Fachhochschul-Bachelorstudiengang
**MEDIZIN- UND BIOINFORMATIK**
A-4232 Hagenberg, Austria

# Aufzugsimulation zum Testen von Zeitplänen für Werbungen

Bachelorarbeit
Teil 2

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

**Daniel Wilfing**

Betreuer: Oliver Krauss MSc, FH OÖ Forschungs & Entwicklungs GmbH, Hagenberg
Begutachter: FH-Prof. DI Dr. Herwig Mayr

Hagenberg, Juli 2015

# Inhaltsverzeichnis

# Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Datum, Unterschrift

# Kurzfassung

Aufzüge werden mit einem neuen Notrufsystem ausgestattet, welches über einen Bildschirm verfügt. Außerhalb von Notfällen soll auf diesen Bildschirmen Werbung abgespielt werden. Dabei soll die Werbung so abgespielt werden, dass sie möglichst oft von der richtigen Zielgruppe gesehen wird. Doch um einen möglichst optimalen Zeitplan berechnen zu können, sind Informationen über Aufzugfahrten notwendig. Diese sollen mithilfe einer Aufzugsimulaiton erzeugt werden. Aufzugsimulationen ermöglichen es die Tätigkeiten von Aufzügen und Personen zu simulieren. Über sie können Aufzugsysteme schnell simuliert werden, um so rasch die benötigten Fahrten generieren zu können. Es existieren bereits einige Programme, die sich mit der Simulation von Aufzugsysteme beschäftigen. Jene bieten allerdings nur sehr beschränkte Möglichkeiten für die Einstellung an. Eine eigene Implementierung ist deswegen notwendig, um bestimmte Aspekte der Simulation einstellen zu können. In dieser Arbeit wird eine Möglichkeit vorgestellt, wie Personen und Aufzüge implementiert werden können, um eine flexible Aufzugsimulation zu entwickeln. Zusätzlich wird auch eine grafische Darstellung der Simulation implementiert. Danach werden verschiedene Szenarien erstellt, welches das Programm simulieren muss. Die verschiedenen Szenarien werden getestet und die Resultate diskutiert.

# Abstract

Elevators will be equipped with a new emergency system. The new system has a monitor, which can display advertisement. The adverts should be shown at such times, so that mainly people of their target audience see them. Information about elevator rides is necessary to calculate an optimized schedule for the advertisements. These elevator rides shall be simulated by an elevator simulation. Elevator simulations simulate the operations of persons and elevators. Through simulations, elevator systems can be simulated very quickly and will then generate the required elevator rides. There are already different programs available which simulate elevator systems. However, these implementations only offer restricted adjustments and are not flexible. To control every aspect of the elevator system, it was necessary to create a new implementation from scratch. One possibility how persons and elevators can be implemented to create a flexible elevator simulation is presented in this thesis. Additionaly, the program has been extended by a graphical visualisation of the simulation. Afterwards, different scenarios have been created. These scenarios have been tested and the results discussed.

# 1 Einleitung

## 1.1 Probleme bei der Simulation von Aufzugsystemen

Die Simulation einer möglichst realitätsnahen Aufzugsimulation scheint auf den ersten Blick eine leichte Aufgabe zu sein. Es müssen lediglich Personen von einem Stockwerk in ein anderes transportiert werden. Komplexeres Personenverhalten und die Implementierung eines Aufzugsystems, welches es ermöglicht, beliebig viele Aufzüge einzusetzen, ist keine leichte Aufgabe. Das Aufzugsystem muss die fahrenden Aufzüge so koordinieren, dass die wartenden Personen möglichst schnell zum gewünschten Stockwerk kommen. Dabei muss auch das Verhalten der einzelnen Personen richtig umgesetzt werden, damit sie mit den Aufzügen und anderen Objekten in der Simulation richtig interagieren.

Die Aufzugsimulation, die im Zuge dieser Arbeit entstandt, soll eingesetzt werden um Zeitpläne von Werbungen zu testen. Dafür werden Aufzugfahrten generiert, auf denen die Werbungen abgespielt werden. Es muss aufgezeichnet werden, welche Person welche Werbung gesehen hat. Aus den daraus entstehenden Fahrten kann dann ermittelt werden, wie erfolgreich ein bestimmter Zeitplan war.

## 1.2 Motivation für den Einsatz von Simulationen

Mithilfe einer Aufzugsimulation ist es leicht möglich, die verschiedenen Zeitpläne schnell zu testen. Das Verhalten der Personen und der Aufzüge wird hierbei simuliert und jede Fahrt aufgezeichnet. Dadurch kann leicht festgestellt werden, wie oft eine Werbung von der richtigen Zielgruppe gesehen worden ist. Weiters ermöglicht eine Simulation es, verschiedene Gebäude mit unterschieldichen Aufzugsanlagen zu simulieren. So können die Zeitpläne in verschiedenen Typen von Gebäuden getestet werden, wie Einkaufszentren oder Wohngebäuden.

Über eine Simulation ist es auch leicht möglich, die Pläne bei verschieden großen Gebäuden mit unterschiedlichen vielen Personen zu testen. Sie ermöglicht es, verschiedenste Szenarien in kurzer Zeit durchzuspielen, um so mögliche Verbesserungen für die Pläne zu finden.

## 1.3 Ziel dieser Arbeit

Ziel dieser Arbeit ist es, eine Aufzugsimulation zu entwickeln, die es ermöglicht, verschiedene Zeitpläne für Werbungen zu testen. Es soll möglich sein, Simulationen in verschiedenen Gebäuden mit unterschiedlichen Personen vorzunehmen. Dabei muss aufgenommen werden, wie oft eine Werbung eine Person ihrer Zielgruppe erreicht hat. Dadurch kann entschieden werden, wie gut ein bestimmter Plan ist. Die Aufzugsimulation soll dabei so implementiert werden, dass es möglich ist, einzelne Teile der Simulation auszutauschen, wie beispielsweise das Personen- oder Aufzugverhalten.

Die Simulation soll auch um Grafiken erweitert werden. Die einzelnen Stockwerke, Aufzüge und Personen sollen grafisch dargestellt werden. Durch diese bildhaften Elemente kann das Verhalten der Personen und Aufzüge während der Simulation leichter nachvollzogen werden. Bei der grafischen Darstellung muss es eine Möglichkeit geben zu erfahren, welche Werbungen in einem Aufzug abgespielt worden sind. Dabei sollte auch die Zielgruppe der Passagiere angezeigt werden.

Die Simulation soll anhand von verschiedenen Szenarien uberprüft werden. Dabei wird kontrolliert, ob die benötigten Aufzugfahrten generiert werden und die Ergebnisse in einem sinnvollen Rahmen sind.

# 2 Stand der Technik

Im folgenden Kapitel wird beschrieben, worum es sich bei Simulationen handelt. Es werden die verschiedenen Arten von Simulationen vorgestellt. Anschließend werden Aufzugsimulationen erläutert und verschiedene Programme vorgestellt, die diese Simulationen umgesetzt haben.

## 2.1 Simulationen

Bei Simulationen werden bestimmte Modelle nachgespielt, die möglicherweise zu komplex oder aufwändig wären, um sie in der Realität zu testen. Die dabei verwendeten Modelle sind vereinfachte Abbildungen eines realen Systems. Über die Simulation ist es dann möglich wiederholbare Experimente an dem Modell durchzuführen. Die Daten der Simulationen werden anschließend analysiert, um dabei zu neuen Erkenntnissen zu gelangen, die auch im realen System zutreffen. [Sokolowski u. Banks 2011]

Kern der Simulation ist das Modell, welches eine Abbildung eines Ereignisses oder bestimmten Systems ist. Je nach Anwendungsfall kann das Modell annähernd das gesamte System abbilden oder nur einen bestimmten Teil. Dem Modell können auch Eigenschaften hinzugefügt werden, die so nicht im abgebildeten System existieren. Jedes Modell dient einem bestimmten Zweck und es kann üblicherweise nicht für verschiedene Aufgaben eingesetzt werden [Stachowiak 1973]. Die Simulation definiert dann das Verhalten des Modells. Es versucht die benötigten Aufgaben und Operationen des realen Systems zu kopieren und wendet diese auf das Modell an. Dieses Verhalten kann parametrisiert und dadurch gesteuert werden, um bestimmte Szenarien mit dem Modell durchspielen zu können.

Simulationen werden benötigt, um mit Systemen experimentieren zu können, bei denen dies in der Realität nicht einfach möglich wäre oder zu gefährlich ist. Beispiele dafür sind Fahr- und Flugsimulatoren, bei denen gefährliche Situationen simuliert werden können, ohne dass Personen gefährdet werden. Bei der Simulation kann hier ein Computer die einzelnen Verhaltensweisen des Systems nachstellen. Computer sind für die Durchführung von Simulationen aber nicht zwingend erforderlich. So sind Crashtests mit Dummys ebenfalls Simulationen, da Verkehrsunfälle simuliert werden.

Neben der Simulation von teuren oder gefählrichen Modellen haben Simulationen noch weitere Vorteile. Dadurch, dass die meisten Simulationen in Computerprogrammen umgesetzt werden, ist es möglich, das Verhalten des Modells zu beschleunigen oder zu verlangsamen. Der Benutzer oder die Benutzerin kann deswegen bestimmte Teile des Modells einfacher analysieren. Weiters ermöglichen Simulationen es, Modelle mit unterschiedlichen Parametern zu testen, um so zu neuen Erkentnisse über die Zusammenhänge innerhalb des Systems zu kommen. Mithilfe von Simulationen kann das Verhalten von Modellen auch visualisiert werden. Dadurch werden manche Teile des Modells verständlicher und es wird leichter, die Ergebnisse der Simulation nachzuvollziehen.[Sokolowski u. Banks 2011]

Der Einsatz von Simulation bringt auch gewisse Nachteile mit sich. Das richtige Entwickeln eines Modells ist nicht leicht, wodurch es zu fehlerhaften Simulationen und Ergebnissen kommen kann. Als ebenfalls schwierig erweist sich auch das richtige Interpretieren der Daten, besonders wenn die gewählten Parameter zufällig zu einem merkwürdigen Ergebnis führen. Weiters sollten Simulationen nicht in jeder Situation eingesetzt werden, zum Beispiel wenn eine Analyse des Problems bereits gereicht hätte.

Laut [Ören 2005] lassen sich die Anwendungsgebiete von Simulationen in fünf Kategorien einteilen:

**1. Training:** Diese Simulationen ermöglichen es, Personen bestimmte Situationen nachspielen zu lassen, um sie besser auf die entsprechenden Situationen in der realen Welt vorzubereiten.

**2. Entscheidungsunterstützung:** Simulationen dieser Kategorie werden eingesetzt, um bestimmte Situationen besser einschätzen und bewerten zu können.

**3. Verständnis:** Bei diesen Simulationen werden Hypothesen, die sich mit komplexen Systemen beschäftigen, getestet.

**4. Ausbildung und Wissen:** Diese Simulationen werden im Bereich der Ausbildung eingesetzt und simulieren Systeme mit dynamischen Verhalten.

**5. Unterhaltung:** Bei diesen Simulationen werden Systeme mit dynamischen Verhalten möglichst realistisch dargestellt.

Die im Laufe dieser Arbeit entstehende Simulation gehört zu der Kategorie der entscheidungsuntertstützenden Simulationen. Es werden bei dieser Aufzugsimulation verschiedene Zeitpläne getestet, um so feststellen zu können, welche Pläne gut geeignet für die echten Aufzügen sind.

## 2.2 Aufzugsimulation

Bei Aufzugsimulationen wird das Verhalten eines klassischen Aufzugsystems nachgestellt. Das Modell umfasst ein Haus mit mehreren Stockwerken und mindestens einem Aufzug. Während der Simulation möchten verschiedene Personen von einem Stockwerk in ein anderes wechseln. Die Simulation muss die Aufzüge dann so zwischen den Stockwerken bewegen, dass die Personen möglichst schnell in das gewünschte Stockwerk transportiert werden. Diese Aufgabe wirkt mit einem kleinem Modell einfach, doch bei mehreren Aufzügen und vielen Personen wird die Berechnung bereits komplexer.

Aufzugsimulationen werden eingesetzt, um verschiedene Verhaltensweisen mit den Aufzüge testen zu können. Diese Verhaltensweisen können auch im echten System getestet werden, allerdings wäre dies zu zeit- und kostenaufwändig. Durch Simulationen dagegen kann ein Aufzugsystem mit Personen schnell getestet werden. Simulationen erlauben, es verschiedenste Verhaltensweisen von Fahrstühlen mit einer unterschiedlichen Anzahl von Personen, Stockwerken und Aufzügen zeitsparend zu testen.

Es existieren bereits mehrere Programme, die sich mit der Simulation von Aufzügen beschäftigen. In den folgenden Abschnitten werden drei dieser Simulationsprogramme vorgestellt.

### 2.2.1 Elevator Simulator

Dieses Simulationsprogramm [Kimmey 2008] ermöglicht es, ein Gebäude mit maximal sechzig Stockwerken und mit bis zu zwölf Aufzügen zu simulieren. Der Benutzer oder die Benutzerin kann die Aufzüge über verschiedene Befehle steuern. Jeder der Aufzüge kann einzeln angesteuert werden, um sie unabhängig voneinander in den verschiedenen Stockwerken bewegen zu können. Dabei kann der Anwender oder die Anwenderin auch kontrollieren, wann die Türen geöffnet und geschlossen werden. Weiters ist es während laufender Simulation möglich, die Anzahl der Stockwerke und Aufzüge zu verändern. In Abbildung 1 ist erkennbar, dass die Position der Aufzüge innerhalb des Gebäudes leicht festzustellen ist.

Dieses Programm verfügt über keine Personen, wodurch die Befehle für das Wechseln der Stockwerke vom Benutzer oder der Benutzerin kommen müssen. Es ist zwar möglich, zufällige Befehle abzusetzen, allerdings wird die Simulation damit auch nur über kurze Zeit angetrieben. Der Anwender oder die Anwenderin muss die Rolle der Personen im Gebäude übernehmen. Diese Simulation beschäftigt sich dadurch hauptsächlich mit der korrekten Steuerung und Bewegung der Aufzüge zwischen den einzelnen Stockwerken.

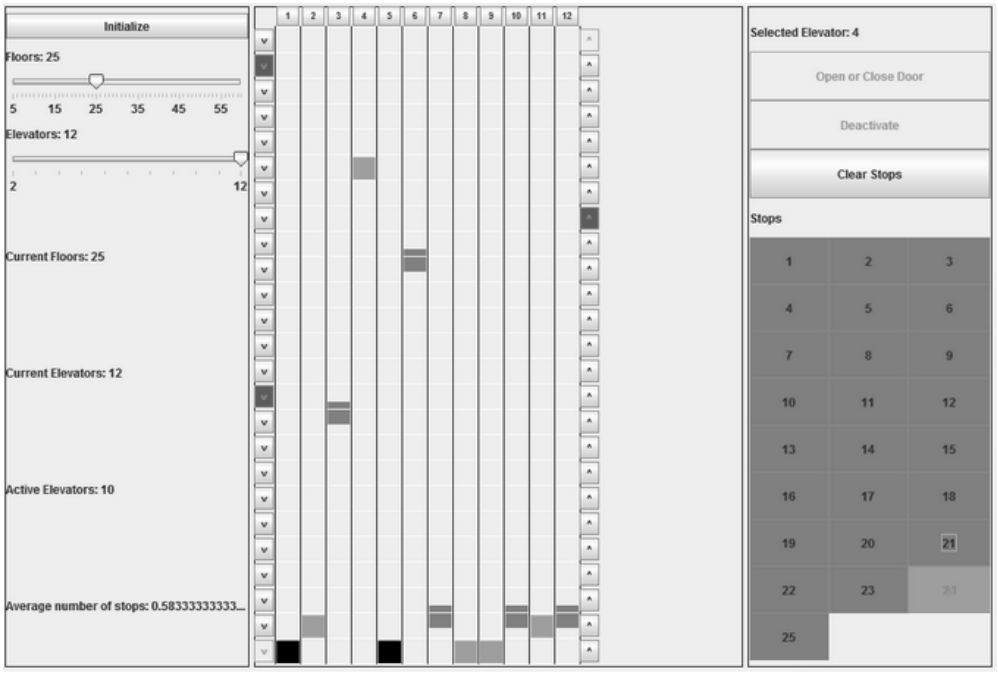


Abbildung 1: Die Aufzüge bewegen sich zu den Stockwerken, die links oder rechts hervorgehoben sind. Schwarze Aufzüge sind inaktiv und dunkelgraue Aufzüge bewegen sich zu einem anderen Stockwerk. (nach [Kimmey 2008])

### 2.2.2 Elevator Sim

Elevator Sim wurde von [Dailey 2014] entwickelt und ist eine flexible Aufzugsimulation. Dieses Programm erlaubt es, beliebig viele Stockwerke mit beliebig vielen Aufzügen und Personen zu simulieren. Jeder Person kann ein bestimmtes Verhalten zugewiesen werden, welches definiert, wann sie ein Stockwerk wechseln möchte. Die Aufzüge transportieren die einzelnen Personen dann automatisch zum richtigen Stock. Dieses Programm hat zwei verschiedene Verhaltensweisen für die Aufzüge implementiert, die bestimmen, in welcher Reihenfolge die einzelnen Stockwerke besucht werden. Weiters ermöglicht dieses Programm, dass bestimmte Teile der Simulation zufällig erzeugt werden, was besonders beim Verhalten der Personen hilfreich ist. Während der Simulation können die Daten zu den verschiedenen Aufzügen ausgelesen werden, wie in Abbildung 2 erkennbar ist. Dabei ist es zusätzlich möglich, die Simulation zu beschleunigen oder zu verlangsamen.

