Konzeptionierung und Aufbau einer Simulationsumgebung zur Abbildung von Bewegungen von Bodenfahrzeugen an Flughäfen

Studienarbeit

Technische Universität Braunschweig

DLR

Aufgabenstellung

Thema:

Konzeptionierung und Aufbau einer Simulationsumgebung zur Abbildung von Bewegungen von Bodenfahrzeugen an Flughäfen

Concept and Development of a simulation environment to model operations of ground handling service vehicles at airports.

Ziel:

Ziel der der vorliegenden Ausschreibung ist die Konzeptionierung und Erstellung einer Fahrzeugsimulation, die die grundlegenden Funktionalitäten zur Abbildung der Bewegung von Vorfeldfahrzeugen abbildet. Hierzu soll zunächst eine Anforderungsanalyse an eine solche Simulation erstellt werden, die operationelle (z.B. Bewegungsmuster verschiedener Fahrzeugtypen) und technische Rahmenbedingungen (Programmiersprachen, Systemanforderungen) als auch bestehende Simulations- und Arbeitsumgebungen des DLR mit einbezieht (Interfaces, Kompatibilität, Datenaustausch). Aufbauend darauf ist ein Konzept zur Umsetzung zu beschreiben, welches eine Architektur und mögliche Systemkomponenten definiert. Abschließend soll eine erste technische Umsetzung begonnen werden.

Durchzuführende Arbeiten:

- Anforderungsanalyse an eine Fahrzeugsimulation für Bodenfahrzeuge an Flughäfen
- Erstellung eines Konzeptes zur Umsetzung (modulares, erweiterbares Konzept, Einbeziehung bestehender DLR-FL Simulations- und Entwicklungsumgebungen)
- Aufbau erster Teilkomponenten.

Eidesstattliche Erklärung

Hiermit erkläre ich, Stefan Frömel, geb. am 12.03.1990, an Eides statt, die vorliegende Studienarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet zu haben.

Braunschweig, 30. Juni 2023

Dokumenteigenschaften

Titel	Konzeptionierung und Aufbau einer Simulationsumgebung zur Abbildung von Bewegungen von Bodenfahrzeugen an Flughäfen
Betreff	Studienarbeit
Martrikelnummer	4751579
Institut	Institut für Flugführung
Autor	Stefan Frömel
Zugänglichkeit	C/II (im Institut für Flugführung unbegrenzt zugänglich)
Freigabe	Die Freigabe erfolgt im gesondertem Freigabeprozess
Datum	2023-06-30
Version	1.0
Dateipfad	SF-Studienarbeit.docx

Abstract

Am Institut für Flugführung des Deutschen Zentrums für Luft- und Raumfahrttechnik wird derzeit am Projekt „Digitaler Flughafen“ gearbeitet. Teil dieses Projekts ist eine Automatisierung der Bewegung der Vorfeldfahrzeuge, um den Prozess der Bodenabfertigung zu optimieren, sodass die Zuverlässigkeit erhöht wird und Verspätungen vermieden werden.

In dieser Studienarbeit wird ein Konzept für eine Verkehrssimulation der Bodenabfertigung erstellt, sowie ein erster Prototyp entwickelt. Es werden theoretische Grundlagen dargelegt, die als Basis für die Anforderungen der Simulation dienen. Des Weiteren werden diese Anforderungen tabellarisch dokumentiert und beschrieben. Ein modularer Aufbau wird konzipiert und es wird dargelegt, welche Programmierarbeiten bereits durchgeführt werden konnten. Am Schluss folgt eine Zusammenfassung, sowie ein Ausblick auf die nächsten durchzuführenden Schritte.

Inhaltsverzeichnis

1. Einleitung

Flughäfen sind heutzutage komplexe Systeme, in denen präzises Timing von großer Bedeutung ist. Jede Verzögerung kann zu weiteren Verspätungen und Kosten führen. Insbesondere bei der Bodenabfertigung können Verspätungen weitreichende Auswirkungen haben und hohe Kosten verursachen.

Trotz moderner Technologien sind Verspätungen bei der Bodenabfertigung keine Seltenheit (s. Kapitel 2.3). Diese werden durch eine Vielzahl von Einflussfaktoren, wie etwa Wetterbedingungen, menschliche Fehler oder technische Störungen, verursacht und führen zu unerwarteten Verzögerungen. Aus diesem Grund besteht ein wachsender Bedarf an Lösungen zur Verbesserung der Effizienz und der Zuverlässigkeit der Bodenabfertigung.

Die Optimierung der Bodenabfertigungsprozesse ist ein Bereich, der großes Potential für Verbesserungen bietet. Durch die effiziente Nutzung der Vorfeldfahrzeuge und einer Optimierung der Prozesse, können Verspätungen vermieden und Ressourcen besser ausgenutzt werden. Dafür ist es zunächst notwendig, ein genaues Verständnis des Ablaufs der Bodenabfertigung zu haben, weshalb Kapitel 2 sich mit diesem Thema befasst.

Die in dieser Studienarbeit konzipierte Verkehrssimulation zielt darauf ab, ein besseres Bild der Dynamik der Bodenabfertigung zu entwickeln und soll so dazu beitragen, mögliche Engpässe zu identifizieren und Verbesserungen ersichtlich zu machen.

Darüber hinaus ist die Studienarbeit auch ein Schritt in Richtung der Automatisierung der Bodenabfertigung. Durch die Simulation und Analyse der Bewegungsmuster verschiedener Fahrzeugtypen, kann ein Beitrag zur Entwicklung der nötigen Algorithmen für Autonome Fahrzeuge geleistet werden, womit die Effizienz und Zuverlässigkeit der Bodenabfertigung verbessert werden kann.

