Wintersemester 2018/19

Fachbereich 4

Angewandte Informatik

MM Entwicklung von Multimediasystemen

Prof. Dr.-Ing. Johann Habakuk Israel
Sebastian Keppler

Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

Interaktives Modellhaus

Vorgelegt von:

Gruppe Y

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

Berlin, den 03.02.2019

Inhalt

Git Repository

Wir nutzten als Versionsverwaltungssystem Git und dabei ins besondere die Plattform GitHub:

https://github.com/NotoriousRonin/EMM-Project

Interaktives Modellhaus – Idee

Unsere Idee ist, ein Modellhaus aus einem Werkstoff anzufertigen und davon ein digitales Abbild zu erstellen. Das Modellhaus soll mit Sensoren versehen werden, so dass es über eine Schnittstelle mit seinem digitalen Abbild kommunizieren kann. So können das Modelhaus und sein digitales Abbild miteinander interagieren.

Einsatzmöglichkeiten

Der Prototyp des Projekts kann als Grundlage für folgende Realisierungen sein:

1. Das Modellhaus kann zentral in einem Smarthome aufgestellt werden und dieses steuern. Durch eine oberhalb des Modellhauses ausgeführte Regen-Geste könnte das Smarthome z. B. angewiesen werden, alle Fenster und Türen zu schließen. Dabei stellt unser digitales Abbild das Smarthome dar.
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 in Modellform greifbar zu präsentieren.
3. Kindern und jungen Jugendlichen könnte mit dem Projekt spielerisch der Umgang mit Technik gezeigt und erklärt, grundlegende Prinzipien so spielerisch vermittelt werden.
4. Das Modellhaus könnte als eine Art Controller verwendet werden. Dabei bietet es vielfältige eingebe Möglichkeiten.