Dieses Programm verfügt zwar über eine Vielzahl von verschiedenen Szenarien, doch es ist nicht möglich, selbst ausgewählte Parameter zu speichern. Besonders wegen der Möglichkeit, bestimmte Parameter zufällig zu setzen, wäre es interessant, eine selbst parametrisierte Simulation mehrmals zu testen. Das Verhalten der Personen ist ebenfalls simpel umgesetzt und es ist nicht möglich, eine Person mehrmals das Stockwerk wechseln zu lassen.

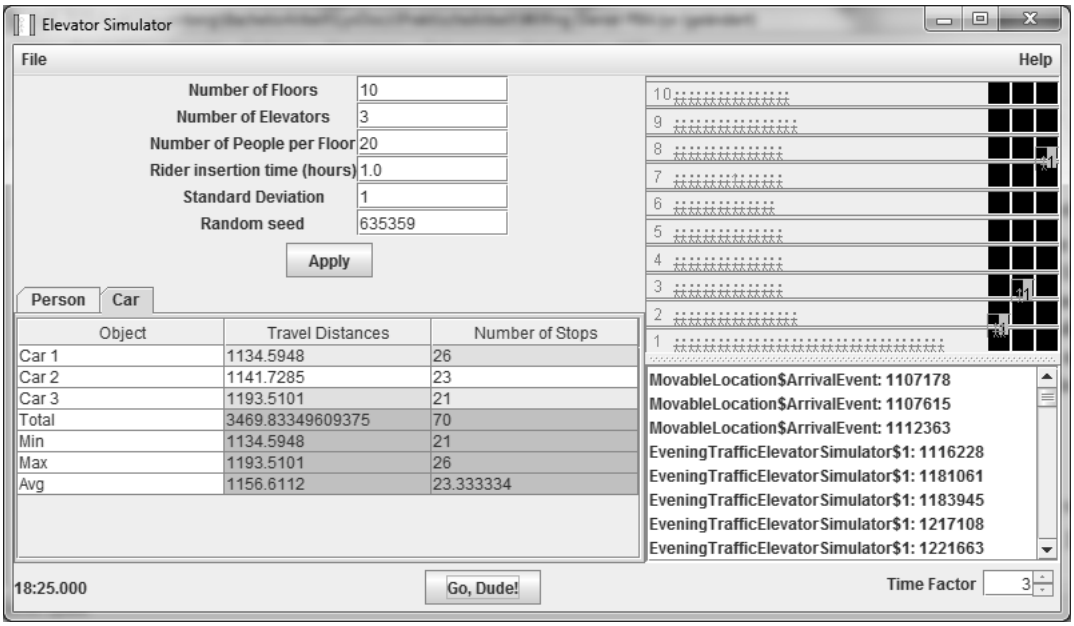


Abbildung 2: Während der Simulation werden Daten bezüglich der Fahrten der einzelnen Aufzüge angezeigt. Rechts oben sind die Aufzüge und Personen in jedem Stockwerk erkennbar. (nach [Dailey 2014])

### 2.2.3 ElViSS

Hierbei handelt es sich um eine komplexe Aufzugsimulation, entwickelt von [Berndt u. a. 2004]. Die Abkürzung ElViSS steht für „**El**evatorsimulation: **Vi**sualisierung, **S**imulation und **S**teuerung eines Fahrstuhlsystems“. Bei diesem Programm ist es möglich, Simulationen mit einer beliebigen Anzahl von Stockwerken, Aufzügen und Personen durchzuführen. Personen haben hauptsächlich ein zufälliges Verhalten, allerdings gibt es Zeiten (zum Beispiel Mittagspause um 12:00), zu denen jede Person ein bestimmtes Stockwerk besuchen möchte. Damit können Stoßzeiten des realen Systems simuliert werden. ElViSS verfügt über zwei verschiedene Verhaltensweisen für die Aufzüge. Bei dem ersten Verhalten bewegen sich die Aufzüge zufällig zwischen den Stockwerken. Das zweite Verhalten bildet das klassische Aufzugverhalten ab, bei dem sich der Aufzug je nach benötigtem Stockwerk bewegt. Es kann auch die Zeit eingegeben werden, die für die verschiedenen Aktionen des Aufzugs benötigt werden, wie zum Beispiel „Tür öffnen“ oder „Wartezeit pro Stockwerk“. Wie die anderen vorgestellten Programme verfügt ElViSS ebenfalls über eine grafische Darstellung des Aufzugsystems. Dabei kann der Benutzer oder die Benutzerin aber zwischen zwei verschiedenen Visualisierungen auswählen. In der simplen Darstellung werden sämtliche Objekte des Systems als geometrische Formen angezeigt. Bei der erweiterten Darstellung werden die Personen als Strichfiguren angezeigt und die einzelnen Abläufe des Aufzugsystems sind animiert. Diese Form der Darstellung ist in Abbildung 3 zu sehen.

Von den in diesem Kapitel vorgestellten Programmen bietet ElViSS die meiste Flexibilität bei der Parametrisierung des Modells. Neu erstellte Simulationen können auch problemlos abgespeichert werden, um eine erneute Durchführung der Simulation zu ermöglichen. Im Gegensatz zu Elevator Sim aus Abschnitt 2.2.2 ist es allerdings nicht möglich, das Verhalten einzelner Personen einzustellen.

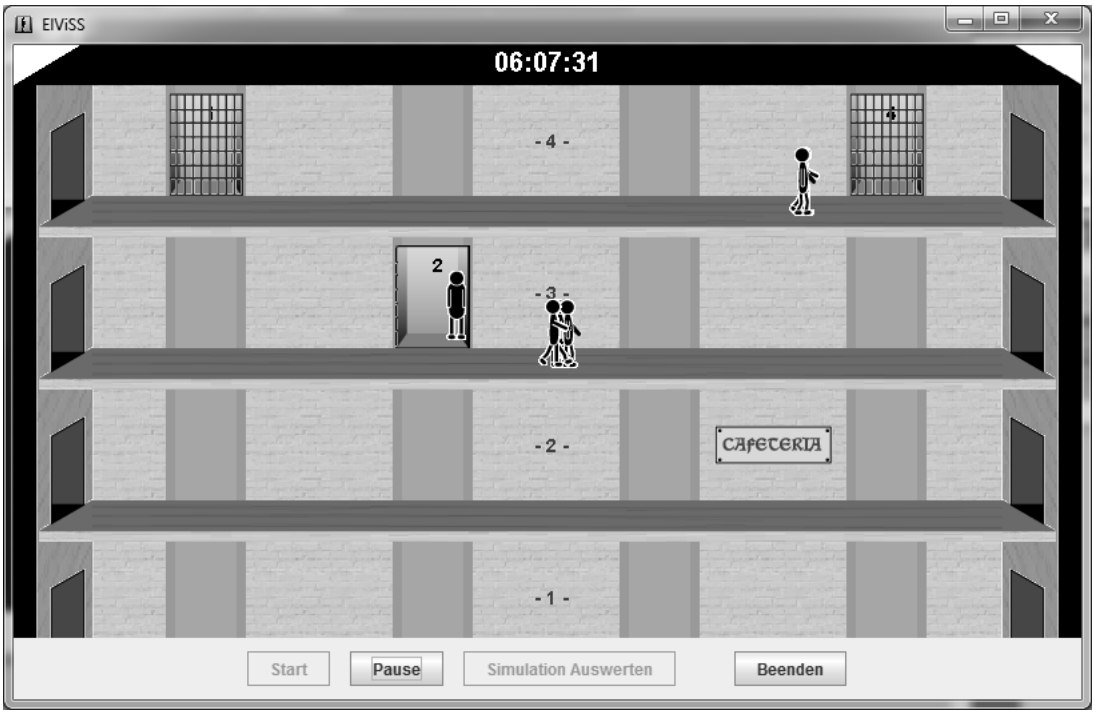


Abbildung 3: Bei der erweiterten Darstellung sind Aufzüge, Personen und die Stockwerke klar erkennbar. (nach [Berndt u. a. 2004])

# 3 Design

In diesem Kapitel werden die verschiedenen Verhalten näher beschrieben, die Personen und Aufzüge haben können. Es wird erläutert, wie die Verhalten umgesetzt werden müssen, um ein möglichst realitätsnahes Verhalten abbilden zu können. Anschließend wird die Struktur beschrieben, die angibt, wie die Personen- und Aufzugverhalten in der Simulation implementiert worden sind. Es werden die einzelnen Klassen der Verhaltensweisen definiert sowie deren Zusammenhänge und Klassenstruktur angegeben.

## 3.1 Personenverhalten

Die Personen treiben die Simulation voran, da sie sich mithilfe der Aufzüge zwischen den Stockwerken bewegen [Tuomas u. a. 2004]. Dafür müssen die Personen allerdings ein bestimmtes Verhalten haben, da sie von allein nicht anfangen werden, ihr Stockwerk zu wechseln. Dabei wäre es gut, wenn das Verhalten möglichst genau einstellbar wäre, damit bestimmte Situationen für die Simulation einstellbar sind. Damit könnten bestimmte realitätsnahe Verhaltensweisen, wie In-Rush und Out-Rush, leicht nachgestellt werden [Newell 1998]. Bei dem *In-Rush* handelt es sich um das Verhalten, dass viele Person vom Erdgeschoß in höhere Stockwerke fahren möchten. Diese Situation tritt häufig am Beginn des Tages auf, wenn viele Leute ihren Arbeitstag beginnen und zu ihrem Büro fahren möchten. Der *Out-Rush* dagegen tritt häufig zur Mittagszeit auf, wenn viele Personen von den höheren Stockwerken Richtung Erdgeschoß möchten, um sich in der Pause etwas zu Essen zu holen.

### 3.1.1 Möglichkeiten für die Umsetzung von Personenverhalten

Die einfachste Möglichkeit, das Verhalten der Personen umzusetzen, wäre es, sie einfach zu bestimmten Zeitpunkten zufällig zu einem anderen Stockwerk zu schicken. Die Simulation würde dadurch bereits funktionieren und die Aufzüge würden von den Personen verwendet werden. Damit könnten die oben genannten Verhalten allerdings nicht nachgestellt werden.

Wird den Leuten in der Simulation ein fixes Verhalten zugewiesen, dann ist es möglich, die einzelnen Personen zu steuern und entsprechend komplexe Situationen zu erstellen. Beispielsweise könnte so ein Verhaltensmuster sein, dass die Person zwischen 11:00 und 13:00 eine Cafeteria aufsuchen soll. Dieses Verhalten kann dann mehreren Leuten in der Simulation zugewiesen werden, wodurch es zum Beispiel möglich ist, den Out-Rush zu simulieren [Newell 1998]. Indem jede Person eine Anzahl verschiedener Verhaltensmuster erhält, kann so der Tagesablauf jeder Person definiert werden.

### 3.1.2 Implementieres Personenverhalten

Für die Implementierung der Aufzugsimulation wurden die fix definierbare Verhaltensmuster umgesetzt. Jede Person verfügt über verschiedene Verhalten, die sich bei unterschiedlichen Zeiten aktivieren. Wird ein Verhalten aktiviert, dann möchte die Person ein bestimmten Stockwerk oder ein Stockwerk mit einem bestimmten Typ (zum Beispiel Cafeteria oder Lobby) besuchen. Falls die Person ein Stockwerk mit einem bestimmten Typ besuchen möchte, dann kann noch entschieden werden, ob sie das näheste Stockwerk des entsprechenden Typs besucht, oder ein zufälliges. Auf den ersten Blick scheint es zwar keinen Sinn zu machen, dass die Person nicht zum nähesten relevanten Stockwerk fährt, allerdings gibt es Situationen, wo es Sinn macht. Solch eine Situation wäre ein Einkaufszentrum, wo viele Etagen mit Geschäften ausgestattet sind. Ohne den zufälligen Aspekt würden alle Personen, die ein Geschäft besuchen wollen, nur einen Stock nach oben oder unten fahren, statt sich im Einkaufszentrum zu verteilen.

Die genaue Implementierung der Klassen, die sich mit den Verhalten der Personen beschäftigen, wird im Abschnitt 4.2.2 näher beschrieben. Falls eine Person ihr Stockwerk wechseln möchte, dann sollte dies in der Simulation möglichst realitätsnah nachgespielt werden. In der Implementierung drückt die Person einen Aufzugknopf, je nachdem ob sie nach oben oder unten will. Dann wartet die Person solange, bis ein Aufzug in ihrem Stockwerk stehen bleibt, der genug Platz hat für eine weitere Person. Danach steigt sie ein, drückt den entsprechenden Knopf für die richtige Etage und wartet solange, bis der Aufzug im richtigen Stockwerk angelangt ist. Die Person steigt dann wieder aus und wartet solange, bis das nächste Verhalten gestartet wird.

In der Implementierung wird dieser Vorgang über Zustände beschrieben. Die Zustände dafür sind „idle“, „wait for cabin“ und „wait in cabin“. Bei dem Zustand „idle“ möchte die Person auf ihrem Stockwerk bleiben. Bei „wait for cabin“ wartet sie auf eine Aufzugkabine und bei „wait in cabin“ wartet die Person innerhalb der Kabine darauf, dass der Aufzug das richtige Stockwerk erreicht. Falls der Aufzug während der Fahrt ausfällt, dann steigt die Person aus der Kabine aus und wartet auf einen anderen Aufzug. In dem Zustandsdiagramm in Abbildung 4 sind die einzelnen Schritte grafisch dargestellt, die nötig sind, damit eine Person ein Stockwerk wechselt.

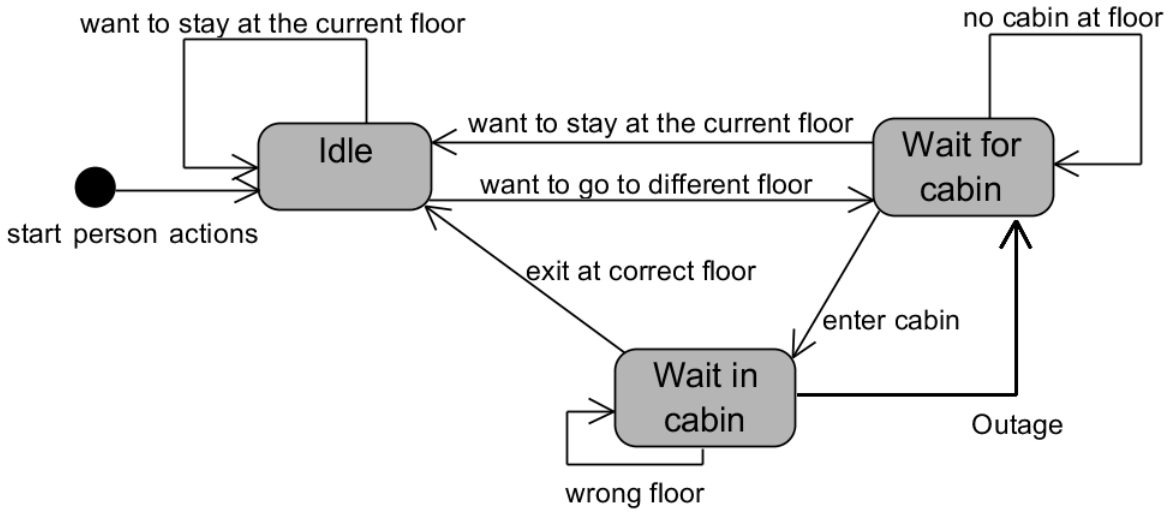


Abbildung 4: Eine Person hat immer einen von drei möglichen Zuständen, der beschreibt, ob sie auf ihrem Stockwerk verweilen möchte oder nicht.