2. Bodenabfertigung

In diesem Kapitel wird der Ablauf der Bodenabfertigung beschrieben. Dieses Wissen ist notwendig, um einen Überblick über die Verkehrsdynamik auf dem Flughafen zu erhalten. Außerdem wird betrachtet, wo bei der Bodenabfertigung Verspätungen entstehen und inwiefern der Ablauf der Bodenabfertigung variieren kann.

2.1. Grundlagen

Die Bodenabfertigung, auch bekannt als Ground Handling, und dessen Optimierung ist für die Airlines von großer Bedeutung, denn diese ist für einen reibungslosen und verzögerungsfreien Ablauf des Luftverkehrs zuständig. In der Regel wird die Bodenabfertigung von spezialisierten Unternehmen, den sogenannten *Ground Handlers*, durchgeführt, die im Auftrag der Fluggesellschaften handeln. Ziele der Bodenabfertigung sind unter anderem die Verfügbarkeit von Ressourcen und die optimale Ausnutzung der Abfertigungsgerätschaften und des Personals zu gewährleisten. Es ist auch die Aufgabe der Bodenabfertigung die Serviceleistungen und Servicezeiten einzuhalten [1], denn viele der Verspätungen sind auf Verzögerungen in der Bodenabfertigung zurückzuführen. In Kapitel 2.3 werden Verspätungen näher erläutert. Somit gibt es aus Sicht der Fluggesellschaften ein großes Interesse an der Optimierung der Bodenabfertigung.

2.2. Bodenabfertigungsprozesse

Eine Möglichkeit die Prozesse der Bodenabfertigung zu unterscheiden, ist die Einteilung in landseitige und luftseitige Bereiche. Der landseitige Bereich umfasst die Abfertigung von Fracht, Gepäck und Passagieren im Terminal. Zur luftseitigen Abfertigung gehören dahingegen die Prozesse am Flugzeug, wie der Transport von Passagieren, Fracht und Gepäck, sowie die Versorgung mit Strom, Treibstoff und Verpflegung [2]. Zudem wird die luftseitige Abfertigung zwischen Bereitstellung, Turnaround und Abzug unterschieden. Bei der Bereitstellung wird ein Flugzeug, welches in einem Hangar oder auf einer Parkposition abgestellt ist, für den nächsten Flug

vorbereitet. Der Abzug bezeichnet den Vorgang ein einkommendes Flugzeug in einen Hangar oder auf eine Parkposition zu schleppen. Der Turnaround hingegen ist der Standard-Fall, bei dem ein Flugzeug nach der Versorgung für den nächsten Flug umgedreht wird [2]. Abbildung 1: Beispielhafter Aufbau der Geräte und Fahrzeuge bei der Bodenabfertigung . zeigt einen allgemeinen Aufbau der für die Bodenabfertigung verwendeten Geräte und Fahrzeuge. Die Passagiere steigen im Normalfall auf der linken Seite ein und aus, auf der rechten Seite finden die meisten Ent- und Versorgungsprozesse statt. Im Folgenden werden die Prozesse der luftseitigen Abfertigung im Turnaround kurz erläutert.

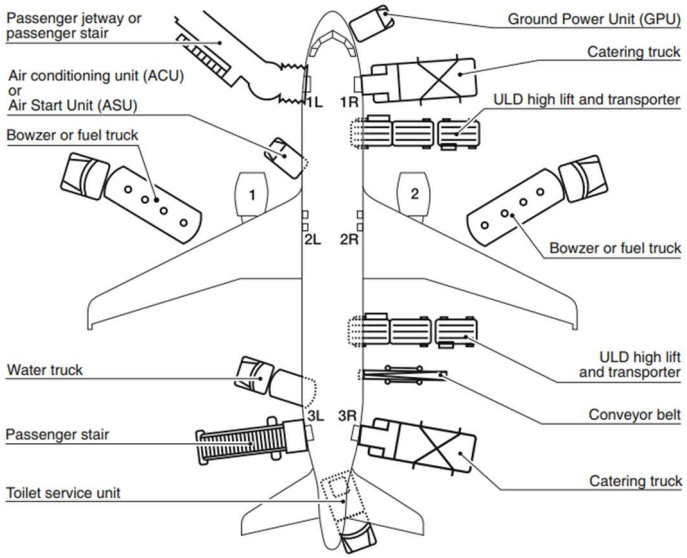

Abbildung 1: Beispielhafter Aufbau der Geräte und Fahrzeuge bei der Bodenabfertigung [2].

Bereitstellen von Fahrzeugen und Geräten:

> Vor dem Eintreffen des Flugzeuges an der Parkposition, werden Fahrzeuge und Geräte, wie die Fluggasttreppen und -busse, Gepäckbänder, Gepäckwagenschlepper, Trolleys und die *Ground Power Unit* (GPU), sowie die *Air Starter Unit* (ASU) in der Nähe bereitgestellt [1].

Aussteigen der Passagiere (Deboarding):

> Bei diesem Vorgang werden zunächst die Fluggasttreppen und -busse an die entsprechende Position gefahren. Die Türen werden geöffnet, die Fluggäste stehen auf und verlassen das Flugzeug. Alternativ kann eine Fluggastbrücke verwendet werden, dann sind weder Treppen noch Busse notwendig [1].

Ausladen von Gepäck und Fracht (Unloading):

> Zeitgleich zum *Deboarding* findet das *Unloading* statt. Es werden die Laderaumtüren geöffnet und das Gepäck und die Fracht ausgeladen [1].