Systembild

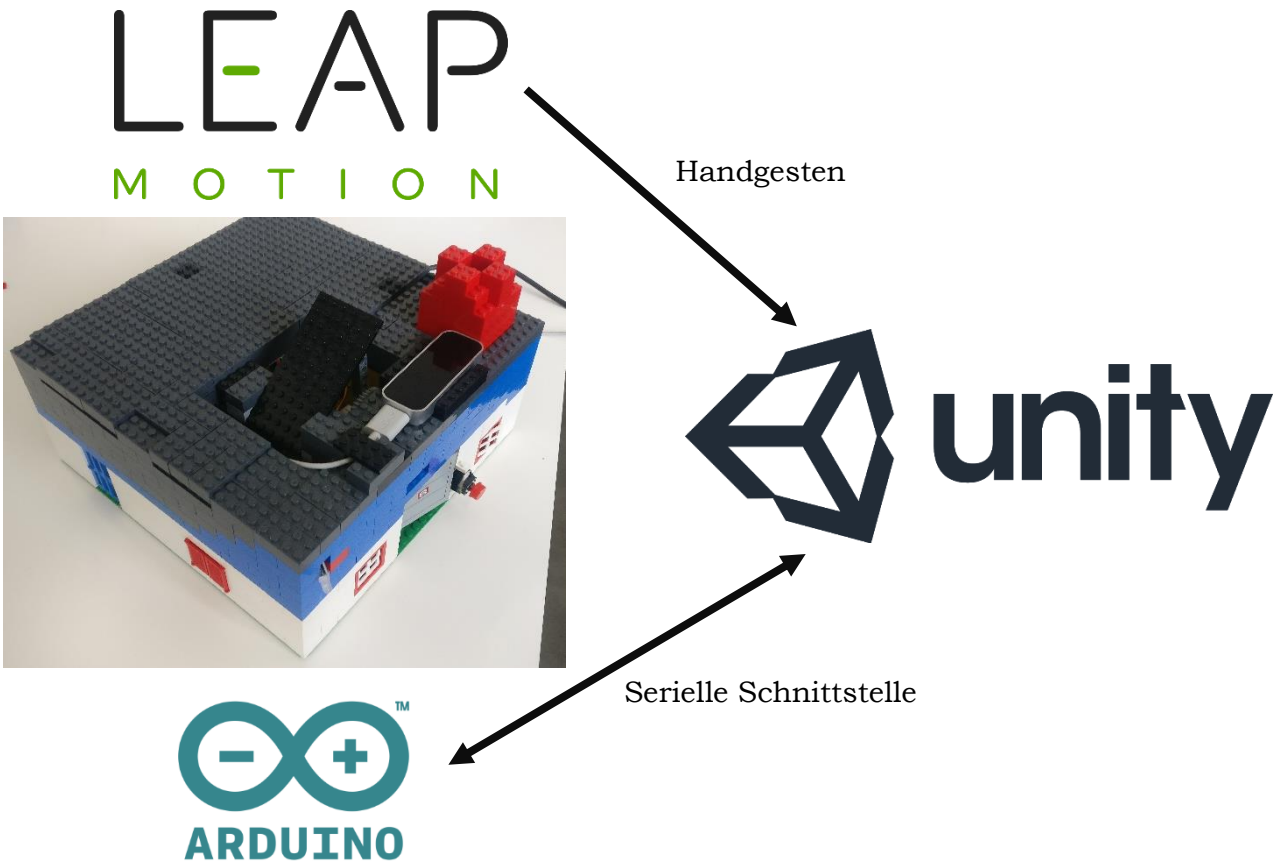

Abbildung 1 Systembild

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, das Haus von verschiedenen Seiten zu beobachten. In der Szene befinden sich insgesamt 3 Kameras. 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 sind, 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. Wenn die linke Hand mit der Handfläche nach unten zeigt und

die Finger bewegt werden, startet die Animation. Damit der User nicht dauerhaft die Motion ausführen muss, was anfänglich geplant war, haben wir uns dafür entschieden, dass die Animation bereits mit der einmaligen Ausführung der beschriebenen Bewegung gestartet wird. Dies bedingte allerdings eine weitere Motion zum Abbrechen der Regenanimation. Wir entschieden uns insoweit wieder für die linke Hand – nunmehr mit allen ausgestreckten Fingern.

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 an Unity übersandt.

Empfängt Arduino von Unity

incommingByte

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

Umsetzung

Um effektiv in der Gruppe an dem Projekt arbeiten zu können, haben wir es untereilt:

Erstellung des Realen Modellhauses

Aufbau des Hauses

Nach der Festlegung, welche Funktionen das Haus haben soll, planten wir die Umsetzung. Wir prüften, welche Sensoren und Motoren benötigt werden und welches Baumaterial sich für das Projekt am besten eignete. Zur Diskussion standen Holz, Pappe und Lego. Wir haben uns für Lego entschieden, weil wir das Material vorrätig hatten und es sich durch seine leichte Verarbeitbarkeit und Vielseitigkeit auszeichnet.