## 3.2 Aufzugverhalten

Die Aufzüge transportieren die verschiedenen Personen von Stockwerk zu Stockwerk. Dafür benötigen sie ebenfalls ein Verhalten, welches definiert, wann sie zu welchem Stockwerk fahren und welche Tätigkeiten sie dabei durchführen. Aufzüge handeln nicht von allein, sondern werden durch die einzelnen Etagen- und Aufzugknöpfe gesteuert, die von den Personen betätigt werden. Der Aufzug muss dann lediglich entscheiden, in welcher Reihenfolge er die notwendigen Stockwerke besucht.

### 3.2.1 Möglichkeiten für die Umsetzung von Aufzugsverhalten

Die einfachste Umsetzungsmöglichkeit wäre hier, die Aufzüge zufällig durch das Gebäude fahren zu lassen, bis die Personen im richtigen Stockwerk angelangt sind. Diese Implementierung ist gut geeignet, um den groben Ablauf der Aufzugsimulation zu testen. Solch ein Verhalten spiegelt allerdings nicht das Verhalten von Aufzügen in der Realität wider. Echte Aufzüge fahren solange in eine Richtung und bleiben bei relevaten Stockwerken stehen, bis es in die entsprechene Richtung keine Aufträge mehr von Stockwerken gibt. Danach drehen sie um und bewegen sich in die andere Richtung [Strakosch u. Caporale 2010].

Dieses Verhalten muss jeder Aufzug besitzen. In einem Aufzgsystem koordinieren sich die einzelnen Kabinen allerdings auch, damit nicht sämtliche Aufzüge immer zum selben Stockwerk fahren. Es muss eine Kontrolleinheit geben, welche die Stockwerke zuweist, auf denen Personen einsteigen wollen [Khiang u. a. 2010]. Diese Kontrolleinheit sollte die Stockwerke den Aufzügen so zuweisen, dass die Stockwerke möglichst schnell erreicht werden. Eine Möglichkeit wäre es, den Aufzug zu dem Stockwerk hinzuschicken, der dem benötigten Stockwerk gerade am nähesten ist. Falls der näheste Aufzug allerdings gerade in die andere Richtung fährt, müsste er umdrehen, was viel Zeit kostet. Die Bedingung muss daher besser spezifiziert werden. Es sollte der Aufzug ausgewählt werden, der am nähesten ist und in die richtige Richtung fährt. Dadurch werden Stockwerke, auf denen Personen warten, schnell von einem Aufzug besucht.

### 3.2.2 Implementiertes Aufzugverhalten

Für die Implementierung in der Aufzugsimulation wurde das Verhalten implementiert, bei dem die Aufzüge solange in eine Richtung fahren, bis keine Stockwerke mehr angefahren werden müssen. Dabei wurden diese Verhalten so generisch entwickelt, dass es möglich ist sie, durch ein anderes Verhalten auszutauschen. Es wurde ein Verhalten entwickelt, welches die Aufzüge zufällig auswählt und durch das Gebäude schickt. Ein anderes Verhalten dagegen simuliert die realistische Vorgehensweise von Aufzügen. Sie fahren solange in eine Richtung und bleiben stehen, falls eine Person ein- oder aussteigen möchte. Muss zu keine Stockwerke in Fahrtrichtung mehr gefahren werden, dann drehen sie um. Dadurch, dass die Verhalten leicht ausgetauscht werden, ist es sehr leicht möglich, die unterschiedlichen Verhalten zu testen.

Die genaue Implementierung und die Klassen für diese Verhalten werden in Abschnitt 4.3.2 näher beschrieben. Falls ein Aufzug nicht benötigt wird, wartet er eine kurze Zeit lang auf Passagiere. Anschließend überpüft der Aufzug erneut, ob er zu einem anderen Stockwerk fahren muss. Neben dem Bewegen zwischen Stockwerken müssen die Aufzüge noch weitere Aktionen durchführen können, um sie möglichst realitätsnah umzusetzen. So müssen sie auch die Türen öffnen und schließen können, sowie unterschiedlich lange auf einem Stockwerk warten, je nachdem wie viele Personen ein und aussteigen. Auch das Absenden von Notrufen muss möglich sein, und es soll möglich sein, dass der Aufzug für längere Zeit außer Betrieb geht. Nachdem ein Notruf beendet ist, wird das normale Verhalten des Aufzugs fortgesetzt. Wird ein ausgefallener Aufzug wieder in Betrieb genommen, dann fährt er zuerst zurück in das Erdgeschoß und setzt dann erst sein übliches Verhalten fort. Sämtliche Zustände der Aufzüge sind im Diagramm in Abbildung 5 einsehbar.

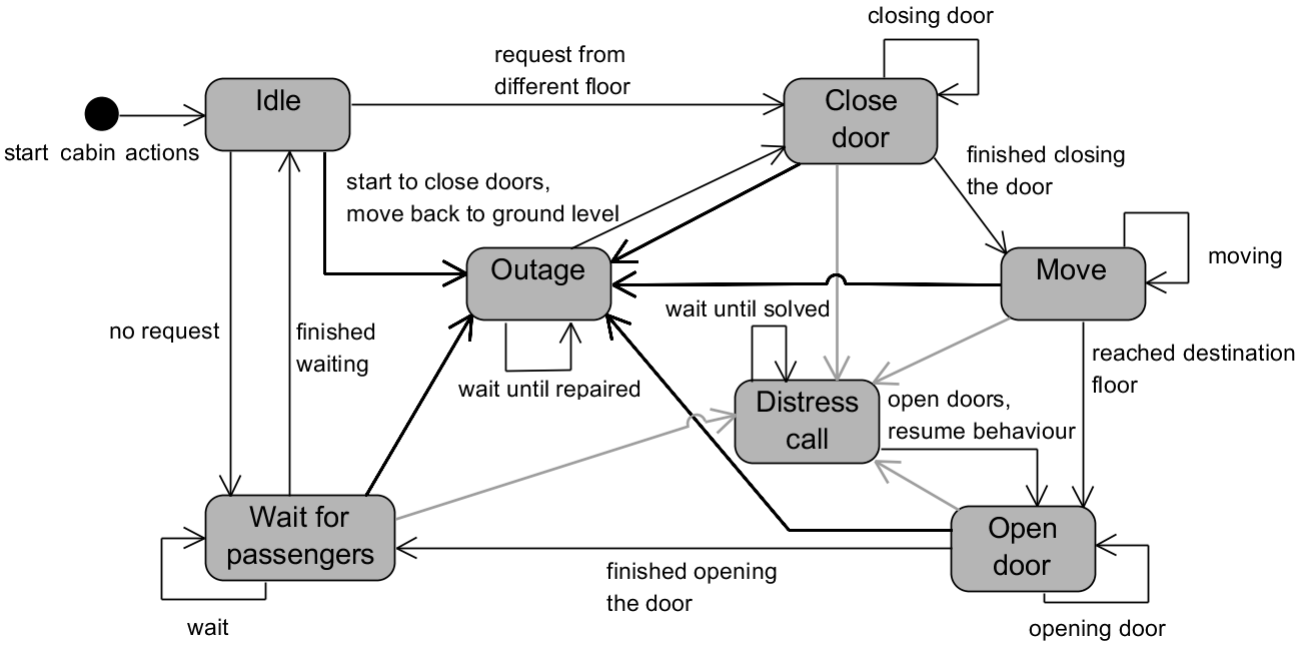


Abbildung 5: Aufzüge können zu fast jedem Zeitpunkt einen Notruf haben oder außer Betrieb genommen werden.

## 3.3 Klassenstruktur der Simulation

Um die Aufzugsimulation zu implementieren, wurde eine Vielzahl von unterschiedlichen Klassen programmiert. In diesem Abschnitt werden die wichtigsten Klassen und ihre gemeinsame Struktur beschrieben. Es wird erläutert, welche Designentscheidungen getroffen worden sind.

**ElevatorSimulationController**

Über diese Klasse ist es möglich, die Aufzugsimulationen einzurichten und zu starten. Sie verfügt über verschiedene Funktionen, um laufende Simulationen zu beeinflussen, wie beispielsweise Funktionen um die Simulationen zu pausieren und zu stoppen. Diese Klasse ist als Singleton umgesetzt, da sie in der Lage ist, beliebig viele Aufzugsimulation zu kontrollieren und zu simulieren. Mehrere Instanzen dieser Klasse sind dadurch nicht notwendig.

**ElevatorSimulation**

Diese Klasse stellt eine Aufzugsimulation dar und enthält die notwendigen Objekte, um eine Simulation durchführen zu können. Jeder *ElevatorSimulation* sind ein Gebäude zugewiesen sowie viele verschiedene Personen. Zusätzlich sind jedem Objekt dieser Klasse auch die Manager-Klassen zugewiesen (siehe *SimulationManager* dieses Abschnitts), welche die Simulation mit den gegebenen Gebäuden und Personen durchspielen. Um mehrere Objekte der *ElevatorSimulation* gleichzeitig ausführen zu können, implementiert diese Klasse das Interface *Runnable* [Krüger u. Hansen 2011].

**TimeManager**

Der *TimeManager* ist das Herzstück der Simulation und sendet in gewissen Abständen Nachrichten an die anderen Manager. Dies geschieht durch das Entwurfsmuster Observer, wobei der *TimeManager* von der Klasse *Observable* ableitet [Ullenboom 2014]. Dadurch beginnen die Personen in die Aufzüge einzusteigen und die Aufzüge fangen damit an, die Personen zu den gewünschten Stockwerken zu befördern.

**SimulationManager**

Dabei handelt es sich um eine abstrakte Klasse, die eine Schnittstelle zu der *ElevatorSimulation* darstellt. Jede andere Manager-Klasse leitet von dieser Klasse ab, um so Zugriff auf das Gebäude und die Personen in der Aufzugsimulation zu bekommen. Die einzige Ausnahme ist der Zeitmanager (*TimeManager*), da diese Klasse von *Observable* ableitet.

**SimulationObserver**

Diese abstrakte Klasse leitet von *SimulationManager* ab und implementiert zusätzlich das Interface *Observer* [Ullenboom 2014]. Diese Klasse wird für die Manager benötigt, die in regelmäßigen Abständen Nachrichten vom *TimeManager* empfangen, um dann verschiedene Aktionen durchzuführen.

**IFloorManager**

Dieses Interface definiert die Methoden, welche benötigt werden, um die Fahrstuhl-Ruftasten zu simulieren. In der Simulation werden diese Methoden von den einzelnen Personen aufgerufen. Klassen, die dieses Interface implementieren, müssen auch vom *SimulationManager* ableiten.

**ICabinManager**

Die Methoden dieses Interfaces beschreiben das Aufzugsystem der Simulation. Dieses System gibt vor, welcher Aufzug zu welchem Stockwerk fahren soll. Erneut müssen implementierende Klassen vom *SimulationManager* ableiten, um so zu den notwendigen Informationen bezüglich Aufzüge, und Stockwerken zu gelangen.

**IPersonManager**

Dieses Interface definiert Methoden für die verschiedenen Personen, die sich in der Simulation durch das Gebäude bewegen. Implementierende Klassen müssen auch von *SimulationObserver* ableiten, da die Personen von sich aus agieren müssen. Durch die ständigen Nachrichten des *TimeManager* kann dann das Verhalten der Personen nachgestellt werden.

### ICabinControl

Das letzte Interface für die Manager-Klassen definiert die notwendigen Methoden für eine einzelne Aufzugskabine. In dieser Klasse werden die Funktionen beschrieben, die benötigt werden, um die Zustände eines Aufzugs (siehe Abbildung 5) in der Simulation abzubilden. Diese Manager-Klasse kommt nicht nur einmal pro Simulation vor. Ihre Anzahl ist definiert durch die Anzahl der Kabinen, die sich in dem Gebäude befinden. Klassen, die dieses Interface implementieren, müssen auch vom *Simulation-Observer* ableiten. Dadurch erhalten sie die Nachrichten vom *TimeManager* und beginnen dadurch ,die notwendigen Aufgaben eines Aufzugs simulieren.

In der Implementierung dieser Arbeit ist es notwendig, dass bestimmte Management-Klassen miteinander kommunizieren. Die *SimulationObserver* für die Personen und Aufzugskabinen werden dem *TimeManager* übergeben, wodurch sie Nachrichten erhalten. Der *FloorManager* wird angesprochen vom *PersonManager* und *CabinManager*, da diese Zugriff auf die Funktionen für die Bedienung der Aufzugruftasten benötigen. Der *CabinManager* und *PersonenManager* können auch auf bestimmte Teile der *CabinControl* zugreifen, um so die Interaktionen zwischen den Personen und den Aufzügen simulieren zu können. Bei dem Klassendiagramm in Abbildung 6 sind die Beziehungen zwischen den einzelnen Klassen grafisch dargestellt.

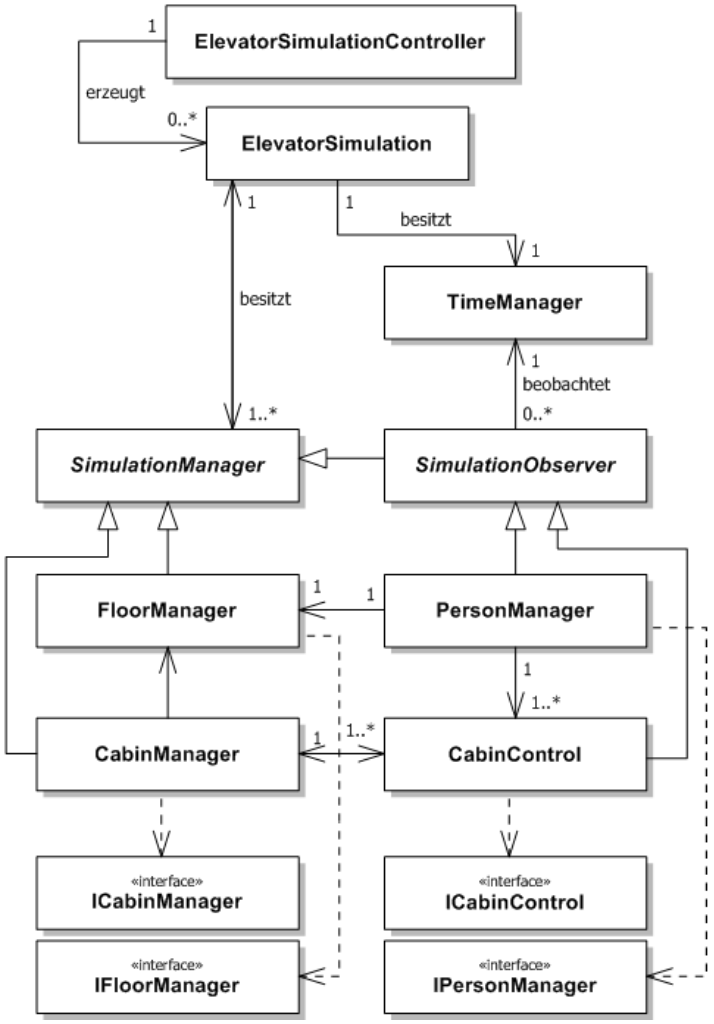


Abbildung 6: Die Manager-Klassen haben Referenzen zueinander, um den Austausch von Informationen für die Simulation zu ermöglichen.

## 3.4 Klassenstruktur der grafischen Darstellung

Der Benutzer oder die Benutzerin soll die Möglichkeit haben, die Aufzugsimulation mitzuverfolgen. Dafür ist es notwendig Klassen zu implementieren, welche bestimmte Vorgänge der Simulation in einem Fenster darstellen. Dieser Abschnitt beschreibt die einzelnen Klassen und Interfaces, die notwendig sind um eine grafische Darstellung für die Simulation umzusetzen.

