Privatsphärenerhaltendes Stadtmodell mit Ereignishervorhebung in Virtual Reality

Nutzendendokumentation

Alexander Link
Christian Goebel
Hans André Affeldt
Moritz Vanderheyden
Sven Brielmayer

Das Programm
Bei dem Privatsphärenerhaltenden Stadtmodell mit Ereignishervorhebung in Virtual Reality (kurz auch Stadtmodell oder City Model) handelt es sich um ein Programm zur Visualisierung von Entitäten und Ereignissen in einer virtuellen 3D-Umgebung. Das Projekt wurde im Rahmen eines Bachelorpraktikums der TU-Darmstadt im Wintersemester 2020/21 im Auftrag des Telekommunikations-Lab entwickelt.
In der Visualisierung ist es für Nutzende möglich, sich frei in Darmstadt zu bewegen. Darüber hinaus ist es möglich die Zeit der Simulation zu beeinflussen. Dies beinhaltet die Funktionen des Beschleunigen, Verlangsamen und Anhalten der Zeit, sowie das Springen an einen der Simulation bekannten Zeitpunkt. Ein solches Ereignis befindet sich im Fall des Auslesens einer fertigen Datei innerhalb der zeitlichen Grenzen des ersten und des letzten Eintrags der Datei. Im Fall des Einlesens über einen Netzwerkstream befindet er sich zwischen dem Zeitpunkt des ersten erhaltenen Datums und dem des Letzten.
Schließlich ist es noch möglich, geschehene Ereignisse innerhalb der Simulation zu erkennen, ihre Zeitpunkte auf der Zeitskala zu finden und an diese zu springen. Springt man zeitlich an ein Ereignis, so wird die Wiedergabegeschwindigkeit automatisch angehalten. Nun ist es möglich, durch umsehen oder nutzen eines Radars das Ereignis ausfindig zu machen und sich zu ihm zu bewegen.

Steuerung
In VR: Zur Steuerung der Visualisierung in Virtual Reality wird ein vollständiges Virtual Reality Equipment Set benötigt. Bewegung ist dabei durch Teleportation möglich. Andere Interaktionen erfordern das Aufnehmen der Bedienungs-Controller innerhalb von Unity(zu finden wenn man nach unten sieht).

- Hintere Griptaste zum Aufnehmen der Bedienungs-Controller
- Seitliche Griptaste zum Feststellen der Bedienungs-Controller an der momentanen Position
- Radar controller
    - Zeigt ein event radar
- UI Controller Controller
    - zeigt die momentaner Wiedergabegeschwindigkeit an
    - Joystick Horizontale beeinflusst die Wiedergabegeschwindigkeit
    - Joystick Vertikale beeinflusst die angezeigte Detailstufe
    - Joystick drücken pausiert die Wiedergabe oder setzt sie fort
- Time UI Controller
    - Zeigt UI mit Zeitleiste und Event combobox an
    - Interaktion mit dem Laserpointer möglich (hintere Griptaste)
    - Ereignisse für Zeitsprünge in Combo-Box

Desktop: Nutzt man die Software ohne vorhandenes VR-Equipment, so ist eine vollständige Bedienung mit Maus und Tastatur möglich. Bewegung ist mit den Tasten W, A, S und D möglich, der Nutzende ist jedoch nicht mehr an den Boden gebunden und kann daher fliegen. Beschleunigen kann er durch die Shift-Taste. Umsehen geschieht durch das Bewegen der Maus. Manuell zwischen den Detailstufen kann der Nutzende mit der Taste C wechseln (0 = LoD, 1 = Abstrakte Darstellung, 2 = unidentifizierbare Entitäten, 3 = identifizierbare Entitäten). Die Zeitmanipulation befindet sich vollständig auf der dauerhaft sichtbaren graphischen Oberfläche. In der Zeitleiste an dem unteren Bildschirmrand sind bisherige Ereignisse sichtbar, das Klicken auf eine Position innerhalb der Zeitleiste oder auf

eines der Ereignisse setzt die Zeit auf diesen Zeitpunkt. Alternativ kann zu den Ereignis-Zeitpunkten auch mit Hilfe der Combo-Box in der oberen linken Ecke gesprungen werden. Mit den Buttons am oberen Bildschirmrand kann die dort angezeigte Wiedergabegeschwindigkeit angepasst werden.

