|  |
| --- |
| Eigene Lösung  Analyse  **Julien Villiger**  **v1.1 18.03.2015** |
| **Berner Fachhochschule**  Technik und Informatik |

Inhaltsverzeichnis

[1 Idee 3](#_Toc414480360)

[1.1 Unity 3](#_Toc414480361)

[1.2 Eigenes Protokoll 3](#_Toc414480362)

[2 Architektur 4](#_Toc414480363)

[2.1 Aufgabe Server 4](#_Toc414480364)

[2.2 Aufgabe Client 4](#_Toc414480365)

[3 Funktionale Anforderungen 5](#_Toc414480366)

[3.1 Multi-Monitor Displays 5](#_Toc414480367)

[3.2 Stereoskopie 5](#_Toc414480368)

[3.3 Implementierung bestehender Unity Anwendungen 5](#_Toc414480369)

[4 Argumentation 6](#_Toc414480370)

[4.1 Pro 6](#_Toc414480371)

[4.1.1 Keine Abhängigkeit 6](#_Toc414480372)

[4.1.2 Community 6](#_Toc414480373)

[4.1.3 Netzwerkauslastung 6](#_Toc414480374)

[4.2 Kontra 6](#_Toc414480375)

[4.2.1 From Scratch 6](#_Toc414480376)

[4.2.2 Aufwand 6](#_Toc414480377)

[5 Zusammenfassung 7](#_Toc414480378)

[6 Literaturverzeichnis 8](#_Toc414480379)

[7 Versionskontrolle 8](#_Toc414480380)

# Idee

Eine Variante besteht darin, auf Frameworks von Drittparteien zu verzichten und eine eigene Lösung zu entwickeln. Dieser Ansatz wird nur in Erwägung gezogen, falls sämtliche Möglichkeiten mit einem bestehenden System als unzureichend oder nicht umsetzbar eingestuft wurden.

Eines der Hauptprobleme wird die Synchronisierung des Servers und der verschiedenen Clients sein. Sobald Scripts Zufallskomponenten beinhalten, darf die Berechnung des Verlaufs des Spiels oder der Simulation nur zentral an einem Ort geschehen. Ansonsten sind die Stationen nicht mehr synchron und es kann kein einheitliches Bild mehr im CAVE dargestellt werden.

Zwei Ansätze sind in Betracht zu ziehen.

## Unity

Unity bietet Module, um Multiplayerspiele zu entwickeln. Das Problem der Synchronisierung tritt vor allem in diesem Bereich auf und ist somit ein zentrales Anliegen der Unity-Entwickler und Anwender. Auf deren Erfahrung und der bereits umgesetzten Module kann zurückgegriffen werden.

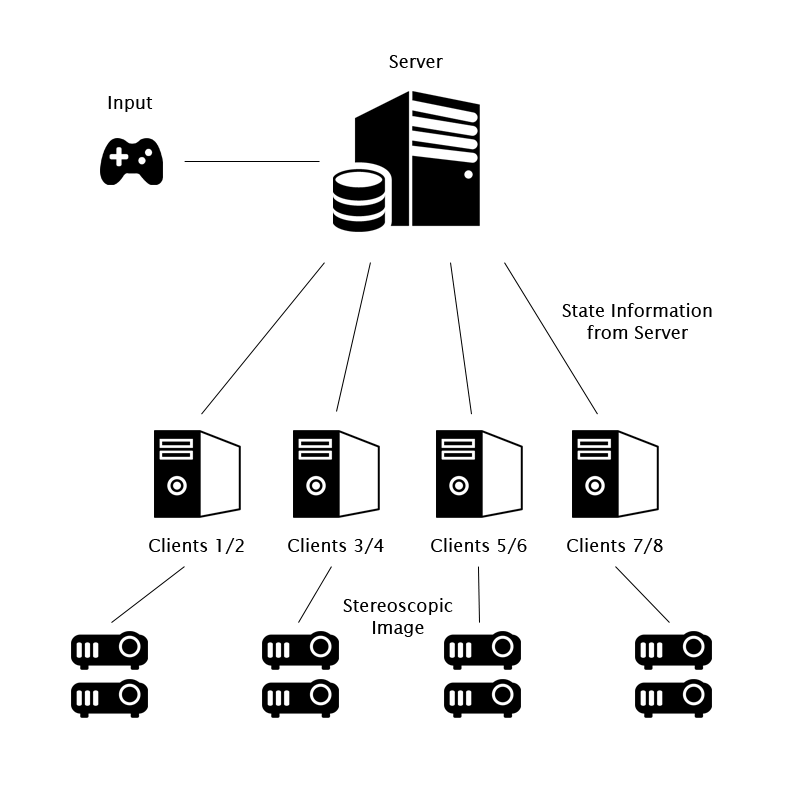
## Eigenes Protokoll

Falls Unity mit den Standardfunktionen zu wenige Möglichkeiten bietet und auch im Asset Store keine hilfreichen Bibliotheken vorhanden sind, müsste ein eigenes System zum Synchronisieren erarbeitet werden, welches die Kommunikation zwischen den Clients und dem Server unabhängig von Unity betreibt.

Zum Einsatz käme das User Datagram Protocol (UDP), welches die nötigen Informationen vom Hauptserver an die Clients schickt.

# Architektur

Das geplante System basiert auf einer Client-Server Architektur. Die Hauptinstanz der Unity Anwendung wird auf einem eigenen Server laufen, welche den Rendering-Clients die nötigen Informationen zukommen lässt, um die Synchronisierung zu gewährleisten.