#### IView

Klassen, die von diesem Interface ableiten, bilden den Kern der grafischen Darstellung. Sie implementieren sämtliche Objekte (beispielsweise Textfelder oder Labels), mit denen die anderen Klassen der Visualisierung interagieren. Dadurch geben sie vor, wie die Darstellung der Aufzugsimulation genau aussehen wird.

#### ViewUtil <IView>

Diese abstrakte Klasse vereinfacht die Benutzung der von *IView* vorgegebenen grafischen Objekte. Sie definiert Methoden, die es den anderen Klassen ermöglicht, die einzelnen Objekte der Darstellung zu manipulieren. Über *ViewUtil* kann kontrolliert werden, dass keine ungültigen Änderungen bei den Objekten von *IView* vorgenommen werden.

#### ViewHandle <ViewUtil>

Sämtliche Klassen, die Änderugen an der grafischen Darstellung vornehmen wollen, müssen von diesem Interface ableiten. Dieses Interface ermöglicht die Benutzung von *ViewUtil*.

#### ICabinElevatorViewHandle <ViewUtil>

In diesem Interface sind Methoden vorgegeben, welche relevant für die Aufzugkabinen in der Simulation sind. Solche Funktionen sind beispielsweise *closeDoors*, *openDoors* und *moveCabin*.

#### IFloorElevatorViewHandle <ViewUtil>

Ableitende Klassen müssen die Funktionen implementieren, die für die einzelnen Aufzugruftasten auf den einzelnen Etagen benötigt werden. Dadurch kann dargestellt werden, welche Tasten auf welchem Stockwerk betätigt worden sind.

#### IInitialisationElevatorViewHandle <ViewUtil>

Dieses Interface wird für die Initialisierung der grafischen Darstellung benötigt. Falls die Visualisierung der Aufugsimulation angezeigt werden soll wird die Methode *initialiseView* aufgerufen. Diese Funktion lädt den derzeitigen Stand der Aufzugsimulation und ändert die einzelnen grafischen Objekte dementsprechend.

#### IPersonElevatorViewHandle <ViewUtil>

Klassen dieses Interfaces definieren die Methoden, die für die Bewegungen der Personen zwischen den Stockwerken notwendig sind. Solche Funktionen sind beispielsweise *addPersonToFloor* und *addWaitingPerson*.

#### ITimeElevatorViewHandle <ViewUtil>

Dieses Interface leitet neben *ViewHandle* auch von dem Standard-Interface *Observer* ab. Dadurch kann es die Nachrichten des *TimeManagers* (siehe Abschnitt 3.3) empfangen. *ITimeElevatorViewHandle* wird benötigt, um die aktuelle Zeit in der Simulation anzeigen zu können.

**IUserElevatorViewHandle <ViewUtil>**

Ableitende Klassen dieses Interfaces implementieren sämtliche Methoden, die benötigt werden damit der Benutzer oder die Benutzerin mit der grafischen Darstellung interagieren können. Dadurch können bestimmte Teile der Visualisierung und Simulation manipuliert werden, wie zum Beispiel die Geschwindigkeit, in der die Simulation abgespielt wird.

Damit die ableitenden Interfaces von *ViewHandle* aufgerufen werden, war es notwendig, die Methoden in den verschiedenen Manager-Klassen (siehe Abschnitt 3.3) aufzurufen. Beispielsweise muss die ableitende Klasse von *IPersonManager* die Methode *addWaitingPerson* von *IPersonElevatorViewHandle* aufrufen, wenn eine Person beginnt, auf einen Aufzug zu warten.

Sämtliche Interfaces der grafischen Darstellung sind im Klassendiagramm in Abbildung 7 einsehbar.


Abbildung 7: Für jede relevante Tätigkeit, die grafisch dargestellt werden soll, steht ein Interface zur Verfügung.

# 4 Implementierung

Im Kapitel 3 wurden die wichtigsten Klassen der Aufzugsimulation vorgestellt. Es wurde die Klassenstruktur erläutert und mit welchen Designentscheidungen die verschiedenen Klassen implementiert worden sind. In diesem Kapitel werden die wichtigsten Methoden und Funktionsweisen der verschiedenen Klassen näher beschrieben. Es werden die verschiedenen Manager-Klassen der Simulation erläutert sowie die Klassen beschrieben, die für eine grafische Darstellung der Simulation benötigt werden.

## 4.1 Gesamtsteuerung

Dabei handelt es sich um die Klassen, mit denen eine Aufzugsimulation eingerichtet und gestartet werden kann. Die betreffenden Klassen sind hierbei der *ElevatorSimulationController* und die *ElevatorSimulation*.

Wie in Abschnitt 3.3 erwähnt, wurde die Klasse *ElevatorSimulationController* als Singleton umgesetzt. Mit *getInstance* können die anderen Methoden dieser Klasse ausgeführt werden. Mit der Funktion *setupSimulation* kann eine Aufzugsimulation erstellt werden. Diese Methode verfügt über unterschiedliche Schnittstellen und je nach eingegebenen Parametern wird ein neues Objekt der Klasse *ElevatorSimulation* erstellt. Dieses Objekt wird anschließend in einen Thread gepackt und in eine Liste gespeichert, um weitere Zugriffe darauf zu ermöglichen. Nach der Erstellung kann die Simulation über die Funktion *startSimulation* gestartet werden. Bei dieser Methode wird der Thread mit der Simulation gestartet. Durch den Einsatz verschiedener Threads ist es möglich, dass mehrere Simulationen gleichzeitig laufen können. Die Simulation kann auch über die Funktion *startSimulationAndVisualisation* gestartet werden, wobei hier zusätzlich eine grafische Darstellung der Simulation angezeigt wird. In Abschnitt 4.4 wird die grafische Darstellung der Simulationen näher beschrieben.

Neben dem Erzeugen und Starten von Simulationen ist es über den *ElevatorSimulationController* auch möglich, die laufenden Simulationen zu manipulieren. So ermöglichen es die Funktionen *pauseSimulation* und *stopSimulation,* eine laufende Aufgzugsimulation zu pausieren oder abzubrechen. Dies geschieht, indem der Zeitmanager (siehe Abschnitt 4.2.1) der entsprechenden Simulation angesprochen und eine boolsche Variable verändert wird.

Während der Simulation werden sämtliche Fahrten der Aufzüge gespeichert. Diese können mit den Funktionen *facilityRunsOfSimulation* und *facilityRunsOfAllSimulations* ausgelesen werden. Die einzelnen Fahrten können ausgewertet werden, um dadurch zu erfahren, wie effektiv der Zeitplan für die Werbungen war.

Da jede Simulation in einem eigenen Thread läuft, muss über andere Funktionen kontrolliert werden, ob die Simulation noch läuft oder fertig ist. Die Methode *allSimulationsFinished* überprüft den Zeitmanager der einzelnen Simulationen und bestimmt dadurch, ob die Simulation noch läuft. Dementsprechen wird dann ein boolscher Wert zurückgegeben. Falls das Programm selbst auf alle Simulationen warten soll, dann muss die Funktion *waitEverySimulation* verwendet werden. Diese knüpft die Threads der einzelnen Simulationen an den Haupt-Thread des Programms. Dadurch wird das Programm erst fortgesetzt, wenn sämtliche Simulationen beendet worden sind.

Die Klasse *ElevatorSimulation* enthält Objekte sämtlicher Manager-Klassen, die benötigt werden, um eine Aufzugsimulation durchführen zu können. Über die Funktion *initialise* werden die einzelnen Manager initialisiert und die notwendigen Variablen gesetzt. Zusätzich werden auch die Referenzen der Manager untereinander gesetzt und es werden zufällige Notrufe und Ausfälle erzeugt, die dann in der Simulation auftreten. Gestartet wird die Simulation über die abgeleitete Methode *run*. Der Zeitmanager (siehe Abschnitt 4.2.1) beginnt Nachrichten zu versenden, bis die benötigte Zeit in der Simulation vergangen ist. Die anderen Manager beginnen verschiedene Methoden zu durchzuführen, wodurch die Simulation startet. Die verschiedenen Methoden der Manager-Klassen werden in Abschnitt 4.2 erläutert.

Als Beispiel werden im folgenden Listing 1 zwei verschiedene Simulationen gestartet. Der Funktion *setupSimulation* werden hierbei zwei verschiedene Ojekte der Klasse *ElevatorSimulationObject* übergeben. Diese Klasse enthält alle notwendigen Daten, um eine Simulation zu starten und implementiert das Interface *Serializable* [Ullenboom 2014]. Dadurch können Objekte dieser Klasse leicht abgespeichert werden.

Listing 1: Die Aufzugsimulationen müssen dem *ElevatorSimulationController* zuerst zugewiesen werden, um gestartet werden zu können.

```
1 //zwei verschiedene ElevatorSimulationObject werden übergeben
2 ElevatorSimulationController.getInstance().setupSimulation(Simulation1);
3 ElevatorSimulationController.getInstance().setupSimulation(Simulation2);
4
5 //alle Simulationen werden gestartet
6 ElevatorSimulationController.getInstance().startSimulations();
7
8 //Programm soll erst fortfahren, wenn sämtliche Simulationen fertig sind
9 ElevatorSimulationController.getInstance().waitEverySimulation();
```

## 4.2 Steuereinheiten der Simulation

Die Simulation der Aufzuganlagen und Personen werden von verschiedenen Klassen übernommen, den so genannten Managern. Jeder dieser Manager stellt Funktionen für einen bestimmten Teil der Simulation zur Verfügung. Solche Teile umfassen beispielsweise die Personen oder die einzelnen Aufzüge. Während der Simulation kommunizieren die Manager untereinander und rufen verschiedene Funktionen auf. Dadurch wird das Aufzugsystem des entsprechenden Gebäudes simuliert. In den folgenden Abschnitten werden die einzelnen Manager und deren wichtigsten Funktionen vorgestellt.

### 4.2.1 Zeitmanager

Dieser Manager (*TimeManager* in der Implementierung) treibt die Simulation voran, indem er in regelmäßigen Abständen Nachrichten an den Personenmanager (siehe Abschnitt 4.2.2) und den Kabinenmanager (siehe Abschnitt 4.2.4) sendet. Der Zeitmanager berechnet zu Beginn zwei verschiedene Daten, die angeben in welchen Zeitraum die Simulation durchgeführt wird. Danach wird in regelmäßigen Abständen ein bestimmter Wert (beispielsweise eine Sekunde) beim ersten Datum hinzugefügt. Die Simulation wird beendet, wenn das erste Datum nach dem zweiten Datum ist.

Mithilfe der abgeleiteten Funktion *notifiyObservers* werden dann die Manager informiert, dass wieder Zeit vergangen ist. Der Zeitmanager kann so eingestellt werden, dass die Simulation in Echtzeit erfolgt. Die Variable *timeLapse* definiert, wie lange der Zeitmanager nach dem Senden jedesmal warten muss. Je höher *timeLapse*, desto schneller wird die Simulation. Ist dieser Wert auf „1“ eingestellt, dann erfolgt die Simulation in Echtzeit. Standardmäßig ist keine Wartezeit definiert, wodurch die Simulation so schnell berechnet wird wie möglich. Für die grafische Darstellung und für realitätsnahe Simulationen ist eine Wartezeit zwischen jedem *notifyObserver* nützlich.

Der Zeitmanager kann mithilfe der boolschen Variable *pause* angehalten werden. Ist diese Variable auf „true“ gesetzt, dann wartet der Zeitmanager in einer Schleife solange, bis *pause* wieder auf *false* gesetzt wird oder die Simulation abgebrochen wird. Im Sequenzdiagramm in Abbildung 8 sind die einzelnen Schritte des Zeitmanagers eingezeichnet.

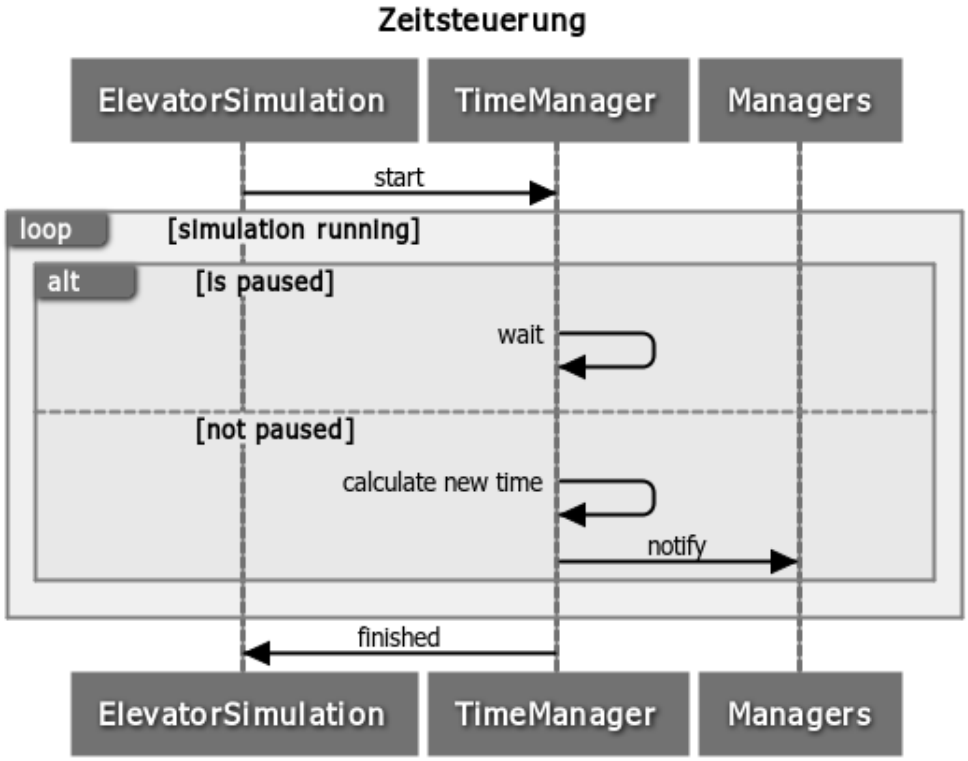


Abbildung 8: In regelmäßigen Abständen werden die anderen Manager benachrichtigt. Ist die notwendige Zeit abgelaufen, dann wird die *ElevatorSimulation* darüber informiert.

### 4.2.2 Personenmanager

Der Personenmanager kümmert sich um das richtige Verhalten jeder einzelnen Person innerhalb der Simulation. Klassen, welche diesen Manager in der Simulation umsetzen sollen, müssen das Interface *IPersonManager* implementieren. In einer Liste sind sämtliche Personen eingetragen sowie auf welchem Stockwerk sie sich befinden. Wenn dieser Manager eine Nachricht des Zeitmanagers (siehe Abschnitt 4.2.1) erhält, dann wird das Verhalten jeder Person überprüft. Falls eine Person das Stockwerk wechseln möchte, dann simuliert die Personensteuerng das Verhalten der Personen. Beispielsweise ruft sie zu Beginn eine Funktion des Etagenmanagers (siehe Abschnitt 4.2.5) auf, um so das Drücken eines Etagenknopfes zu simulieren.

Bevor der Personenmanager verwendet werden kann, muss das Verhalten sämtlicher Personen mittels der Funktion *initialise* sortiert werden. Diese Methode sortiert die einzelnen Verhalten jeder Person nach ihrer Zeit. Verhalten, die früher passieren, erscheinen dadurch zuerst in der Liste. Neben dem Sortieren werden auch noch die Personen auf den verschiedenen Stockwerken verteilt.

Der Personenmanager implementiert das Interface *Observer*, wodurch sie die Funktion *update* erhält. Diese Funktion wird in regelmäßigen Abständen von dem Zeitmanager (siehe Abschnitt 4.2.1) aufgerufen. Bei jedem Aufruf wird überprüft, ob eine Person ihr derzeitiges Stockwerk wechseln möchte oder ob eine Person gerne eine andere Tätigkeit unternehmen möchte. Die verschiedenen Tätigkeiten und Abläufe sind in Abbildung 4 einsehbar. Um die verschiedenen Tätigkeiten in der Simulation umzusetzen, wurden verschiedene Funktionen implementiert. Die Methode *checkPersonWantsFloor* wird aufgerufen, wenn die Person ihr Stockwerk wechseln möchte. Bei dieser Methode wird zuerst kontrolliert, ob der entsprechnde Etagenknopf bereits gedrückt ist. Wenn nicht, dann führt der Personenmanager die Funktion für das Drücken eines Knopfes aus. Im weiteren Lauf der Simulation wird dadurch das Aufzugssteuerungssystem (siehe Abschnitt 4.2.3) informiert, welches einen Aufzug zum entsprechenden Stockwerk schickt. Die Funktionen *checkPersonWaitingOutsideCabin* und *checkPersonWaitingInsideCabin* implementieren die Tätigkeiten der Personen, die gerade in einen Aufzug einsteigen oder innerhalb der Kabine auf das richtige Stockwerk warten.