Die Input-Dateien: Die Stellschraube zur Nutzung alter und damit vorhandener Input Dateien ist mit Hilfe der Unity-Oberfläche ansteuerbar. Das Deaktivieren der Checkbox *Use Stream* im Manager-Skript des Szenenobjekts *ManagerObject* ermöglicht das Einlesen der Input-Daten via Datei statt über den Netzwerkstream. Dafür müssen die BIN-Datei der Entitäten und die CSV-Datei der Ereignisse in den Feldern *File Path* und *Event Filename* eingetragen werden. Die dazugehörigen Dateien müssen sich im Ordner Assets/CSVInput befinden. Die Sensorliste ist händisch zu pflegen. Verlinkt wird sie im *Sensorhandler* - hier ist auch die Wahl einer anderen Datei möglich. Einträge (=Zeilen) können problemlos entfernt werden, jedoch werden Entitäten, die keine legitime SensorID besitzen nicht angezeigt. Einträge können hinzugefügt oder bearbeitet werden, müssen dabei allerdings folgende durch Semikolons getrennte Daten in gezeigter Reihenfolge beinhalten:

- SensorID: Eine einzigartige Ganzzahl, innerhalb des 32-Bit Integer Rahmens
- 3x Position: Die globalen Koordinaten des Sensors innerhalb der Unity Szene - Gleitkommazahl mit Punkt oder Komma möglich
- 3x Rotation: Die Ausrichtung des Sensors - Gleitkommazahl mit Punkt oder Komma möglich

Zur Bestimmung von Rotation und Position ist es empfehlenswert ein Kameraobjekt an die Position in der Szene zu bewegen, welche jene der Realität entspricht. Handelt es sich bei dem Sensor um eine Kamera kann nun das Objekt so gedreht werden, dass das gezeigte Bild dem der echten Kamera entspricht. Nun können die Positions- und Rotationsdaten direkt abgelesen werden.

Server

Die Netzwerkschnittstelle: Das Programm dient bei der Nutzung eines Netzwerkstreams als Klient und versucht sich beim Programmstart mit einem Server zu Verbinden. Die Verbundene IP kann in Unity im *ManagerObject* unter dem Skript *Manager* bei dem Feld Ip Adresse angepasst werden. Der Klient nutzt die Ports 13000 für die Entitäten und 13001 für die Ereignisse. Dies ist im Zweifelsfall anpassbar in der Klasse *CityModell.Assets.Scripts.RealObjectScripts.ManagerWithProzessor*.
Der Klient erwartet die gesendeten Daten in einem Format als Binärdaten in der jeweiligen Reihenfolge:

- Entitäten:
    - 32-Bit-Integer - ID - Eine eindeutige Identifikation der Entität
    - 64-Bit-Double - Timestamp - Der Zeitpunkt der Positionsaufnahme
    - 3x 32-Bit-Float - Position - Sensorrelative Position der Aufnahme - (x,y,z)-Koordinaten, x zeigt nach Rechts, y nach Oben und z nach Vorne
    - 3x 32-Bit-Float - Ausrichtung - Sensorrelative Punkt, zu dem die Entität ausgerichtet ist
    - 3x 32-Bit-Float - Blickpunkt - Sensorrelative Punkt, den die Entität anschaut - Im Fall von Fahrzeugen entspricht dies der Ausrichtung der Räder
    - 2x 32-Bit-UInt - Farben - Individuelle Farben der Entität auf höchster Detailstufe - Erster Wert bestimmt die Untere, der Zweite die obere Farbe bei