Reinigen der Kabine (Cleaning):

Nach dem Aussteigen der Passagiere beginnt die Reinigung der Kabinen. Diese fällt unterschiedlich aus, je nachdem, welche Fluggesellschaft beteiligt ist oder um was für eine Art von Flug es sich handelt. Bei einem Transitflug werden bspw. die oberen Gepäckfächer und die Bordküche nicht gereinigt. Die *International Air Transport Association* (IATA) schreibt vor, dass im *Standard Cleaning* Abfälle beseitigt, Klapptische abgewischt und Sitze, Gänge, Bordküchen und Toiletten gereinigt werden müssen [1].

Betankung (Refueling):

Aus Sicherheitsgründen kann die Betankung erst dann starten, wenn der letzte Passagier das Flugzeug verlassen hat. Zunächst fährt das Tankfahrzeug auf die vorgesehene Position. Die Tankverbindung wird hergestellt, woraufhin der Tankvorgang beginnt. Bei der Fertigstellung wird die Verbindung getrennt und das Tankfahrzeug entfernt sich. Je nach Flugzeugtyp kann es vorkommen, dass die Tragflächentanks getrennt befüllt werden müssen. In diesem Fall muss zusätzliche Zeit für das Tankfahrzeug bereitgestellt oder ein zweites hinzugezogen werden [1].

Anlieferung der Bordverpflegung (Catering):

Das Catering beginnt mit dem Positionieren des Catering-Fahrzeuges. Die Service-Tür wird geöffnet, die Bordküche wird mit Lebensmitteln versorgt und die leeren Servierwagen werden durch volle ausgetauscht. Abhängig davon, ob eine zweite Bordküche vorhanden ist, kann es vorkommen, dass das Fahrzeug umgesetzt werden muss, um auch diese zu beliefern. Eine Alternative ist ein zweites Catering-Fahrzeug, was aber von der Catering-Firma abhängig ist. Bei Low-Cost Flügen ändert sich der Ablauf. Hier werden oftmals nur morgens die Lebensmittel aufgefüllt und müssen dann für den ganzen Tag reichen [1].

Schmutzwasserentsorgung (Waste Water Service):

Um das Schmutzwasser zu entsorgen, muss zunächst das Fahrzeug positioniert werden. Daraufhin wir die Verbindung zum Abwassersystem des Flugzeugs hergestellt und das Schmutzwasser wird abgepumpt. Die Verbindung wird wieder getrennt und das Fahrzeug verlässt seine Position. Falls mehrere Toiletten vorhanden sind, wird der Vorgang wiederholt [1].

Frischwasserversorgung (Portable Water Service):

Für die Frischwasserversorgung wird das Fahrzeug auf der vorgesehenen Position geparkt. Die Anschlüsse werden verbunden und die Tanks aufgefüllt. Bei der Fertigstellung verlässt das Fahrzeug seine Position [1].

Einladen des Gepäcks und der Fracht:

Es werden zunächst die beladenen Gepäckwagen an ihrer Parkposition positioniert. Das Flugzeug wird gemäß Ladeplan mit dem Gepäck und der Fracht beladen, die Fracht wird gesichert, die Frachtraumnetzte und die Frachtraumtüren werden geschlossen. Die Gepäckwagen werden wieder entfernt [1].

Einsteigen der Passagiere (Boarding):

Die Passagiere werden über die Fluggastbrücke oder mit Hilfe von den Vorfeldbussen zum Flugzeug transportiert. Dort steigen sie ein, werden von der Kabinenbesatzung begrüßt, verstauen ihr Handgepäck und nehmen ihre Sitzposition ein. Die Flugzeugtür wird geschlossen und der Einsteigevorgang ist damit abgeschlossen [1].

Off-Block:

Es werden die Bremsklötze von den Rädern des Fahrwerks entfernt und eine „Off-Block“ Meldung wird an die Operationszentrale weitergeleitet [1].

Pushback:

Das Pushback-Fahrzeug wird vor dem Luftfahrzeug positioniert und mit dem Flugzeug verbunden. Nach dem Entfernen der Bremsklötze wird das Flugzeug von der Taxilane auf den Taxiway zurückgeschoben. Währenddessen werden für gewöhnlich die Triebwerke angelassen. Der Vorgang endet, sobald sich das Fahrzeug vom Flugzeug entfernt [1].

2.3. Verspätungen

Ein Flug gilt dann als verspätet, wenn ein Flugzeug mit einer Verzögerung von mehr als 15 Minuten startet. Abbildung 2 zeigt den Verlauf der Pünktlichkeit in den Jahren 2002–2021.

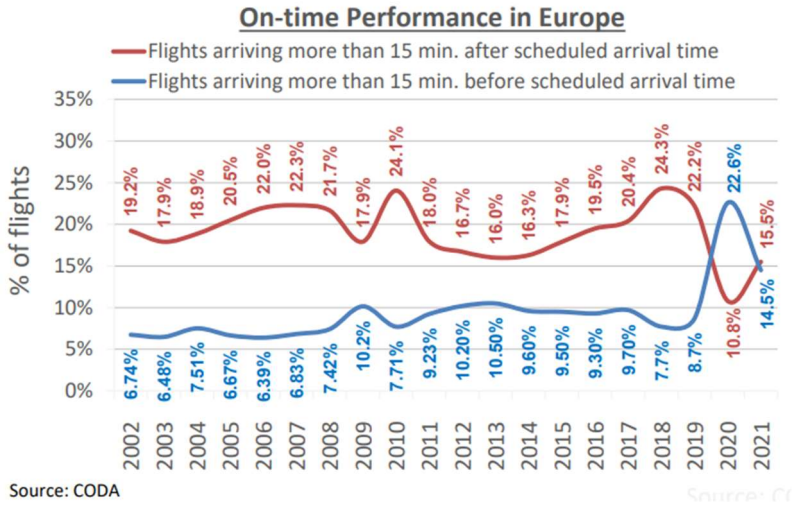

Abbildung 2: Pünktlichkeit in den Jahren 2002 bis 2021 [3].