Wenn eine Person ihr gewünschstes Stockwerk erreicht hat, dann wird das durchgeführte Verhalten nach hinten in der Liste verschoben. Wenn jedes Verhalten einmal durchgeführt worden ist, dann werden die Verhalten der Person für den restlichen Tag nicht mehr überprüft. Dies geschieht über eine Liste von boolschen Werten, die angeben, welche Personen für den jetzigen Tag noch aktiv sind und welche nicht. Beginnt ein neuer Tag, dann werden sämtliche Werte in dieser Liste auf „true“ gesetzt, wodurch wieder jede Person überprüft wird.

In dem Sequenzdiagramm in Abbildung 9 werden die Abläufe des Personenmanagers grafisch veranschaulicht.

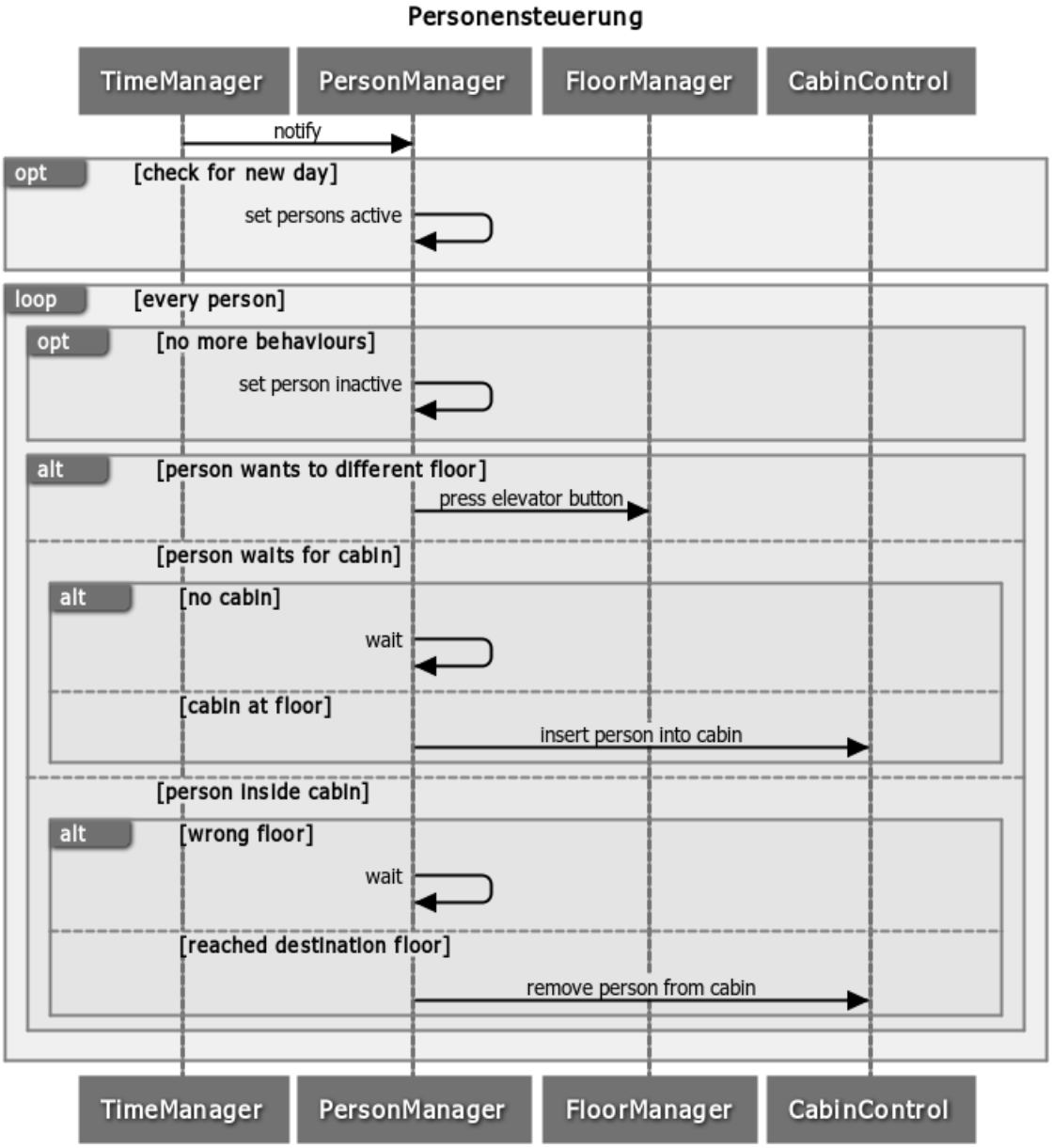


Abbildung 9: Nach jeder Nachricht des Zeitmanagers werden alle aktiven Personen überprüft. Falls eine Person aus einer Aufzugkabine ein- oder aussteigt, dann wird der entsprechende Kabinenmanager darüber informiert.

### 4.2.3 Aufzugsteuerungssystem

Diese Manager-Klasse kümmert sich um die Verteilung der Aufzüge zwischen den einzelnen Stockwerken. Implementierende Klassen müssen das Interface *ICabinManager* verwenden. Dieser Manager

entscheidet anhand einer Strategie, welcher Aufzug zu welchem Stockwerk fahren soll. Die Strategie ist dabei austauschbar, wodurch es möglich ist, verschiedene Verhalten bei der Auswahl von Aufzügen zu simulieren. Besagte Strategien werden im Abschnitt 4.3.2 näher beschrieben.

Das Aufzugsteuerungssystem verfügt über verschieden viele Queues, je nachdem wieviele Aufzüge im Gebäude vorhanden sind. Jeder Queue ist einer bestimmten Kabine zugewiesen. Über diese Queues kann das Steuerungssystem entscheiden, welche Stockwerke von welchem Aufzug besucht werden sollen. Aufzüge können mithilfe der Funktion *currentFloorQueueForCabin* abfragen, ob ein Stockwerk angefahren werden soll. Diese Methode liefert die Queue mit den benötigten Stockwerken für den jeweiligen Aufzug. Vor der Retournierung der Queues wird zuerst die Funktion *updateFloorQueues* ausgeführt. In dieser Funktion kontrolliert das Aufzugsteuerungssystem zuerst den Etagenmanager (siehe Abschnitt 4.2.5), und erhält von dieser sämtliche Stockwerke, auf denen Personen auf einen Aufzug warten. Anschließend wird anhand einer Strategie entschieden, wie die neuen Stockwerke auf die Aufzüge verteilt werden. Danach kann *currentFloorQueueForCabin* die aktualisierte Queue mit den Stockwerken zurückgeben. Der Ablauf dieser Methode wird in Abbilung 10 dargestellt.

Die Funktionen *deleteUpFromFloorQueue* und *deleteDownFromFloorQueue* entfernen ein Stockwerk aus der Queue eines Aufzugs. Diese Methoden werden von den verschiedenen Kabinenmanagern (siehe Abschnitt 4.2.4) aufgerufen. Neben dem Entfernen aus der Queue benachrichtigen sie auch den Etagenmanager darüber, dass ein Aufzug ein bestimmtes Stockwerk erreicht hat. Der Etagenmanager muss darüber informiert werden, damit die Aufzugruftasten des entsprechenden Stocks wieder inaktiv geschalten werden.

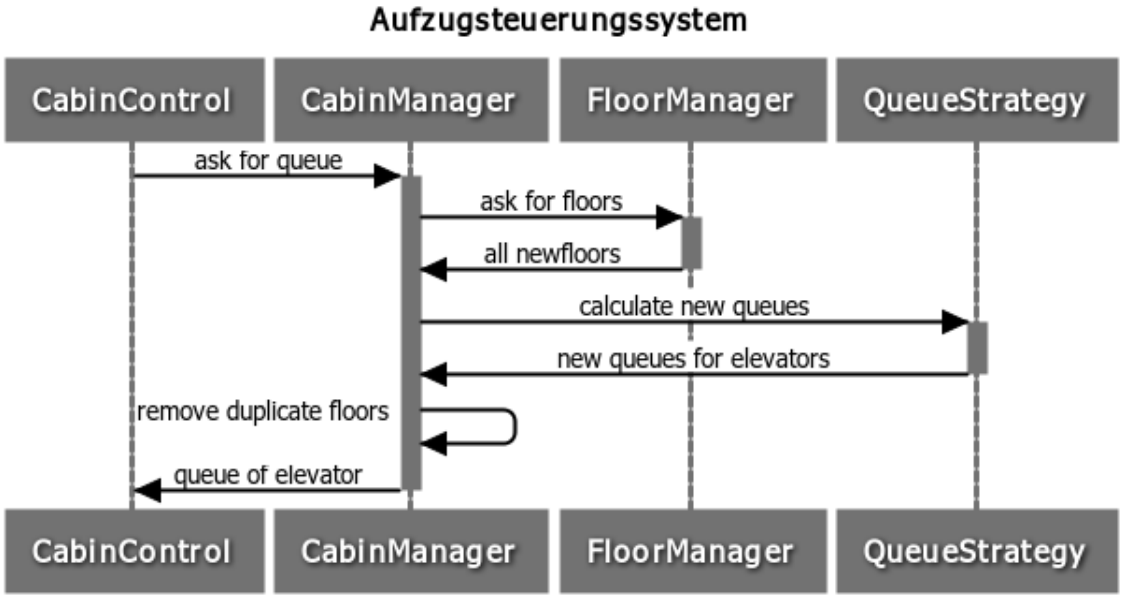


Abbildung 10: Die Queues eines Aufzugs werden erst neu berechnet, wenn der entsprechende Aufzug danach fragt.

### 4.2.4 Kabinenmanager

Der Kabinenmanager implementiert sämtliche Methoden, die benötigt werden um das Verhalten eines einzelnen Aufzugs zu simulieren. Klassen, welche diese Rolle der Simulation übernehmen sollen, müssen das Interface *ICabinControl* verwenden. Er simuliert die verschiedenen Tätigkeiten, die benötigt werden um einen Aufzug realitätsgetreu simulieren zu können. Besagte Tätigkeiten wurden bereits in Abschnitt 3.2.2 definiert. Der Kabinenmanager ist die einzige Manager-Klasse, die in einer Simulation mehrmals vorkommen kann. Die Anzahl von benötigten Objekten dieser Klasse ist gegeben durch die Anzahl der Aufzüge in der Simulation. Sämtliche Objekte dieser Klasse erhalten Nachrichten von dem Zeitmanager (siehe Abschnitt 4.2.1). Bei Erhalt einer Nachricht führen die einzelnen Kabinen ihre Tätigkeiten aus. Dabei wird in einer Variable gespeichert, wie lange eine bestimmte Tätigkit noch andauert.

Diese Klasse setzt das Interface Observer ein, wodurch es notwendig ist, die Methode *update* zu implementieren. Diese Funktion ist das Herzstück des Kabinenmanagers, da hier entschieden wird, welche Tätigkeit der Aufzug als nächstes ausführt. Hat der Aufzug beispielsweise gerade nichts zu tun, dann wird die Funktion *actionIdle* ausgeführt. Bei dieser Funktion berechnet der Aufzug das nächste Stockwerk, zu dem er sich bewegen muss. Dies geschiet mithilfe von Aufzugstrategien (siehe Abschnitt 4.3.1), welche entscheiden, in welcher Reihenfolge die notwendigen Stockwerke besucht werden. Falls kein anderes Stockwerk besucht werden muss, wartet der Aufzug eine bestimmte Zeit lang auf seinem Stockwerk und ruft dann erneut *actionIdle* auf. Die Methoden *actionDoorOpens* und *actionDoorCloses* simulieren die Zeit, die der Aufzug benötigt, um die Türen zu öffen und zu schließen. *actionMoveUpOrDown* bewegt den Aufzug nach oben oder nach unten, wobei die benötigte Zeit von den zurückgelegten Stockwerken abhängt. Wenn die Kabine auf einem neuen Stockwerk angelangt ist, wird die Funktion *actionWaitPassengers* ausgeführt. Bei dieser Methode wartet die Kabine darauf, dass Personen die Aufzugkabine betreten und verlassen. Jeder ein- und aussteigende Passagier fügt dabei ein paar Sekunden bei der Wartezeit hinzu, um so die zusätzliche Zeit zu simulieren, die ein- und aussteigende Personen benötigen.

Da während den simulierten Fahrten Werbungen abgespielt werden müssen, verfügt der Kabinenmanager auch dafür über Funktionen. Wenn der Aufzug beginnt sich zu bewegen, dann wird die Methode *facilitySetNewProject* aufgerufen. Diese Funktion erzeugt eine neue Werbeeinschaltung und speichert die Zeit ab, wann die Werbung begonnen hat. Wenn die Werbung fertig abgespielt werden konnte, wird die Methode *facilitySetEndTime* aufgerufen. *facilitySetEndTime* speichert den Zeitpunkt ab, zu dem die Werbung beendet worden ist. Falls der Aufzug noch fährt, wird die nächste Werbung über *facilitySetNewProject* abgespielt. Wenn die Aufzugkabine das richtige Stockwerk erreicht und zum Stehen kommt, wird die letzte Werbung fertig abgespielt und danach keine Werbeschaltung mehr durchgeführt. Werbungen werden daher nur abgespielt, wenn sich der Aufzug zwischen den einzelnen Stockwerken bewegt.

Zur grafischen Veranschaulichung ist ein Sequenzdiagramm der oben erwähnten Abläufe in Abbildung 11 dargestellt.

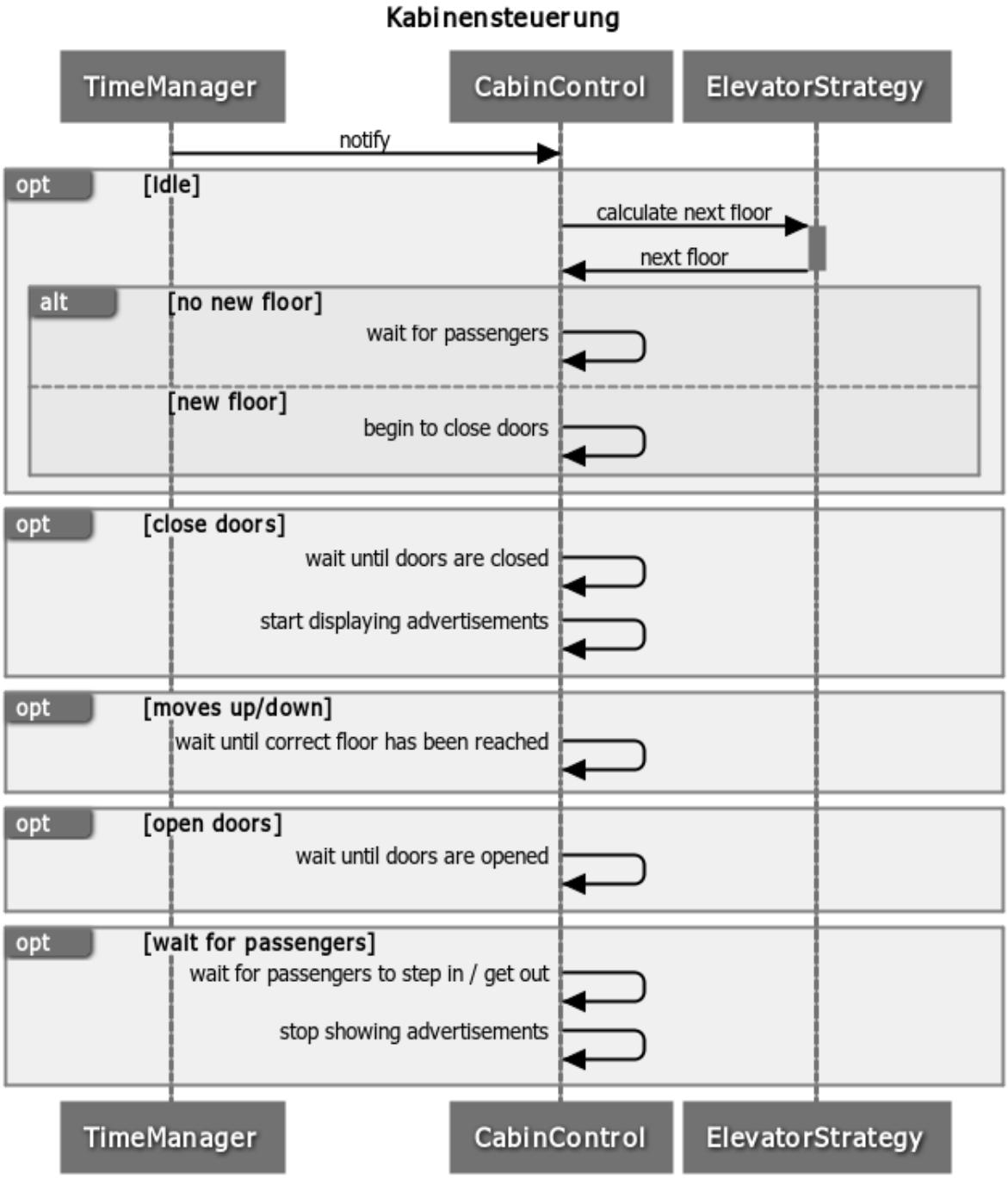


Abbildung 11: Jede Kabine verfügt über dieselben Tätigkeiten, die in einer bestimmten Reihenfolge durchgeführt werden. Dadurch kann die Simulation der Aufzüge realitätsnah erfolgen.