Menschen, andere Entitäten nutzen bisher nur den ersten Wert - Die Farben sind als RGBa abgespeichert
    - 32-Bit-Integer - Typ - Die Art der Entität - 0 erzeugt einen Menschen, 1 einen PKW, 2 einen Bus und 3 ein Fahrrad
    - 32-Bit-Integer - SensorID - Die ID des Sensors von dem die Aufnahme stand
- Ereignisse:
    - 8-Bit-Byte - lengthtype - Ereignisse können verschiedene Längen besitzen, je nachdem welche optionalen (*kursiven*) Daten sie beinhalten. Um die Länge des Binärstream-Datenpakets zu kennen muss daher die Art angegeben werden - 0: ohne optionale Daten, 1: mit Endzeit, 2: Mit idealer Betrachtungsposition, 3: mit beiden optionalen Parametern
    - 32-Bit-Integer - SensorID - Die ID des Sensors von dem die Aufnahme stand
    - 64-Bit-Double - Zeitpunkt - Der Zeitpunkt, an dem das Ereignis stattfindet
    - *64-Bit-Double - Endzeitpunkt - Der Endzeitpunkt einer Zeitspanne (Optional)*
    - 3x 32-Bit-Float - Position - Sensorrelative Position der Aufnahme
    - *3x 32-Bit-Float - IdealPos - Sensorrelative, ideale Position zum Betrachten des Ereignis (Optional)*
    - 32-Bit-Integer - Typ - Die Art des Ereignisses - 0 repräsentiert einen Autounfall, 1 eine fallende Person, 2 das Losgehen einer Konfettikanone und 3 eine Explosion

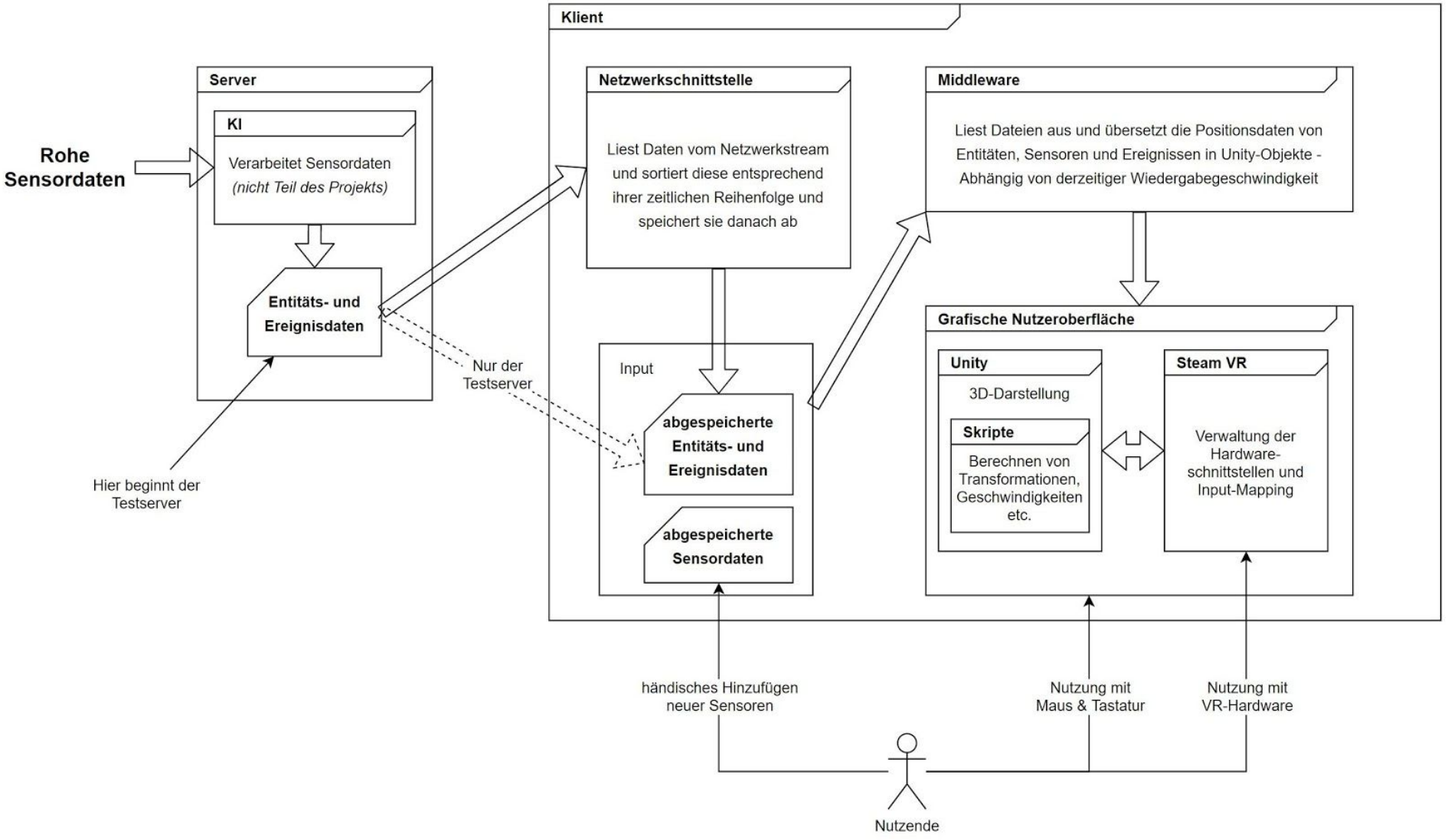
Der Testserver: Zum Testen der Visualisierung wurde ein Server erstellt, der auf unterschiedliche Weisen konfiguriert werden kann:

- Einfache Daten: Entitäten auf vordefinierten Positionsdaten bewegen sich in vordefinierter Geschwindigkeit in eine vordefinierte Richtung
- Stresstest: Erschafft eine hohe Zahl zufälliger Entitäten, die sich in zufälliger Geschwindigkeit in eine zufällige Richtung bewegen
- Daten aus Vorlage: Entitäten werden aus einer CSV-Datei ausgelesen und passend gesendet - Die Datei muss zeitlich geordnet, die Werte kommasepariert und Kommazahlen mit Punkten dargestellt werden, die Reihenfolge entspricht der der Versendung
- Server: Der Server kann die Daten entweder als Netzwerkstream zur Verfügung stellen und damit als Server agieren oder die Daten in Dateien schreiben, die direkt vom Klienten eingelesen werden können - Dadurch muss er die Daten nicht erst empfangen und kann ein wenig entlastet werden

Für Informationen zur korrekten Nutzung der Funktionalitäten kann der Server mit dem Parameter *--help* gestartet werden. Dadurch wird die korrekte Syntax zu Nutzung der jeweiligen Funktionen dargestellt.

Dataflow
Siehe Abbildung auf Seite 5.

Das Programm erhält seine Entitäts- und Ereignisdaten über einen Netzwerkstream (Die dafür relevanten Klassen finden sich in *Assets/Scripts/NetworkReader*). Dieser speichert die empfangenen Daten zeitlich sortiert in Dateien (BIN für Entitäten, CSV für Ereignisse, zu finden in *Assets/CSVInput*), welche zusammen mit der Sensorliste, (CSV, händisch zu pflegen) den für die korrekte Nutzung des Programms notwendigen Input darstellt. Verarbeitet wird der Input von dem Prozessor (zu finden in *Assets/Scripts/Prozessor*), der als Schnittstelle dient und die erstellten Objekte den Unity-Skripten (zu finden in *Assets/Scripts/ObjectScripts*) zur Verfügung stellt. Diese berechnen neue Positionen, Rotationen, Blickrichtungen, Animationen und Weiteres der Entitäten, stellen Ereignisse und Sensoren dar und beinhalten den Programmcode, der für die Zeitmanipulation verantwortlich ist. Das Ergebnis davon wird von Unity in 3D dargestellt und kann von Nutzenden beobachtet werden. Nutzende können dabei entweder VR-Hardware (dazugehörige Skripte befinden sich in *Assets/Scripts/Handhelds*) oder Maus und Tastatur (dazugehörige Skripte befinden sich in *Assets/Scripts/UIScripts*) benutzen.


Klient
Server
KI
Verarbeitet Sensordaten
(nicht Teil des Projekts)
Rohe Sensordaten
Entitäts- und Ereignisdaten
Hier beginnt der Testserver
Nur der Testserver
Netzwerkschnittstelle
Liest Daten vom Netzwerkstream und sortiert diese entsprechend ihrer zeitlichen Reihenfolge und speichert sie danach ab
Input
abgespeicherte Entitäts- und Ereignisdaten
abgespeicherte Sensordaten
Middleware
Liest Dateien aus und übersetzt die Positionsdaten von Entitäten, Sensoren und Ereignissen in Unity-Objekte - Abhängig von derzeitiger Wiedergabegeschwindigkeit
Grafische Nutzeroberfläche
Unity
3D-Darstellung
Skripte
Berechnen von Transformationen, Geschwindigkeiten etc.
Steam VR
Verwaltung der Hardware-schnittstellen und Input-Mapping
händisches Hinzufügen neuer Sensoren
Nutzung mit Maus & Tastatur
Nutzung mit VR-Hardware
Nutzende