Hier ist zu erkennen, dass ein nicht unerheblicher Teil von 16%–24% der Flüge verspätet sind, wenn man die Auswirkungen der COVID-19 Pandemie nicht berücksichtigt. Dabei wird zwischen *primären* und *reaktionären* Verspätungen unterschieden. Primäre Verspätungen treten auf, wenn ein Flugzeug trotz ausreichender Bodenzeit verspätet startet. Reaktionäre Verspätungen entstehen dann, wenn ein Flugzeug mit Verspätung landet und kein ausreichend großer Puffer vorhanden ist, um pünktlich starten zu können. Im Jahr 2019 waren 32,6% aller Verspätungen von primärer und 44,4% reaktionärer Art. Eine Minute primäre Verspätung verursacht dabei 0,80 Minuten reaktionärer Verspätung [3]. Abbildung 3 gibt einen genaueren Überblick darüber, wo genau die Verspätungen entstehen. Zu sehen ist hier ein Boxplot, bei dem verschiedene Verspätungsklassen auf der X-Achse abgebildet sind. Die Breite der Boxen gibt an, wie häufig eine Verspätung auftritt. Auf der Y-Achse ist abgebildet, wie groß die Streuung der jeweiligen Verspätung ist. Die Daten stammen aus den Jahren 2006/2007. Demnach sind die häufigsten Ursachen für eine Verspätung die Flugzeugrotation (*AC rotation*) mit 17,9% und die *ramp congestion* mit 15,2%. Verspätungen durch unerwartete Wetterlagen treten nur selten auf, verursachen jedoch die größten Verzögerungen. Weitere relevante Ursachen für Verspätungen sind Flugzeug-Defekte (*AC defects / maintenance*), Gepäckausladungen (baggage offload) und Verspätungen in der Crew-Rotation (*crew rotation*). [2]. Zu beachten ist hier, dass die Daten nur für Flugzeuge der Gruppe C gültig sind, also Flugzeuge für den Kurz- und Mittelstreckenflug.

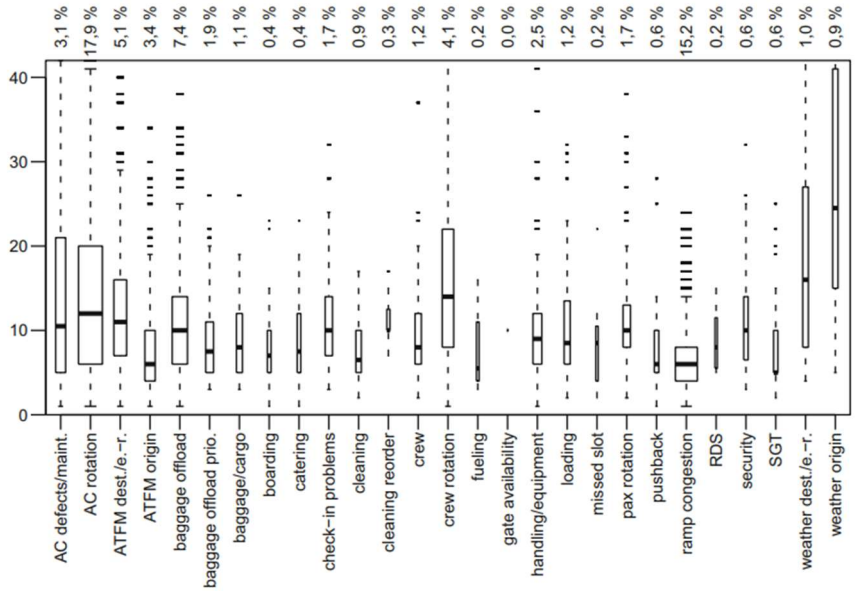

Abbildung 3: Verspätungszeiten aufgeteilt in Verspätungsklassen [2].

2.4. Variationen im Ablauf der Bodenabfertigung

Der Prozess der Bodenabfertigung hängt nicht nur von der Luftfahrtgesellschaft und dem Flughafen, sondern auch von dem Flugzeugmodell ab. In diesem Kapitel werden zunächst die Unterschiede im Ablauf anhand des Airbus A320 und des A330 dargestellt.

Ausrüstung für die Bodenabfertigung

AC	Air-Conditioning Unit	LDCL	Lower Deck Cargo Loader
AS	Air Start Unit	LV	Lavatory Vehicle
BULK	Bulk Train	PBB	Passenger Boarding Bridge
CAT	Catering Truck	PS	Passenger Stairs
CB	Conveyor Belt	TAT	Turnaround Time
CLEAN	Cleaning Truck	TOW	Tow Tractor
FUEL	Fuel Hydrant Dispenser or Truck	ULD	Unit Load Device Train
GPU	Ground Power Unit	WV	Portable Water Vehicle
GSE	Ground Service Equipment		

Tabelle 1: Abkürzungen für die Bodenabfertigung [4].