### 4.2.5 Etagenmanager

Diese Manager-Klasse wird benötigt, um die Aufzugruftasten auf jedem Stockwerk zu simulieren. Implementierende Klassen müssen das Interface *IFloorManager* benutzen. Dieser Manager spielt eine wichtige Rolle für den Personenmanager (siehe Abschnitt 4.2.2), da die einzelnen Personen über diese Klasse die Aufzüge zu sich holen können. Der Etagenmanager stellt Funktionen zur Verfügung, die es ermöglichen, das Drücken von Aufzugruftasten zu simulieren. Weiters speichert diese Klasse ab, welche

Tasten auf welchem Stockwerk gedrückt sind und gibt diese Informationen an den Kabinenmanager (siehe Abschnitt 4.2.4) weiter.

Mit den Funktionen *isUpButtonPressed* und *isDownButtonPressed* können die einzelnen Personen erfahren, ob die entsprechende Aufzugrufaste auf ihrem Stockwerk bereits gedrückt ist. Um die Tasten zu drücken, müssen die Methoden *pressUpButton* und *pressDownButton* verwendet werden. Der Etagenmanager speichert dann ab, welche Taste auf welchem Stockwerk gedrückt worden ist. Der Ablauf dieser Methoden ist in Abbildung 12 dargestellt. *setUpButtonInactive* und *setDownButtonInactive* setzen die entsprechenden Tasten dagegen wieder inaktiv. Diese Methoden werden von den Kabinenmanagern verwendet, um die Tasten wieder inaktiv zu schalten, wenn sie ein Stockwerk erreicht haben.

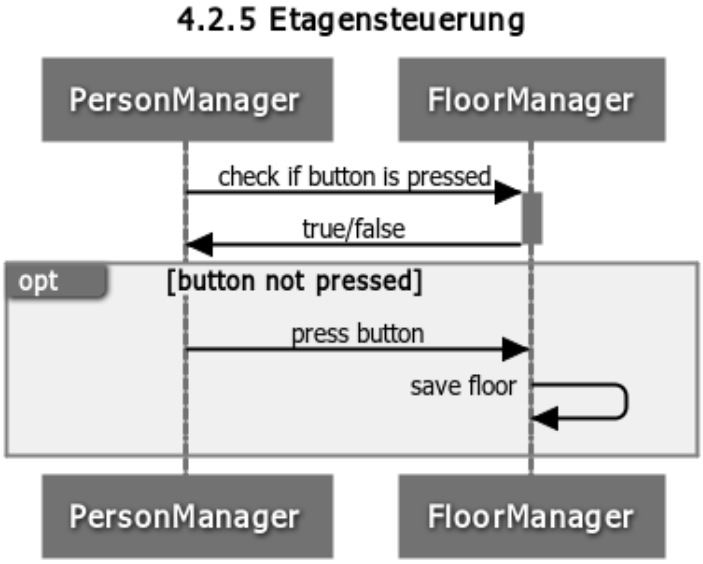


Abbildung 12: Personen kontrollieren zuerst, ob ihre gewünschte Aufzugruftaste schon gedrückt worden ist. Wenn nicht, dann betätigen sie die Taste.

## 4.3 Strategien

Aufzugsysteme können verschiedene Strategien verwenden, mit denen sie entscheiden, zu welchem Stockwerk sie als nächstes fahren. Um unterschiedliche Strategien in der Simulation zu ermöglichen, wurden die Interfaces *IElevatorStrategy* und *IQueueStrategy* entwickelt. Über diese Interfaces können neue Strategien hinzugefügt werden, die dann in der Simulation verwendet werden können.

Klassen, die *IElevatorStrategy* implementieren, definieren die Strategien, mit denen jeder einzelne Aufzug für sich entscheidet, zu welchem Stockwerk er sich als nächstes bewegt. Sie beeinflussen nicht die anderen Aufzüge. Dadurch ist es möglich, jedem Aufzug in der Simulation eine andere Strategie zuzuweisen.

Die *IQueueStrategy* kommt in jeder Simulation nur einmal vor. Sie wird vom Aufzugssteuerungssystem (siehe Abschnitt 4.2.3) verwendet und über diese Strategien wird entschieden, welcher Aufzug zu welchem Stockwerk fahren soll. Das entsprechende Stockwerk wird dafür in den Queue des Aufzugs geschrieben. Der Aufzug entscheidet dann über die *IElevatorStrategy*, wann genau er zu dem verlangten Stockwerk fährt. Beispielsweise ermöglichten *IQueueStrategy* es, dass Aufzüge, die gerade untätig sind oder sich näher in dem benötigten Stockwerk befinden, ausgewählt werden, um zu der entsprechenden Etage zu fahren.

Im folgenden Abschnitt wird erläutert, welche Strategien implementiert worden sind und wie ihre Funktionsweise ist.

### 4.3.1 Aufzugstrategien

Jeder Aufzug eines Gebäudes verfügt über eine Aufzugstrategie. Mit dieser Strategie entscheidet die Kabine, zu welchen Stockwerk sie als nächstes fährt. Jede Aufzugstrategie muss das Interface *IEleva-*

*torStragy* implementieren. Dieses Interface verfügt nur über eine einzige Methode, nämlich *nextFloor*. Diese Funktion entscheidet anhand eines Sets von Parametern, zu welchem Stockwerk sich der Aufzug als Nächstes bewegen soll. Für diese Entscheidung benötigt *nextFloor* zwei verschiedene Listen. In der einen Liste sind sämtliche Stockwerke eingetragen, auf denen Personen auf einen Aufzug warten. Die zweite Liste enthält sämtliche Stockwerke, zu denen die Personen innerhalb des Aufzugs fahren möchten. Weiters müssen die Aufzugkabine angegeben werden, die Anzahl der Stockwerke im Gebäude und die Richtung, in die sich der Aufzug vorher bewegt hat.

Im Laufe dieser Arbeit wurden die nachfolgend beschriebenen Aufzugstrategien implementiert.

#### Zufallsstrategie

Diese Strategie wählt ein zufälliges Stockwerk im Gebäude aus und retourniert es. Der Aufzug bewegt sich dadurch komplett zufällig durch das Gebäude, bis das richtige Stockwerk erreicht werden konnte. Der Aufzug bewegt sich dabei allerdings nur, wenn Passagiere eine Aufzugruftaste betätigen oder innerhalb der Kabine eine Taste gedrückt haben.

Diese Strategie wurde zu Beginn der Arbeit implementiert. Sie diente lediglich dazu, die allgemeine Funktionsweise der Aufzugsimulation zu testen und hat ansonsten keinen nennenswerten Nutzen.

#### Richtungsstrategie

Bei dieser Strategie wird der Aufzug zum obersten Stockwerk geschickt. Dort angekommen muss er wieder umdrehen und sich zum Erdgeschoß bewegen. Dabei bleibt der Aufzug auf jedem Stockwerk stehen, damit Personen ein- und aussteigen können. Der Aufzug bewegt sich nur, wenn eine Aufzugruftaste oder eine Taste innerhalb der Kabine betätigt worden ist. Wie bei der Zufallsstrategie wird nicht direkt zu den gewünschten Stockwerken der Personen gefahren.

Wie bei der Zufallsstrategie wurde diese Strategie zum Testen der Aufzugsimulation implementiert. Bei dieser Strategie können die Bewegungen der Aufzüge sehr leicht nachvollzogen werden. Dadurch ist die Richtungsstrategie gut geeignet, um die Basisfunktionen der Aufzugsimulation zu testen. Für realitätsnahe Simulationen ist diese Strategie aber nicht geeignet.

#### Verbesserte Richtungsstrategie

Diese Strategie stellt eine Verbesserung der Richtungsstrategie dar und ermöglicht ein realitätsnahes Verhalten der Aufzüge. Bei dieser Strategie beachtet der Aufzug die gewünschten Stockwerke der Passagiere. Sie steuert nur Etagen an, auf denen die Aufzugruftaste gedrückt worden ist oder zu denen Passagiere in der Kabine fahren möchten. Dabei fährt der Aufzug solange in eine Richtung, bis es keinen Auftrag mehr aus der Richtung gibt oder er das Erdgeschoß oder letzte Stockwerk erreicht hat. Danach dreht der Aufzug um und kümmert sich um die benötigten Stockwerke, die in der anderen Richtung liegen. Diese Strategie wurde definiert in [Strakosch u. Caporale 2010].

Beispielsweise fährt ein Aufzug nach unten und hat zwei Passagiere. Der erste Passagier will zum Erdgeschoß, der zweite ins oberste Stockwerk. Der Aufzug liefert dann zuerst den ersten Passagier beim Erdgeschoß ab und schaut nach, ob weitere Aufträge von weiter unten kommen können. Da er das unterste Stockwerk erreicht hat dreht er um und bringt die zweite Person ins oberste Stockwerk. Auf dem Weg dorthin würde er weitere Personen einsammeln, falls jemand die Aufzugruftaste betätigen würde. Falls nicht, dann würde er direkt zur letzten Etage fahren.

Diese Strategie ermöglicht es, die Aufzüge mit einem realitätsnahen Verhalten auszustatten. Deswegen sollte diese Strategie im Normalfall verwendet werden.

### 4.3.2 Queuestrategien

Diese Strategien teilen den Aufzügen die Stockwerk zu, auf denen Personen auf einen Aufzug warten. Sie schreiben die Etage in die Queue des Aufzugs, wodurch sich dieser später zum dem entsprechenden Stockwerk bewegt. Queuestrategien müssen das Interface *IQueueStrategy* implementieren. Wie bei der *IElevatorStrategy* verfügt dieses Interface nur über eine einzige Funktion, und zwar *queuesForCabins*. Diese Methode entscheidet darüber, welcher Aufzug als nächster ausgewählt wird. Dafür benötigt er die einzelnen Queues der Aufzüge, sämtliche Aufzüge des Gebäudes und das Stockwerk, welches zugewiesen werden soll.

Die nachfolgend beschriebenen Implementierungen von *IQueueStrategy* werden in dieser Arbeit entwickelt.

#### Zufallsstrategie

Diese Strategie wählt einen zufälligen Aufzug und weist ihm das Stockwerk zu. Es wird nicht darauf geachtet, wo sich die Aufzüge befinden und ob sie bereits zu einem anderen Stockwerk fahren.

Die Zufallsstrategie wurde zu Testzwecken implementiert, um ein einfaches Testen der Aufzusimulation zu ermöglichen. Außerhalb von Testzwecken macht eine Verwendung dieser Strategie wenig Sinn.

#### Strategie des nähesten Aufzugs

Hierbei wird das Stockwerk dem Aufzug zugewiesen, welches der Etage am nähesten ist. Die anderen Tätigkeiten des Aufzugs werden für diese Strategie ignoriert. Sind mehrere Aufzüge gleich nah von dem Stockwerk entfernt, dann wird immer der Aufzug mit der niedrigeren Identifikationsnummer gewählt.

Diese Strategie ist wesentlich sinnvoller als die Zufallsstrategie, weist allerdings noch ein paar Probleme auf. So kann es beispielsweise passieren, dass ein Aufzug für das Stockwerk ausgewähl ist, der zwar näher ist, aber gerade in die falsche Richtung fährt. Ein anderer Aufzug hätte das Stockwerk schneller erreichen können. Diese Strategie sollte also verfeinert werden, um auch andere Kriterien in die Entscheidung miteinfließen zu lassen.

#### Strategie des nähesten wartenden Aufzugs

Diese Strategie wählt nur Aufzüge aus, die zurzeit keiner Tätigkeit nachgehen und auf ihrem Stockwerk warten. Aus den wartenden Aufzügen wird dann jener ausgewählt, welcher der benötigten Etage am nähesten ist. Ist jeder Aufzug beschäftigt, dann wird die Kabine stattdessen mithilfe der *Strategie des nähesten Aufzugs* bestimmt.

Dadurch, dass nur wartende Aufzübe in die Entscheidung miteinbezogen werden, kann das Problem der *Strategie des nähesten Aufzugs* gelöst werden. Weiters werden die Aufträge so über die Aufzüge verteilt, dass auch weniger oft beschäftige Kabinen zum Einsatz kommen.

#### Strategie des nähestens Aufzugs in Richtung des Stockwerks

Bei dieser Strategie werden nur jene Aufzüge für die Berechnung miteinbezogen, die gerade in Richtung des benötigten Stockwerks fahren. Wie bei der *Strategie des nähesten wartenden Aufzugs* wird jene Kabine ausgewählt, die der entsprechenden Etage am nähensten ist. Fährt kein Aufzüg in die richtige Richtung, dann wird der Aufzug über die *Strategie des nähesten Aufzugs* bestimmt.

Diese Strategie stellt eine andere Möglichkeit da, um das Problem der *Strategie des nähestend Aufzugs* zu lösen. Im Gegensatz zur *Strategie des nähesten wartenden Aufzugs* werden bei dieser Strategie die Aufzüge priorisiert, die bereits in Bewegung sind.

## 4.4 Grafische Darstellung

In Abschnitt 3.4 wurden die verschiedenen Klassen und Interfaces vortgestellt, die notwendig sind, um eine grafische Darstellung der Aufzugsimulation zu ermöglichen. Im folgenden Abschnitt wird eine mögliche Implementierung für die Darstellung beschrieben. Es wird erläutert, mit welchen Mitteln die verschiedenen Tätigkeiten der Simulation visualisiert werden.

### 4.4.1 Aufbau der grafischen Darstellung

In der Klasse *ElevatorView*, welche das Interface *IView* (siehe Abschnitt 3.4) implementiert, werden die verschiedenen grafischen Objekte initialisiert. In der Implementierung dieser Arbeit werden Objekte von Java Swing verwendet. Java Swing wird eingesetzt, da es einen flexiblen Umgang mit den verschiedenen grafischen Objekten ermöglicht und der Zugriff auf die verschiedenen Eigenschaften leicht möglich ist [Ullenboom 2014].

In der grafischen Darstellung soll das Gebäude mit den verschiedenen Aufzügen erkennbar sein. Dafür wurden *JLabel*-Objekte benutzt, da sie es ermöglichen, einen beliebigen Text einzufügen, der nicht vom Benutzer verändert werden kann. Weiters können den Objekten der Klasse *JLabel* auch Bilder zugewiesen werden. Bestimmte Teile der Simulation, wie zum Beispiel die Aufzüge, können so durch verschiedene Bilder dargestellt werden. Weiters können Objekte der Klasse *JLabel* während der Laufzeit erstellt werden. Je nachdem, wieviele Stockwerke oder Aufzüge ein Gebäude hat, können dadurch entsprechend viele Objekte von *JLabel* erzeugt werden. Die verschiedenen *JLabel* werden in ein *GridLayout* gepackt. Dieses Layout positioniert die verschiedenen Objekte so in einem Raster, dass jedem Objekt gleich viel Platz zugewiesen wird. Bei Punkt 1 der Abbildung 13 sind die einzelnen *JLabel* und das *GridLayout* gut zu erkennen.