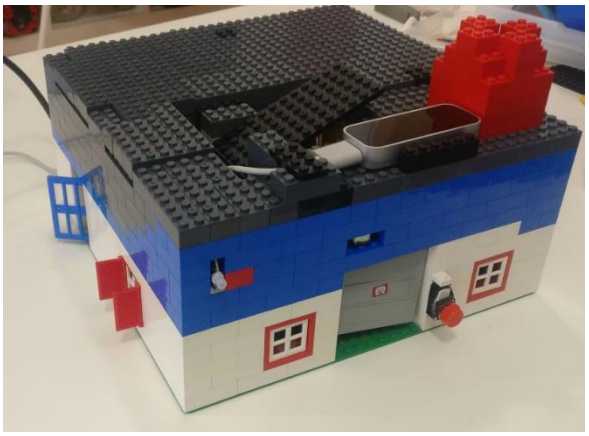

Abbildung 2 Reales Modellhaus

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

Programmierung des Arduinos

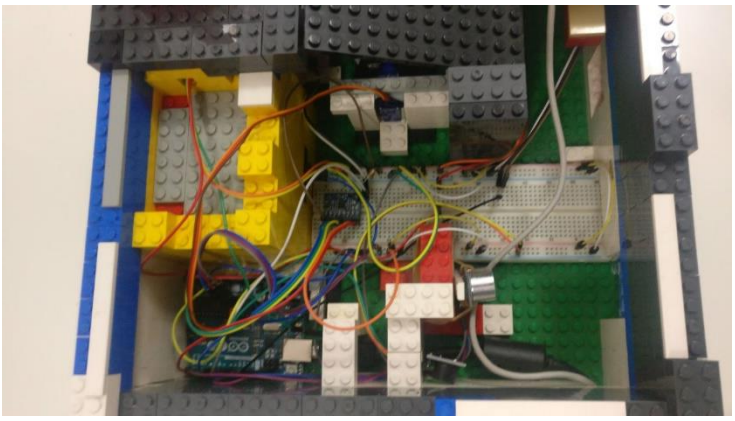

Abbildung 3 Reales Modellhaus Verkabelung

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. Der schwierigste Teil war das Gyroskop zum laufen zu bekommen. Als wir endlich Werte des Gyroskops über die serielle Schnittstelle ausgeben konnten, mussten wir die Unterschiedlichen Achsen-Modelle von Unity und des Gyroskops zusammenbringen. Durch die Werte und einigem ausprobieren ergab sich das die Y-Achse in Unity der Z-Achse im Gyroskop entspricht. Dies haben wir dann im Unity Skript berücksichtigt. Um die Rotation des Hauses zu berechnen, ziehen wir die Werte des Gyroskops von der Startrotation ab. Uns viel allerdings auf das die Werte des Gyroskops für die Drehung (Y-Achse in Unity) zu sehr schwankten. Daher entschlossen wir uns die Rotation komplett außer Acht zu lassen und nur die Neigung des Gyroskops zu benutzen.

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 („SerialCommunicator„) aus, das wir dem GameObject „Controller“ 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.

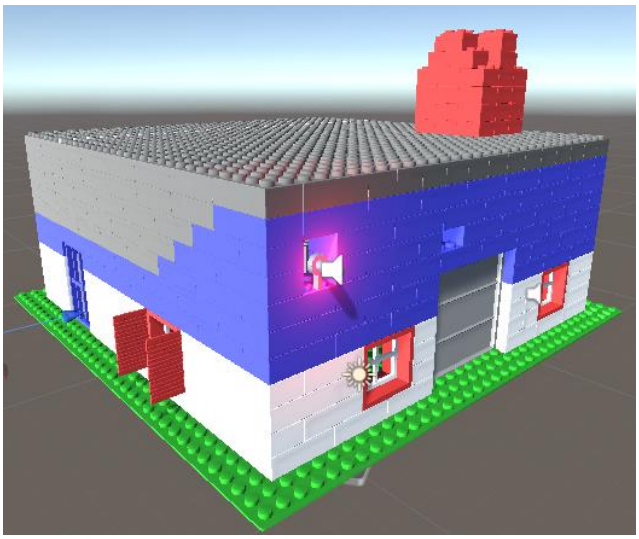

Abbildung 4 3D Modellhaus

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 Hause 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 mithilfe der Finger-Eigenschaft isExtended aus der Leap Motion API geschaut, ob die Finger, die getrackt wurden und einem Fingertyp aus der Liste entsprechen, ausgestreckt sind.

Das Umsetzen der zweiten Interaktionsmöglichkeit gestaltete sich schwieriger, da sich in der Leap Motion v4, anders als in den vorherigen Versionen, keine vorgefertigten Funktionen zum Tracking von Bewegungen und somit selbst implementiert werden mussten. 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 minimalen Bewegungen der Hand etwa durch Zittern lag. Die Geste, damit die Regenanimation gestoppt wird, wurde im Allgemeinen für den Kamerawechsel bereits implementiert.

Controller: Der alles verbindet

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

Aufgetretene Probleme

Git

Als Versionsverwaltungssystem haben wir uns für GitHub entschieden, um unsere Fortschritte gegenseitig zu teilen. Da wir alle mit Git noch recht unerfahren waren, hat uns das uns ein paar Mal ein paar Steine in den Weg gelegt. Wir konnten uns jedoch immer noch weiterhelfen.

Quellen

Bezeichnung	Link	Für was benutzt?	Letzter Zugriff
Gyro	https://playground.arduino.cc/Main/MPU-6050	Arduino Programmierung	15.01.2019
Alarm Sound	https://www.youtube.com/watch?v=PowGPSdAxTI	Alarm Sound in der Unity-Szene	15.01.2019
Klingel Sound	http://soundbible.com/1466-Doorbell.html	Klingel Sound in der Unity-Szene	15.01.2019