Im Folgenden werden einige Abkürzungen verwendet. Diese werden in Tabelle 1 erläutert. Zunächst wird der Aufbau bei der Bodenabfertigung zwischen den beiden Flugzeugen verglichen. Abbildung 4 und Abbildung 5 zeigen einen beispielhaften Aufbau auf dem Vorfeld. Da die Beiden Flugzeuge unterschiedliche Dimensionen besitzen, sind die Positionen der Bodenfahrzeuge generell verschieden. Beim A330 werden jedoch zusätzliche

Fahrzeuge benötigt. So werden insgesamt drei *Passenger Bridges,* drei *Catering Trucks* und zwei *Fuel Trucks* platziert, einer für jede Tragfläche [4] [5].

Abbildung 4 und Abbildung 5zeigen den zeitlichen Ablauf der Bodenabfertigung jeweils beim A320 und beim A330. Unterschiede gibt es in der TRT (Turn Round Time) und somit auch im Umfang der zu verrichtenden Arbeiten. Beim A320 beträgt die TRT 44 Minute, beim A330 sind es hingegen 59 Minuten. Die Zeiten unterscheiden sich vor allem dadurch, dass das Laden der Fracht, das *Refueling*, das *Catering* und das *Cleaning* mehr Zeit benötigt [4] [5].

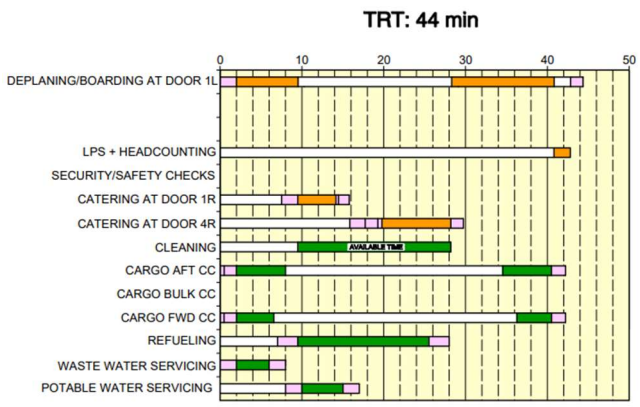

Abbildung 4: Zeitlicher Ablauf der Bodenabfertigung bei einer Airbus A320 [4].

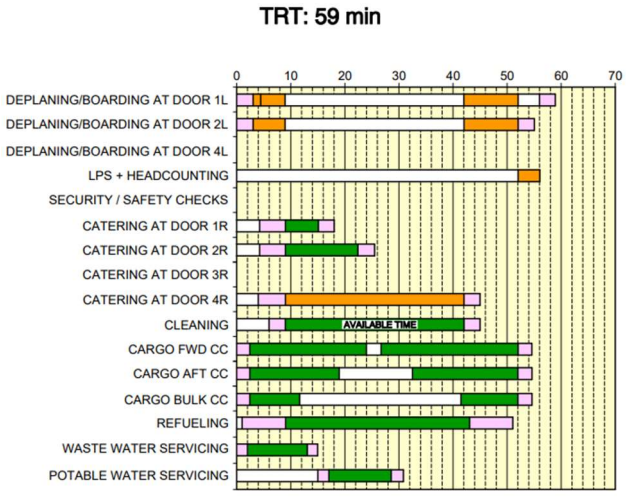

Abbildung 5: Zeitlicher Ablauf der Bodenabfertigung bei einer Airbus A330 [5].

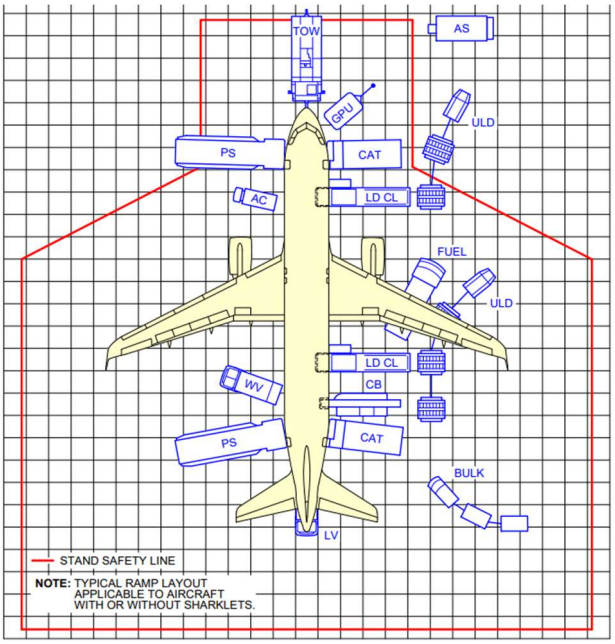

Abbildung 6: Positionierung der Bodenfahrzeuge bei einer Airbus A320 [4].

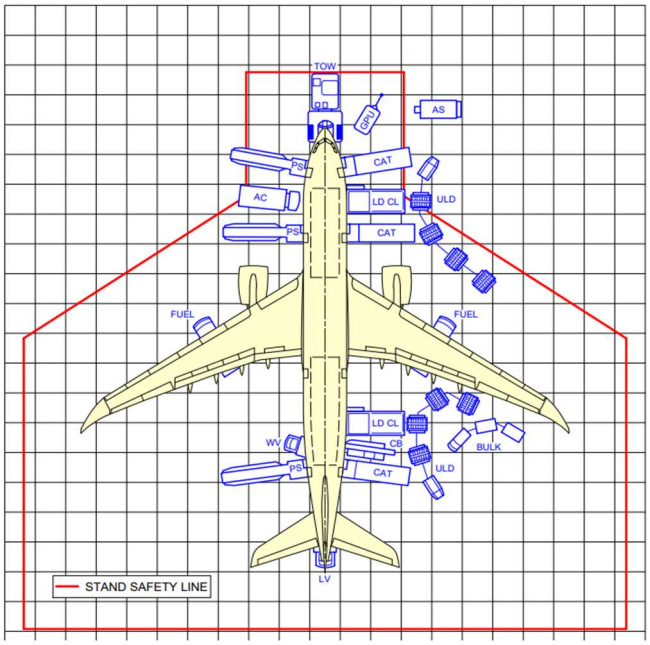

Abbildung 7: Positionierung der Bodenfahrzeuge bei einer Airbus A330 [5].