In der grafischen Darstellung werden zu jedem Stockwerk zusätzliche Informationen angezeigt (siehe Punkt 1 der Abbildung 13). In der ersten Spalte ist das Stockwerk angegeben, in der zweiten Spalte der Typ des Stockwerks. Die dritte Spalte gibt, an wieviele Personen sich auf dieser Etage befinden. In der vierten Spalte ist angegeben, wieviele Personen auf einen Aufzug warten. In den letzten drei Spalten werden die Personen, Aufzugruftasten und Aufzüge als Bilder dargestellt. Dabei steht bei den Aufzügen zusätzlich noch, welcher Tätigkeit sie nachgehen.

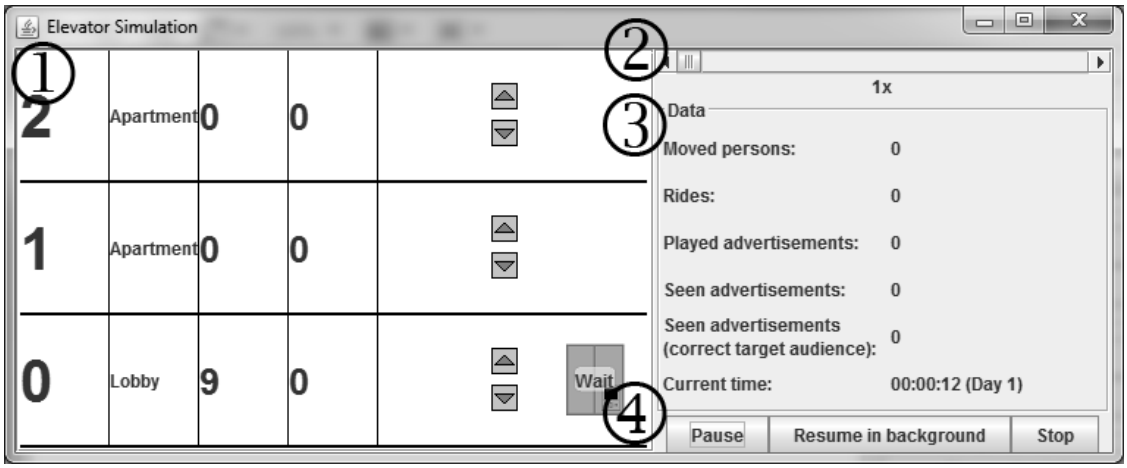


Abbildung 13: Bei dieser Simulation wird ein dreistöckiges Gebäude mit einem Aufzug dargestellt. Punkt 1 zeigt ein Raster mit den verschiedenen Informationen zu dem Gebäude. Bei Punkt 2 ist ein Schieberegler, mit dem die Geschwindigkeit eingestellt werden kann. In der rechten Spalte (Punkt 3) werden weitere Informationen zu der Simulation angezeigt. Mit den Buttons (Punkt 4) kann die Simulation pausiert oder gestoppt werden.

Durch die Verwendung von *JLabel* ist es leicht möglich, beliebig große Gebäude mit beliebig vielen Aufzügen zu erstellen. So zeigt Abbildung 14 ein Gebäude mit zehn Stockwerken und fünf Aufzügen, während das Gebäude in Abbildung 13 nur drei Etagen hat.

Auf der rechten Seite der Visualisierung werden Informationen über die Aufzugsimulation dargestellt (siehe Punkt 3 der Abbildung 13). Beispielsweise wird hier die Zeit in der Simulation angezeigt, wieiviele Personen bis jetzt Stockwerk gewechselt haben und wieviele Werbungen von den richtigen Zielgruppen gesehen worden sind. Der Schieberegler und die Buttons werden in Abschnitt 4.4.2 erläutert.

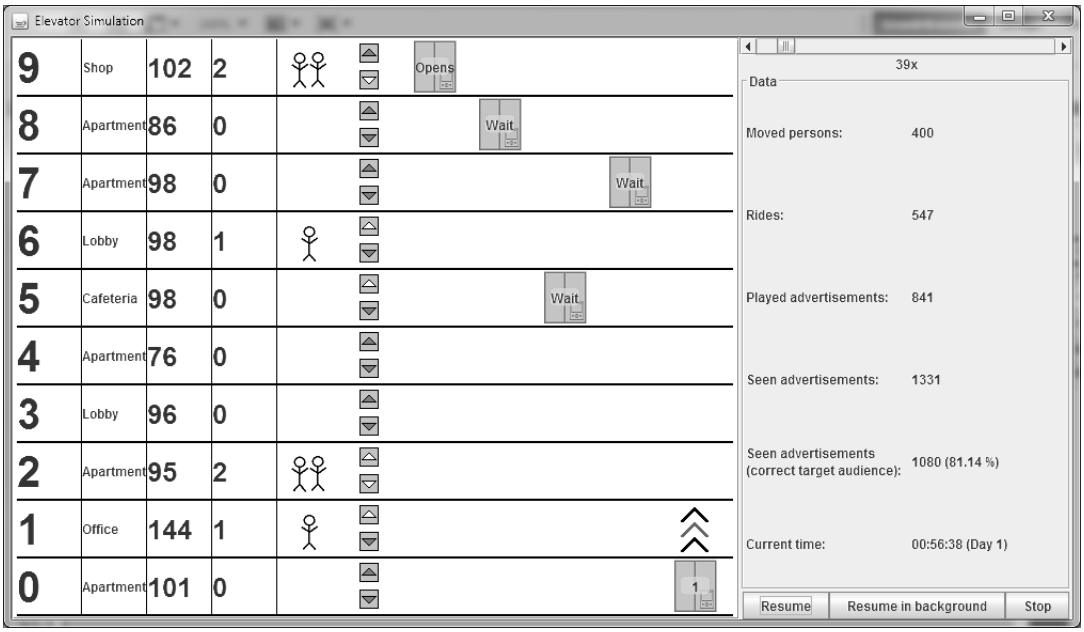


Abbildung 14: Bei dieser Simulation wollen sich bereits mehrere Personen durch das Stockwerk bewegen. Der Text bei den Aufzügen beschreibt dabei ihre Tätigkeit. Der Aufzug in der letzten Spalte zeigt über Pfeile, dass er sich nach oben bewegt.

### 4.4.2 Interaktionsmöglichkeiten

Dem Benutzer oder der Benutzerin stehen verschiedene Möglichkeiten zur Verfügung, um die grafische Darstellung der Simulation zu manipulieren. Die Geschwindigkei kann über einen Schieberegler (siehe Punkt 2 in Abbildung 13) eingestellt werden. Standardmäßig ist dieser Regler auf 1 gestellt, wodurch die Simulation in Echtzeit abgespielt wird. Der Benutzer oder die Benutzerin hat mit diesem Regler die Möglichkeit die Simulation um das bis zu 500-fache zu beschleunigen.

Rechts unten befinden sich drei Buttons (siehe Punk 4 in Abbildung 13). Mit dem Button „Pause“ oder „Resume“ kann die Simulation pausiert und wieder fortgesetzt werden. Pausieren ist besonders nützlich, um bestimmte Zeitpunkte der Simulation in Ruhe betrachten zu können. Der zweite Button „Resume in background“ schließt die grafische Darstellung der Simulation. Die Aufzugsimulation wird allerdings nicht beendet, sondern wird fertig berechnet. Der letzte Button „Stop“ schließt die grafische Darstellung und stoppt die Simulation. Beendet der Benutzer oder die Benutzerin das Fenster über das rote X rechts oben, dann wird die Simulation ebenfalls gestoppt.

Während der Simulation spielen die Aufzüge Werbungen ab. Welche Werbung während einer Fahrt abgespielt worden ist, kann auch über die grafische Darstellung eingesehen werden. Dafür muss der Benutzer oder die Benutzerin auf einen Aufzug klicken. Über eine Nachrichtenbox werden dann Daten zu den Personen und Werbungen angezeigt. In Abbildung 15 ist so eine Nachrichtenbox dargestellt.

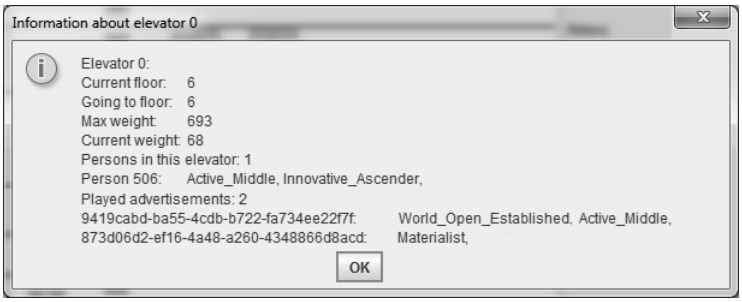


Abbildung 15: Jede Werbung und Person hat Zielgruppen zugewiesen. Bei diesem Beispiel wurde die Person 506 von der ersten Werbung angesprochen.

# 5 Evaluierung

In den vorhergehenden Kapiteln wurde die Aufzugsimulation geplant und implementiert. Um die Implementierung sinnvoll testen zu können, ist es notwendig, verschiedene Szenarien zu entwerfen und zu simulieren. Im folgenden Kapitel werden drei verschiedene Szenarien vorgestellt, mit denen die Aufzugsimulation getestet wurde.

## 5.1 Szenarien für die Simulation

Die Aufzugsimulation dieser Arbeit soll in der Lage sein, beliebig große Gebäude mit beliebig vielen Personen zu simulieren. Dementsprechend wäre ein Szenario gut geeignet, welches die Simulation auslastet. Die Personen sollten sich sehr viel bewegen, um dadurch erkennen zu können, wie die verschiedenen Strategien der Simulation darauf reagieren. Weiters sollten auch Szenarien von verschieden großen Gebäuden simuliert werden, in denen sich die Personen realistisch bewegen.

### 5.1.1 Szenario mit zufälligem Personenverhalten

Bei diesesm Szenario wird ein zehnstöckiges Gebäude mit fünf verschiedenen Aufzügen simuliert. Die einzelnen Stockwerke des Gebäudes sind dabei zufällig. In dem Gebäude werden 1000 verschiedene Personen simuliert, die alle über ein komplett zufälliges Verhalten verfügen. Die Personen bewegen sich dabei zur jeder Zeit zwischen den Stockwerken und beanspruchen die Aufzüge dadurch sehr oft. Durch dieses Szenario kann getestet werden, wie die verschiedenen Strategien der Simulation damit umgehen, wenn die Aufzüge ständig benötigt werden. Dadurch stellt dieses Szenario auch einen Härtetest für die Implementierung da.

Ein Vorteil dieses Szenarios ist, dass die zufälligen Personen sehr leicht generiert werden können. Dadurch können auch verschiedene zufällige Szenarien in kurzer Zeit erstellt und getestet werden. Nachteil dieser zufälligen Personen ist aber, dass auftretende Fehler möglicherweise nicht sofort reproduziebar sind.

### 5.1.2 Szenario mit kleinem Miethaus

Im nächsten Szenario wird ein fünfstöckiges Gebäude mit einem Aufzug entwickelt. Das Erdgeschoß des Miethauses beinhaltet die Lobby, während die restlichen Etagen Apartments sind. In der Simulation werden fünfzig Personen abgebildet, wobei die meisten davon Bewohner des Gebäudes simulieren. Die Personen weisen ein realistisches Verhalten auf und gehen bestimmten Tätigkeiten nach. Beispielsweise verlassen manche Personen das Gebäude zwischen 07:00 und 09:00, um zur Arbeit zu gehen (in der Simulation dargestellt durch die Lobby). Andere Personen dagegen bleiben im Gebäude und wechseln ihr Stockwerk, um Freunde zu besuchen. Am Abend kehren die meisten Personen wieder zurück.

Dieses Szenario versucht bereits das Verhalten möglichst realitsnah abzubilden. So entstehen Stoßzeiten bei den Aufzügen, wie zum Beispiel in der Früh und am Abend. Die Simulation muss in der Lage sein, mit diesen Stoßzeiten umzugehen, da diese Verhalten auch eine große Rolle in realen Aufzugsystemen spielen [Siikonen 2000]. Da dieses Szenario klein ist, kann es schnell simuliert werden.

### 5.1.3 Szenario mit großem Einkaufscenter und Hotel

Hierbei wird ein großes Einkaufscenter mit einem Hotel simuliert. Das Gebäude ist zwanzig Stockwerke hoch und verfügt über Geschäfte, Restaurants, Büros und Apartments. Sieben Aufzüge sind im Gebäude untergebracht. 2000 Personen werden sich zu verschiedenen Zeiten im Einkaufscenter bewegen, je nachdem ob sie Käufer, Angestellte oder Besucher des Hotels sind.

Wie bei dem Szenario bei Abschnitt 5.1.2 wird es auch hier zu Stoßzeiten kommen. Durch die höhere Zahl der Personen wird der Andrang allerdings wesentlich größer sein. Die Aufzüge der Simulation muss in der Lage sein die große Anzahl von Personen möglichst schnell zu transportieren. Die verschiedenen Stockwerke dieses Gebäudes erlauben ein komplexes Verhalten der Personen. Die Persone können dadurch sehr genau gesteuert werden, um so bestimmte Situationen in der Simulation herbeizuführen. Durch die hohe Anzahl an Personen wird die Berechnung dieser Simulation allerdings viel Zeit beanspruchen.

## 5.2 Testen der Szenarien

In der Testphase wird jedes Szenario viermal durchgeführt. Es wird kontrolliert, ob jede Simulation ohne Probleme durchgeführt werden kann und ob die benötigten Aufzugfahrten generiert werden. Es wird beschrieben ob bestimmte Verhalten oder Situationen zu Problemen geführt haben. Bei den vier Testläufen jedes Szenarios erfolgen zwei ohne grafische Darstellung, die anderen mit. Dadurch soll kontrolliert werden, ob die Darstellung Auswirkungen auf die Simulation hat. Die Simulationen, bei denen die Darstellung angezeigt wurde, wurden mit 500-facher Beschleunigung simuliert. Ansonsten würden die Testläufe zu viel Zeit beanspruchen. Jedes Szenario wurde einen Simulationstag lang abgespielt. Jeder Aufzug verwendete die *Verbesserte Richtungsstrategy* (siehe Abschnitt 4.3.1). Als Queuestrategie wurde bei jeder Simulation die *Strategie des nähesten wartenden Aufzugs* (siehe Abschnitt 4.3.2) eingesetzt.

Nach den Testläufen wird angegeben, wieviele Fahrten unternommen worden sind und wie lange die Simulation angedauert hat. Nachdem jedes Szenario getestet worden ist, werden die Erkenntnisse zusammengefasst, um Rückschlüsse daraus ziehen zu können.

### 5.2.1 Testläufe der zufälligen Simulation

Bei den ersten Testläufen wurde das Szenario aus Abschnitt 5.1.1 simuliert. Die Resultate dieser Testläufe sind in Tabelle 1 einzusehen.

| Testlauf | Grafische Darstellung | Laufzeit | Aufzugfahrten | Durchschnittliche Fahrt | Längste Fahrt | Kürzeste Fahrt |
|---|---|---|---|---|---|---|
| #1 | Nein | 23,06 s | 13.245 | 16 s | 52 s | 8 s |
| #2 | Nein | 38,29 s | 13.690 | 15 s | 52 s | 9 s |
| #3 | Ja | 3 m 20,33 s | 12.835 | 16 s | 53 s | 9 s |
| #4 | Ja | 3 m 36,94 s | 12.767 | 15 s | 44 s | 10 s |

Tabelle 1: Neben der Laufzeit und der Anzahl der Fahrten wird auch die längste und kürzeste Fahrt jedes Testlaufs angegeben.

Obwohl die Personen und Gebäude in jedem Testlauf zufällig generiert sind, unterscheiden sich die Werte nicht stark voneinander. Die einzige Ausnahme bilden die Laufzeiten bei den Testläufen mit grafischer Darstellung. Das liegt daran, dass die Simulation durch Anzeigen der Visualisierung sehr stark verlangsamt wird. Bemerkenswert ist die hohe Zahl an Aufzugfahrten. Der Grund dafür ist das zufällig generierte Personenverhalten. Die Personen fahren die ganze Simulation über mit den Aufzügen, wodurch entsprechend viele Fahrten erzeugt werden.

