnet.

# Aufgetretene Probleme

## Gyroskop

Zunächst ließ sich das Gyroskop nicht auffinden in dem von uns bestellten Sensorenkit, da dieser ungünstig Positioniert war, sodass wir ihn erst eine Woche später als eigentliches Gyroskop identifizieren konnten und einbauen konnte. Da das Bauteil schließlich doch noch gefunden wurde, war es ein eher kleineres Problem.

Das größere war die Programmierung des Arduinos zum Auslesen der Daten des Gyroskops.

## Git

Wir hatten uns für GitHub entschieden um unseren Fortschritt uns gegenseitig zu teilen. Da wir alle noch recht unbewandert waren mit Git, hat uns das uns ein paar Mal ein paar Steine in den Weg geworfen. Wir konnten uns jedoch immer noch weiterhelfen.

# Quellen

|  |  |  |
| --- | --- | --- |
| Bezeichnung | Link | Für was benutzt? |
| Gyro | https://playground.arduino.cc/Main/MPU-6050 | Arduino Programmierung |
| Alarm Sound | https://www.youtube.com/watch?v=PowGPSdAxTI | Alarm Sound in der Unity-Szene |
| Klingel Sound | http://soundbible.com/1466-Doorbell.html | Klingel Sound in der Unity-Szene |