3. Verkehrssimulation im Allgemeinen

In diesem Kapitel werden verschiedene Verkehrssimulationsmodelle vorgestellt und das Intelligent Driver Model (IDM) näher beschrieben. Außerdem werden agentenbasierte Modelle und Wegfindungs-Algorithmen betrachtet

3.1. Verkehrssimulationsmodelle

Es gibt verschiedene Simulationsmodelle, welche das Verkehrsgeschehen auf unterschiedliche Arten modellieren. Im Folgenden sollen einige Modelle vorgestellt werden. Abbildung 4 gibt einen Überblick über einige verschiedene Modelle.

Makroskopische Modelle nutzen Flüssigkeitsmodelle oder gaskinetische Gleichungen, um den Verkehr zu beschreiben. Betrachtet werden dynamische Größen, wie die mittlere Geschwindigkeit $V(x,t)$, die Verkehrsdichte $\rho(x,t)$, der Fluss $Q(x,t)$ und die Geschwindigkeitsvarianz $\sigma_v^2(x,t)$. Diese werden lokal aggregiert und sind räumlich und zeitlich abhängig. Mit diesen Daten können Staus und die Ausbreitungsgeschwindigkeit von Störungen im Verkehr beschrieben werden [6]. Makroskopische Modelle eignen sich daher besonders gut zur Simulation großer Verkehrsmengen, da sie im Vergleich zu anderen Simulationsansätzen eine geringere Datenmenge benötigen und somit die erforderliche Rechenleistung reduziert wird [6] [7].

Mikroskopische Modelle beschreiben hingegen das Fahrverhalten jedes Verkehrsteilnehmers, wie etwa das

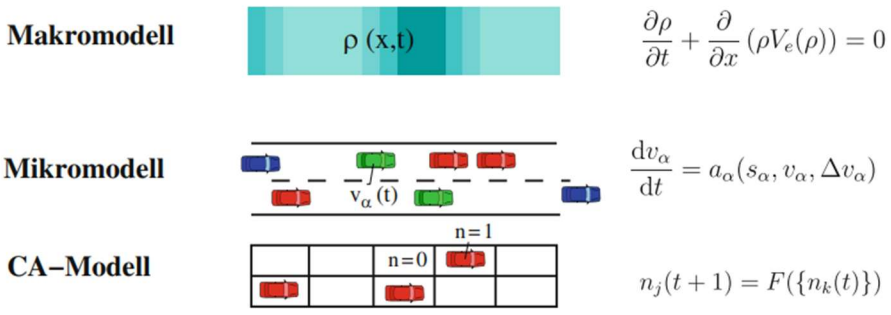

Abbildung 8: Simulationsmodelle mit jeweils einer typischen Modellgleichung [6].

Beschleunigen, Bremsen oder den Spurwechseln in Abhängigkeit von anderen nahen Fahrzeugen. Die von dem Modell betrachteten Größen sind die Position $x_\alpha(t)$, die Geschwindigkeit $v_\alpha(t)$ und die Beschleunigung $\dot{v}_\alpha(t)$. Angewendet wird dieses Modell unter anderem um den Einfluss eines einzelnen Fahrzeuges auf den Straßenverkehr zu untersuchen. Zudem lässt sich mit einem mikroskopischen Modell auch das menschliche Fahrverhalten und die Interaktion zwischen verschiedenen Verkehrsteilnehmern, wie PKWs, LKWs und Bussen nachstellen. Das Ziel ist eine möglichst realistische Darstellung des Verkehrs [7] [6].

Mesoskopische Modelle beinhalten sowohl makroskopische als auch mikroskopische Ansätze. Beispielsweise können makroskopische Größen, wie der Staubeginn oder das Stauende, durch die Änderungsrate der mikroskopischen Größen für die ein- und ausfahrenden Fahrzeuge beschrieben werden. Ein weiteres Beispiel wäre, dass die Parameter des makroskopischen Modells die Parameter des mikroskopischen Modells beeinflussen [6] [7].

3.2. Mikroskopische Simulationen im Detail

Mikroskopische Simulationen bestehen aus einer Vielzahl von Verkehrsteilnehmern, wobei jeder Verkehrsteilnehmer als Akteur oder Agent bezeichnet wird. Es wird zwischen Agenten und intelligenten Agenten

unterschieden. Einen Agenten zeichnet aus, dass er sich in einer Umgebung befindet, in der er bestimmte Aktionen ausführen kann, um sein Ziel zu erreichen. Ein intelligenter Agent interagiert zusätzlich mit anderen Agenten, nimmt die Umgebung wahr, um auf Änderungen reagieren zu können und handelt proaktiv [8] [7]. Die betrachteten Variablen sind die Position, die Geschwindigkeit und die Beschleunigung. Wegen dieser Eigenschaften lassen sich mit der Hilfe von Agenten komplexe Verkehrsszenarien abbilden. Daraus folgt eine hohe Genauigkeit bei der Analyse des Fahrverhaltens und der Reaktionen auf verschiedene Situationen. Im Vergleich zu anderen Verkehrsmodellen sind die Rechenanforderungen zwar höher, mit modernen Computern ist dies jedoch in den meisten Fällen kein Problem mehr [7].