Die Dauer der längsten Fahren und kürzesten Fahrt scheinen auch vernünftig zu sein. Die Zeiten der längsten Fahrten sind entstanden, wenn Personen vom Erdgeschoß in das letzte Stockwerk gefahren sind, oder auch umgekehrt. Bei diesen Testläufen ist zu beachten, dass die längste und kürzeste Fahrt nicht vom selben Aufzug stammt. Da die Aufzüge ebenfalls zufällig generiert sind, kommt es häufig vor, dass verschieden schnelle Aufzüge im selben Gebäude vorkommen.

In sämtlichen Testläufen sind keine Fehler aufgetreten. Die Aufzüge konnten die Personen zu den richtigen Stockwerken transportieren, ohne dass ein fehlerhaftes Verhalten aufgetreten ist.

### 5.2.2 Testläufe des kleinen Miethauses

Diese Testläufe beschäftigen sich mit der Simulation des Szenarios aus Abschnitt 5.1.2. In Tabelle 2 sind die Ergebnisse dieser Simulationen erkennbar.

| Testlauf | Grafische Darstellung | Laufzeit | Aufzugfahrten | Durchschnittliche Fahrt | Längste Fahrt | Kürzeste Fahrt |
|---|---|---|---|---|---|---|
| #1 | Nein | 3,54 s | 353 | 15 s | 23 s | 11 s |
| #2 | Nein | 3,69 s | 350 | 15 s | 23 s | 11 s |
| #3 | Ja | 2 m 57,16 s | 353 | 15 s | 23 s | 11 s |
| #4 | Ja | 2 m 55,96 s | 351 | 15 s | 23 s | 11 s |

Tabelle 2: Die Ergebnisse der verschiedenen Testläufe sind, mit Ausnahme der benötigten Zeiten, sehr ähnlich.

Bei diesen vier Testläufen sind die Werte der einzelnen Simulationen sehr nah beieinander. Der einzige größere Unterschied ist hier wieder die benötigte Zeit bei den Testläufen ohne und mit grafischer Darstellung. Bei der längsten Fahrt von 23 Sekunden ist der Aufzug vom Erdgeschoß ins letzte Stock gefahren. Wird die Summe der benötigten Zeit für jede Tätigkeit gebildet, so entspricht das dem gefundenen Ergebnis der Testläufe (Tür schließen [3 Sekunden] + 4 Stockwerke [4 * 4 Sekunden] + Tür öffnen [3 Sekunden]). Bei der kürzesten Fahrt hat sich der Aufzug dagegen nur um ein Stockwerk bewegt.

Während der Simulation konnte kein seltsames Verhalten der Aufzüge oder Personen festgestellt werden. Die Kabine war in der Lage, sämtliche Personen zum richtigen Stockwerk zu bewegen, ohne dass es zu Zwischenfällen gekommen ist.

### 5.2.3 Testläufe des Einkaufscenters

Die letzten vier Testläufe beschäftigen sich mit dem Szenario des Abschnitts 5.1.3. Die Ergebnisse dieser Simulationen sind in Tabelle 3 einsehbar.

| Testlauf | Grafische Darstellung | Laufzeit | Aufzugfahrten | Durchschnittliche Fahrt | Längste Fahrt | Kürzeste Fahrt |
|---|---|---|---|---|---|---|
| #1 | Nein | 33,83 s | 6550 | 23 s | 1 m 42 s | 11 s |
| #2 | Nein | 32,09 s | 6467 | 23 s | 1 m 42 s | 11 s |
| #3 | Ja | 3 m 8,30 s | 6719 | 23 s | 1 m 42 s | 11 s |
| #4 | Ja | 3 m 3,44 s | 6591 | 23 s | 1 m 42 s | 11 s |

Tabelle 3: Im Vergleich zu den anderen Szenarien ist die durchschnittliche Fahrzeit bei diesem besonders hoch. Grund dafür ist das größere Gebäude.

Die Verwendung der grafischen Darstellung benötigt wieder wesentlich mehr Zeit. Ansonsten unterscheiden sich die einzelnen Ergebnisse aber nicht stark voneinander. Dass die längste Fahrt 1 Minute und 42 Sekunden dauert, ist möglich durch das 20-stöckige Haus. Manche Personen sind direkt vom Erdgeschoß in das oberste Geschoß gefahren oder umgekehrt.

Sämtliche Personen konnten zwar zu ihrem gewünschten Stockwerken gebracht werden, allerdings sind bei diesen Testläufen seltsame Verhalten aufgetreten. Besonders viele Besucher des Einkaufscenters haben sich nicht richtig im Gebäude verteilt, sondern sind nur zum nähesten Stockwerk mit dem richtigen Typ gereist. Dadurch ist es passiert, dass eine Etage mit Geschäften über 1000 Besucher hatte, während auf den anderen Stockwerken desselben Typs nur 50 bis 200 Personen waren. Die Simulation war so eingestellt, dass sich diese Besucher eigentlich gleichmäßig verteilen hätten sollen.

Die Aufzüge haben zwar ihr Verhalten richtig umgesetzt, allerdings haben sie nicht gut mit den Stoßzeiten umgehen können. Wenn sehr viele Personen auf einem Stockwerk warteten, dann ist trotzdem immer nur ein Aufzug dorthin gefahren. Optimalerweise hätten sich mehrere Aufzüge zu diesem Stockwerk bewegen müssen, da eine Kabine allein nicht ausgereicht hat. Die Queuestrategie (siehe Abschnitt 4.3.2) sollte deswegen angepasst werden, damit die Aufzüge richtig auf die Stoßzeiten reagieren können.

### 5.2.4 Resultate der Testläufe

Sämtliche in Abschnitt 5.1 vorgestellte Szenarien konnten fast problemfrei simuliert werden. Auch die Ergebnisse für die verschiedenen Testfälle sind sinnvoll. So erzeugen die Simulationen mit zufälligen Personen besonders viele Aufzugfahrten, da die Passagiere hier ständig mit dem Aufzug fahren möchten. Bei den anderen beiden Szenarien wurden realistische Verhalten modelliert. Die Anzahl der generierten Fahrten ist bei diesen Szenarien auch weitaus niedriger ausgefallen.

Einziges Problem war bei den Testläufen des Einkaufscenters (siehe Abschnitt 5.2.3). Die Personen haben sich nicht richtig in den Stockwerken verteilt, sondern haben die unteren Etagen bevorzugt. Bei diesen Testläufen konnte auch festgestellt werden, dass die Queuestrategie (siehe Abschnitt 4.3.2) nicht gut auf Stoßzeiten bei den Aufzügen reagiert. Deswegen sollte eine Strategie implementiert werden, die auch richtig mit Stoßzeiten umgehen kann.

# 6 Zusammenfassung

Dieses Kapitel beschäftigt sich mit der Zusammenfassung und Diskussion sämtlicher Ergebnisse. Zusätzlich wird noch beschrieben, welche Vorgänge und Teile der Implementierung verbessert werden können.

## 6.1 Resultate

Diese Arbeit beschäftigte sich mit der Implementierung einer Aufzugsimulation, um Zeitpläne für Werbungen zu testen. Dafür wurden zuerst andere Programme getestet und beschrieben, die sich mit dem Thema der Aufzugsimulation beschäftigen. Die verschiedenen Implementierungen wurden auf ihre Stärken und Schwächen untersucht. Mit dem Wissen, wie die einzelnen Programme die Simulationen durchführen, wurde damit begonnen eine eigene Implementierung zu entwickeln.

Es wurden verschiedene Möglichkeiten vorgestellt, die beschreiben, wie das Verhalten der Personen und Aufzüge in der Simulation umgesetzt werden kann. Um die Simulation zu implementieren, wurden die Manager-Klassen entwickelt, die verschiedene Teile der Simulation steuern. Zusätzlich wurden auch Interfaces implementiert, die es ermöglichen, eine grafische Darstellung der Aufzugsimulation zu entwickeln. Über diese Darstellung war es dann leicht möglich, die Vorgänge innerhalb der Simulation zu beobachten und zu kontrollieren.

Um die Implementierung zu testen wurden, verschiedene Szenarien entworfen. Jedes Szenario wurde viermal durchgeführt, um potenzielle Fehlerquellen zu finden. Das Programm war in der Lage sämtliche Szenarien zu simulieren. Es wurden die benötigten Aufzugfahrten generiert und auch die anderen Werte waren in einem, dem Szenario entsprechenden, sinnvollen Rahmen. Dabei wurde auch festgestellt, dass die verwendete Queuestrategie nicht gut mit Stoßzeiten bei den Aufzügen umgeht. Weiters verteilen sich die Personen in manchen Fällen nicht wie gewünscht regelmäßig im Gebäude, sondern bevorzugen die unteren Etagen.

## 6.2 Diskussion

Bei der Bewertung der anderen Aufzugsimulationen ist aufgefallen, dass sehr wenig Kontrolle über die Personen möglich ist. Die meisten der getesten Simulationen haben das Personenverhalten zufällig erzeugt, wodurch es nicht möglich war, spezifische Situationen zu simulieren. Zusätzlich waren auch die Einstellungsmöglichkeiten dieser Aufzugsimulationen sehr eingeschränkt. Dadurch eigneten sich diese Programme nicht, um Aufzugfahrten mt Werbungen zu generieren, über die dann die Zeitpläne getestet werden.

Die Implementierung dieser Arbeit setzt das Personenverhalten so um, dass es möglich ist, komplexe Situationen gezielt herbeizuführen. Es ist möglich genau zu definieren, wann welche Person sich zu welchem Stockwerk bewegen soll. Dadurch ist die Implementierung dieser Arbeit weitaus flexibler als die der anderen vorgestellen Simulationsprogramme (siehe Abschnitt 2.2).

Die Testläufe der verschiedenen Simulationen hat ergeben, dass die Aufzugsimulation großteils richtig funktioniert. Je nach Einstellung wollen die Personen zu einem anderen Stockwerk. Die Aufzüge reagieren richtig auf die Wünsche der Personen und bringen sie zu dem richtigen Stockwerk. Dies wurde mithilfe der grafischen Darstellung kontrolliert, die es ermöglicht, die verschiedenen Tätigkeiten der Aufzugsimulation zu beobachten.

Die Testläufe des Szenarios (Abschnitt 5.1.3) haben einen Fehler im Verhalten der Personen aufgedeckt. Obwohl sich die Personen gleichmäßig im Gebäude verteilen hätten sollen, haben sie dennoch die unteren Etagen bevorzugt. In den unteren Etagen haben sich dann zu viele Personen angesammelt. Es sollte die Stelle des Personenverhaltens überprüft werden, die entscheidet, welches Stockwerk eine Person besuchen möchte.

Bei den selben Testläufen konnte auch ein Problem bei der Queuestrategie (siehe Abschnitt 4.3.2) aufgedeckt werden. Wenn sich besonders viele Personen auf einem Stockwerk befinden und den Aufzug benutzen wollen, dann kümmert sich nur ein Aufzug um diese Passagiere. Optimalerweise sollten hier mehrere Aufzüge diese Stockwerke besuchen, da nicht alle in einen Aufzug passen. Dieses Problem kann behoben werden, indem die bereits existierenden Strategien so erweitert werden, dass auch kontrolliert wird, ob ein Aufzug allein die Personen befördern kann. Eine weitere Möglichkeit wäre das Implementieren einer neuen Queuestrategie, die darauf ausgelegt ist, mit Stoßzeiten umzugehen.

## 6.3 Ausblick

Die Aufzugsimulation dieser Arbeit wurde so implementiert, dass es leicht möglich ist bestimmte Teile der Simulation zu erweitern oder auszutauschen. Es exisiteren Interfaces für jede Manager-Klasse. Dadurch wäre es möglich, ein vollkommen neues Personen- oder Aufzugverhalten zu entwickeln. Diese Interfaces bieten dadurch die Möglichkeit Aufzugsimulationen zu entwerfen, die sich in ihrem Ablauf grundlegend unterscheiden. Im Laufe dieser Arbeit ist nur eine Implementierung für jede Manager-Klasse erfolgt. Hier wäre noch großes Potenzial vorhanden, um das Programm zu erweitern.

Neben den Manager-Klassen gibt es auch Interfaces für die Aufzug- und Queustrategien. Es wurden zwahr mehrere verschiedene Klassen von jeder Strategie implementiert, allerdings könnten noch weitere Strategien entworfen werden, welche die Aufzüge anders steuern. Beim Testen der Simulation hat sich zum Beispiel herausgestellt, dass die vorhandenen Strategien nicht gut mit sehr vielen Personen gleichzeitig umgehen können. Das Implementieren einer eigenen Strategie die sich darum kümmert, wäre dadurch angebracht. Beispielsweise kann hier die derzeitige Last des Aufzugs zur Hilfe genommen werden, um zu ermitteln ob weitere Aufzüge notwendig sind, um sämtliche Passagiere befördern zu können [Siikonen u. Roschier 1995].

Während den Testläufen wurden verschiedene Szenarien getestet. Dabei wurde immer dieselbe Aufzug- und Queuestrategie verwendet. In einer längeren Testphase wäre es möglich, zusätzlich auch die Strategien auszutauschen, um dadurch zu neuen Erkenntnissen zu kommen. Dadurch ist es möglich festzulegen, welche Strategie in welchem Szenario gut geeignet ist, oder welche Strategien allgemein besser abschneiden. Beispielsweise könnten sich manche Aufzugstrategien besser für größere Gebäude eignen als andere [Siikonen 1997].

# Literatur


[Berndt u. a. 2004] Berndt, Daniela ; Blaser, Hans ; Gröss, Stefan ; Schmickal, Andreas ; Wimmer, Mario: *ELVISS (Elevatorsimulation: Visualisierung, Simulation und Steuerung eines Fahrstuhlsystems)* / Fachhochschule Rosenheim, Deutschland. 2004. – Forschungsbericht

[Dailey 2014] Dailey, Chris: *Elevator Sim 0.4*. `http://www.softpedia.com/get/Science-CAD/Elevator-Simulator.shtml`. Version: 4 2014. – Letzter Aufruf am 12.04.2015

[Khiang u. a. 2010] Khiang, Tan K. ; Khalid, Marzuki ; Yusof, Rubiyah: *Intelligen Elevator Control by Ordinal Structure Fuzzy Logic Algorithm* / Universiti Teknologi Malaysia. 2010. – Forschungsbericht

[Kimmey 2008] Kimmey, Jason: *Elevator Simulator*. `https://www.chasemaier.com/project/Elevator-Simulator`. Version: 4 2008. – Letzter Aufruf am 12.04.2015

[Krüger u. Hansen 2011] Krüger, Guido ; Hansen, Heiko: *Handbuch der Java-Programmierung*. Addison-Wesley Verlag, 2011

[Newell 1998] Newell, Gordon F.: *Strategies for Serving Peak Elevator Traffic* / Institute of Transportation Studies, University of California. 1998. – Forschungsbericht

[Ören 2005] Ören, Tuncer I.: *Toward the Body of Knowledge of Modeling and Simulation* / University of Ottawa, Canada. 2005. – Forschungsbericht

[Siikonen 1997] Siikonen, Marja-Liisa: *Planning and Control Models for Elevators in High-Rise Buildings* / Helsinki University of Technology. 1997. – Forschungsbericht

[Siikonen 2000] Siikonen, Marja-Liisa: *On Traffic Planning Methodology*. In: *Elevator Technology 10*, 2000

[Siikonen u. Roschier 1995] Siikonen, Marja-Liisa ; Roschier, Nils-Robert: *Determination of the Number of Persons Entering and Leaving an Elevator Car*. Patent, 1995

[Sokolowski u. Banks 2011] Sokolowski, John A. ; Banks, Catherine M.: *Principles of Modeling and Simulation: A Multidisciplinary Approach*. Wiley, 2011

[Stachowiak 1973] Stachowiak, Herbert: *Allgemeine Modelltheorie*. Springer, 1973

[Strakosch u. Caporale 2010] Strakosch, George R. ; Caporale, Robert S.: *The Vertical Transportation Handbook*. John Wiley & Sons, 2010

[Tuomas u. a. 2004] Tuomas, Susi ; Sorsa, Janne ; Siikonen, Marja-Liisa: *Passenger Behaviour in Elevator Simulation*. In: *Elevator Technology 14*, 2004

[Ullenboom 2014] Ullenboom, Christian: *Java ist auch eine Insel: Insel 1: Das umfassende Handbuch*. Galileo Computing, 2014

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Programm-Listings