Grafik 1, Client-Server Architektur, Quelle: <http://www.flaticon.com/>

## Aufgabe Server

Die Hauptinstanz berechnet den Ablauf des Spiels oder der Simulation. User-Inputs werden hier verwaltet und entsprechende Geräte sind deshalb an diesem Rechner angeschlossen.

Nachdem die Anwendung gestartet wurde, ist sie bereit für die regelmässige Auslieferung der Informationen zum Synchronisieren der Clients und wartet auf deren Verbindungsaufbau.

Die Position der Kamera und der aktuelle State des Spiels oder der Simulation werden laufend an die Clients übermittelt, damit die Darstellung im CAVE entsprechend angepasst werden kann.

## Aufgabe Client

Sobald der Server gestartet wurde, können sich die Clients beim Server anmelden und erhalten in regelmässigen Abständen Informationen über den aktuellen Status des Spiel-, bzw. Simulationsablaufs. Weil jeder Client für eine Sicht (ein Auge) auf eine Leinwand im CAVE zuständig ist, wird der Bildausschnitt dementsprechend reduziert und die Bildinformationen werden für die stereoskopische Darstellung auf einen Projektoren geschickt. Jeder Client rendert jeweils nur eine Sicht der Anwendung, so dass zwei Clients zusammen die 3D Illusion, mit je einem eigenen Projektor, darstellen können.

Der Ablauf des Spiels oder der Simulation wird hier nicht berechnet. Lediglich die Informationen, die nötig sind um die virtuelle Welt darzustellen, werden empfangen und interpretiert.

# Funktionale Anforderungen

## Multi-Monitor Displays

Weil jede Leinwand nur einen Viertel des kompletten Bildes darstellt, liefert jeder einzelne Client selektiv Bildinformationen. Die erarbeitete Lösung sollte also in der Lage sein, basierend auf den Einstellungen des Clients, nur den entsprechenden Bildanteil zu projizieren.

## Stereoskopie

Nicht-Stereoskopische Anwendungen sollten in Stereoskopische umgewandelt werden können, wobei passive (Polarisierte Lichtprojektion) Stereoskopie unterstützt wird. Unity unterstützt diese Funktion.

## Implementierung bestehender Unity Anwendungen

Eine eigene Unity Anwendung in den CAVE zu implementieren sollte mit wenigen Clicks möglich sein. Die Rendering-Clients sind bereits für die jeweiligen Leinwände vorkonfiguriert. Es gilt lediglich, die erstellte Lösung in eine Unity 4.6 Anwendung mit vorhandenem Sourcecode einzubinden. Das Unity Projekt wird anschliessend für Windows exportiert und auf die Clients / den Server verteilt.

Nachdem die Anwendung auf dem Server gestartet wurde, können die Clients einzeln den Bootvorgang durchlaufen und sich mit dem Server verbinden, um die nötigen Informationen zur Darstellung der Projektion zu erhalten.

# Argumentation

## Pro

### Keine Abhängigkeit

Weil das gesamte System eine Eigenentwicklung ist, bestehen bis auf die Verbindung zu Unity und der benutzten Infrastruktur keinerlei Abhängigkeiten. Die maximale Flexibilität ist somit gewährleistet. Ausserdem entfällt das Einarbeiten in ein bestehendes Framework.

### Community

Unity ist ein sich stetig weiterentwickelndes Framework und geniesst eine immer grösser werdende Community, welche sich aktiv in Foren beteiligt. Sollten Probleme auftauchen, ist das Internet eine grosse Informationsquelle, die täglich an neuen Informationen reicher wird.

### Netzwerkauslastung

Durch die Client-Server Architektur, bekannt aus Multiplayerspielen, die auch übers Internet gespielt werden, stellt das interne Netzwerk keinen Flaschenhals dar. Die nötigen Informationen für die Clients sind lediglich der Status des aktuellen Hauptspielablaufs.

## Kontra

### From Scratch

Weil auf keine Bibliothek oder Framework zurückgegriffen werden kann, muss alles von Grund auf selber programmiert werden. Lediglich die Funktionen von Unity können und sollten Verwendung finden.

### Aufwand

Der Aufwand und die möglichen Probleme sind schwer abzuschätzen. Das gesamte System der Synchronisierung, der Stereoskopie und des Einpflegens in den CAVE muss geplant, umgesetzt und Debugged werden. Die Verwendung eines fertigen Frameworks, welches von einem Entwicklerteam stammt und sich in der Praxis bewährt hat, kann auf eine Robustheit zurückgreifen, die bei einer eigenen Entwicklung nicht per se gegeben ist.

# Zusammenfassung

Einen eigenen Lösungsansatz zu verfolgen besticht durch seine Flexibilität. Mit der Game-Engine Unity wird auf das richtige Pferd gesetzt und ist zukunftsorientiert. Der Aufwand, ein eigenes Produkt zu erstellen ist im Gegensatz zur Implementierung eines fertigen Frameworks um einiges höher und birgt Gefahren. Deshalb ist die sorgfältige Analyse der bestehenden und geprüften Lösungen entscheidend. Erst wenn durch stichhaltige Argumente die anderen Möglichkeiten ausgeschlossen werden können, wird dieser Ansatz weiterverfolgt. Die Implementierung mittels einer eigenen Lösung ist aber, mit entsprechendem Aufwand, vielversprechend.

# Literaturverzeichnis

**Unity Website***(http://unity3d.com/)*

**Flaticon Database**  
(http://www.flaticon.com/)

# Versionskontrolle

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Beschreibung** | **Autor** |
| 1.0 | 18.03.2015 | Dokument erstellt + erster Entwurf | Julien Villiger |
| 1.1 | 18.03.2015 | Dokument überarbeitet und fertiggestellt | Julien Villiger |