3.2.1. Mikroskopische Simulationsmodelle

Es gibt unterschiedliche Arten von mikroskopischen Modellen. Die häufigsten verwendeten werden hier kurz vorgestellt.

Fahrzeugfolgemodelle gehören zu den beliebtesten mikroskopischen Verkehrsmodellen. Es wird zwischen Modellen der Längsdynamik (Longitudinalmodelle) und Querdynamik (Spurwechselmodelle) unterschieden. Ersteres modelliert Beschleunigungs- und Bremsvorgänge, letzteres den Spurwechsel. Um ein möglichst realitätsnahes Modell zu erhalten, werden beide Modelle kombiniert und subtile Wechselwirkungen zwischen der Beschleunigung und dem Spurwechsel integriert. Die Kombination beider Modelle ist oft der Hauptbestandteil kommerzieller Software. Fahrzeugfolgemodelle modellieren immer das Fahrverhalten der simulierten Fahrzeuge im Verhältnis zum vorausfahrenden Fahrzeug. Diese handeln also nicht autonom. Dabei ist der Abstand zum vorderen Fahrzeug im Verhältnis zur aktuellen Geschwindigkeit ausschlaggebend für das aktuelle Fahrverhalten. Es gibt sowohl zeitkontinuierliche als auch zeitdiskrete Modelle, bei welcher der Zeitschritt vorgegeben ist und meistens zwischen $0{,}1s$ und $1s$ liegt. Eine Zeitdiskrete Variante ist weniger rechenaufwändig, da die Bewegungsdaten seltener berechnet werden müssen. Eins der ersten mathematischen Modelle kommt von Reuschel und Pipes. Ein weiteres beliebtes Modell ist das *Intelligent Driver Model* [7] [6].

In einem *Zellulären Automaten* (engl. *cellular automata*, CA) liegen alle Werte diskretisiert vor, was heißt, dass der Ort in feste Zellen und die Zeit in feste Zeitschritte unterteilt ist (s. Abbildung 4). Jede Zelle führt dabei ein Programm aus, welches mit den Nachbarzellen kommuniziert und in jedem Zeitschritt wird dieses Programm erneut ausgeführt. Das bekannteste Modell ist das Nagel-Schreckenberg-Modell [6] [7].

Agentenbasierte Modelle werden benutzt, um autonom fahrende Fahrzeuge zu simulieren, welche in einem gemeinsam benutzten Bereich agieren und auf nahe Agenten und Umgebungsbedingungen reagieren. Jeder Agent kann eigene Ziele und Strategien verfolgen und besitzt unterschiedliche Verhaltensweisen im Straßenverkehr. Bekannte Modelle sind z. B. *Matsim* oder Sumo [7] [6].

3.2.2. Mathematische Beschreibung von Fahrzeugfolgemodellen

Es soll nun das Fahrzeugfolgemodell mathematisch beschrieben werden. Zunächst wird das Thema durch ein Minimalbeispiel eingeführt, danach erfolgt eine Vorstellung des IDM.

Formel (1) beschreibt den Abstand s_α zwischen zwei Fahrzeugen. Der Index $\alpha - 1$ bezeichnet hierbei das vorausfahrende Fahrzeug. Alternativ wird stattdessen der Index l verwendet. x ist die Position und l die Länge des Fahrzeugs [6].

$$s_\alpha = x_{\alpha-1} - l_{\alpha-1} - x_\alpha = x_l - l_l - x_\alpha \tag{1}$$

Die Fahrerreaktion wird häufig in Abhängigkeit vom Abstand s_α,der eigenen Geschwindigkeit v_α und der Geschwindigkeit des vorrausfahrenden Fahrzeuges v_l modelliert. Bei einem Zeitkontinuierlichen Modell wird die

Reaktion durch eine Beschleunigungsfunktion a_{mic}modelliert, womit man folgende gewöhnliche Differentialgleichungen erhält [6]:

$$\dot{x}_\alpha(t) = \frac{\mathrm{d}x_\alpha(t)}{\mathrm{d}t} = v_\alpha(t) \tag{2}$$

$$\dot{v}_\alpha(t) = \frac{dv_\alpha(t)}{dt} = a_{mic}(s_\alpha, v_\alpha, v_l) = \tilde{a}_{mic}(s_\alpha, v_\alpha, \Delta v_\alpha) \tag{3}$$

Sollen die Abstände mit der Geschwindigkeit gekoppelt werden, lassen sich die Gleichungen (1) und (2) kombinieren [6]:

$$\dot{s}_\alpha(t) = \frac{ds_\alpha(t)}{dt} = v_l(t) - v_a(t) = -\Delta v_\alpha(t) \tag{4}$$

Bei einer Zeitdiskretisierung mit festen Zeitschritten Δt ergeben sich folgende Gleichungen [6]:

$$v_\alpha(t + \Delta t) = v_{mic}(s_\alpha(t), v_\alpha(t), v_l(t)) \tag{5}$$

$$x_\alpha(t + \Delta t) = x_\alpha(t) + \frac{v_\alpha(t) + v_\alpha(t + \Delta t)}{2}\,\Delta t \tag{6}$$

Somit werden die Fahrer-Reaktionen hier nicht durch eine Beschleunigungsfunktion a_{mic}, sondern durch eine Geschwindigkeitsfunktion v_{mic} modelliert, welche die gewünschte Geschwindigkeit für den nächsten Zeitschritt beschreibt [6]. Dieses Modell ist nicht realistisch oder gar vollständig, da zu einem viel zu hohe Beschleunigungswerte auftreten, zum anderen nicht alle Verkehrssituationen, wie die freie Fahrt, oder Annäherungen an langsame oder stehende Fahrzeuge, dargestellt werden können.

