# Einleitung:

In diesem Projekt habe ich ein vielseitiges Programm entwickelt, das sich mit Anpassungen auf eine Vielzahl von logistischen Problemen anwenden lässt. Das zentrale Konzept des Programms besteht darin, einen Score zu berechnen – eine numerische Bewertung, die beispielsweise die Übereinstimmung mit bestimmten Kriterien oder die Priorität eines Ziels repräsentiert. Z.b. in einem Aufzugssystem, steht der Score für die Priorität einer Etage. Dieser Wert wird anschließend genutzt, um die bestmögliche Entscheidung zur Zielerreichung zu treffen.

Für die Umsetzung habe ich die Lua-Programmiersprache verwendet, die sich besonders gut für flexible und effiziente Programmierung eignet. Zur Veranschaulichung und besseren Visualisierung der Funktionalität nutze ich eine modifizierte Version des beliebten Spiels Minecraft. Diese Modifikation erlaubt es, die Prinzipien und Abläufe meines Programms in einer interaktiven Umgebung darzustellen und zu testen.

Die Gewichtungen in der Berechnung des Scores können Manuell oder mit einer Funktion automatisch festlegen werden. Das heißt der Algorithmus passt sich an um das bestmögliche Ergebnis zu erhalten. Um dies zu optimieren wird in manchen Beispielen eine simple version von Maschinlerning angewendet.

# Beispiel 1: Aufzug

In diesem Beispiel steuert das Programm den Aufzug nicht direkte, stattdessen sendet es Befehle an einen Computer, der dann wiederum den Aufzug zur im Befehl spezifizierten Etage fährt. Man muss auseredem auch wissen in diesem Beispiel würde man die Etage bei dem Rufen des Aufzuges festlegen um ein effizientes Ergebnis zu erhalten.

Das Beispiel hat drei Etagen und eine Beliebige Anzahl an Leuten die auf den Etagen verteilt sind:

Über den Personen steht wohin sie möchten.

“[]” ist der lehre Aufzug.

| | | |
|---|---|---|
| **3** | Score: 2<br>1 2<br>  | [ ] |
| **2** | Score: 0<br>1<br>  | Distant zum Aufzug: 1<br>(-1 Score) |
| **1** | Score: 0<br>3 3<br>  | Distanz zum Aufzug: 2<br>(-2 Score) |

Im nächsten Schritt gehen die Leute in den Aufzug. Es ist wichtig das die Leute im Aufzug mehr priorisiert werden damit der Aufzug nicht zu voll wird, deshalb werden sie doppelt gezählt. Wo die Personen hin wollen, ist erst relevant wenn sie im Aufzug sind. Es werden außerdem nur gezählt wie viele Leute in den Etagen und in dem Aufzug sind.

| | | |
|---|---|---|
| **3** | Score: 0 | Im Aufzug:<br>1 2<br>  |
| **2** | Score: 2<br>1<br>  | Distanz zum Aufzug: 1<br>(-1 Score) |
| **1** | Score: 2<br>3 3<br>  | Distanz zum Aufzug: 2<br>(-2 Score) |

Als nächstes fährt der Aufzug zu Etage 1 da diese als erstes von dem Computer wahrgenommen* wird. Nun steigt eine Person aus und zwei ein:

| | | |
|---|---|---|
| **3** | Score: 2 | Distanz zum Aufzug: 2<br>(-2 Score) |
| **2** | Score: 2<br>1<br>  | Distanz zum Aufzug: 1<br>(-1 Score) |
| **1** | Score: 0 | Im Aufzug:<br>2 3 3<br>  |

Nun würde der Aufzug zu 2, dann 3 fahren. In dem Programm werden auch noch weitere Faktoren wie die Wartezeit genutzt, um das Ziel optimal zu erreichen.

In dem Beispiel in Minecraft nutze ich ComputerCraft um den Aufzug zu steuern und die Personen auf den Etagen darzustellen.

## Beispiel 2: Sotierer

Es lässt sich das selbe Prinzip auf ein Sortieralgorytmus anwenden. Dieser sortiert nicht nur die Gleichen sachen zusammen sondern nach ähnlichkeit:

| | |
|---|---|
| Fisch | Gefrierfach: |
| Eis | Fisch, Eis, Pizza |
| Apfel → | Obstkorb: |
| Pizza | Apfel, Banane |
| Banane | |

Im vergleich zum Aufzug Beispiel ersetzen die Gegenstände die Personen und das Gefrierfach/Obstkorb die Etagen.

Dies könnte z.B. auf Waren, Personen und Daten anwenden.

## Beispiel 3: Navigator

In diesem Beispiel bewegt  ein separaten Pfadfinealgorytmus um den besten weg zu einem der Angegebenen Zielpunkte. Mein Algorithmus nutzt das selbe Prinzip um die “Fahrzeuge” bewegen, er sendet den “Fahrzeugen” befehle die diese dann Ausführen.