Das IDM ist eins der einfachsten zeitkontinuierlichen Modelle, welches in allen Verkehrssituationen realistische Beschleunigungswerte modellieren kann. Dafür muss dieses einige Anforderungen erfüllen:

- Die Beschleunigung muss den Plausibilitätsbedingungen folgen. Die erste Bedingung besagt, dass das Fahrzeug auf die Wunschgeschwindigkeit v_0 beschleunigen muss, falls es keine Behinderung durch andere Fahrzeuge oder Hindernisse gibt (7). Die zweite Bedingung besagt, dass die Beschleunigung sinken soll, je geringer der Abstand im gebundenen Verkehr zum vorausfahrenden Fahrzeug ist (8). Als dritte Bedingung wird vorausgesetzt, dass bei einer dynamischen Annäherung an z. B. ein langsameres Fahrzeug die Verzögerung umso stärker ist, je höher die Annäherungsrate an das vorausfahrende Fahrzeug ist (9). Die vierte Bedingung besagt, dass das Fahrzeug einen Mindestabstand s_0 zum vorausfahrenden Fahrzeug einhalten soll (10) [6].

$$\frac{\partial a_{mic}(s, v, v_l)}{\partial v} < 0, \lim_{s\to\infty} a_{mic}(s, v_0, v_l) = 0 \text{ für alle } v_l \tag{7}$$

$$\frac{\partial a_{mic}(s, v, v_l)}{\partial s} \geq 0, \lim_{s\to\infty} \frac{\partial a_{mic}(s, v, v_l)}{\partial s} = 0 \text{ für alle } v_l \tag{8}$$

$$\frac{\partial \tilde{a}_{mic}(s,v,\Delta v)}{\partial \Delta v} \leq 0 \text{ bzw. } \frac{\partial a_{mic}(s,v,v_l)}{\partial \mathrm{v_l}} \geq 0, \lim_{\mathrm{s}\to\infty} \frac{\partial a_{mic}(s,v,v_l)}{\partial v_l} = 0 \tag{9}$$

$$a_{mic}(s,0,v_l) = 0 \text{ für alle } v_l \geq 0, s \leq s_0 \tag{10}$$

- Der Abstand bei einer Folgefahrt sollt durch einen Mindestabstand s_0 und einen Sicherheitsabstand vT definiert sein und somit $s_0 + vT$ nicht unterschreiten [6].

- Bei der Annäherung an ein stehendes Fahrzeug oder Hindernis soll eine kollisionsvermeidende intelligente Bremsstrategie angewandt werden. Bei einer kontrollierten Bremssituation soll eine komfortable Verzögerung aufgebaut und kurz vor dem Stillstand wieder abgebaut werden. Bei einer Notfallbremsung soll stärker gebremst werden und, falls die Situation wieder unter Kontrolle liegt, mit einer komfortablen Verzögerung fortgefahren werden [6].

- Bei der Folgefahrt und beim Annähern an ein stehendes Fahrzeug sollten die Übergänge sanft sein. Der Ruck J (engl. *Jerk*) bei der Beschleunigung ist die Zeitableitung eben dieser und sollte endlich bleiben. Dazu muss die Beschleunigungsfunktion $a_{mic}(s,v,v_l)$ nach allen Variablen stetig differenzierbar sein [6].

- Das Modell sollte auf anschauliche Modellparameter setzen, wobei jeder Parameter einen Aspekt des Fahrverhaltens beschreibt [6].

Die eben genannten Forderungen werden durch folgende Beschleunigungsformel erfüllt [6] [8]:

$$\dot{v} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right] \tag{11}$$

Diese Formulierung beschreibt den Beschleunigungsvorgang, indem die Geschwindigkeit v mit der Wunschgeschwindigkeit v_0 und der aktuelle Abstand s mit dem Wunschabstand s^* verglichen wird. Der Wunschabstand ist wie folgt definiert [6] [8]:

$$s^*(v,\Delta v) = s_0 + \max\left(0, vT + \frac{v\Delta v}{2\sqrt{ab}}\right) \tag{12}$$

Die Gleichung enthält nicht nur den Mindestabstand und den Sicherheitsabstand, sondern auch einen dynamischen Anteil, der die „intelligente“ Bremsstrategie umsetzt.

Die Parameter lassen sich anhand dreier Fahrsituationen veranschaulichen [6] [8]:

1. Bei freier Fahrt wird zunächst mit der Maximalbeschleunigung a beschleunigt. Die Beschleunigung wird bei der Annäherung an die Wunschgeschwindigkeit v_0 reduziert und geht für $v = v_0$ gegen 0. Je größer das δ ist, desto später wird die Beschleunigung verringert.

2. Bei der Folgefahrt wird der Abstand durch die Folgezeit T und dem Minimalabstand s_0 charakterisiert.

3. Bei der Annäherung an ein langsameres oder stehendes Fahrzeug wird die komfortable Verzögerung b für gewöhnlich nicht überschritten.

Zwischen diesen drei Fahrsituationen gibt es dabei nahtlose Übergänge.

Modellparameter des IDM

Parameter	Typ. Wert Autobahn	Typ. Wert Stadtverkehr
Wunschgeschwindigkeit v_0	120 km/h	54 km/h
Folgezeit T	1.0 s	1.0 s
Minimalabstand s_0	2 m	2 m
Beschleunigungsexponent δ	4	4
Beschleunigung a	1.0 m/s^2	1.0 m/s^2
Verzögerung b	1.5 m/s^2	1.5 m/s^2

Tabelle 2: Modellparameter des IDM für unterschiedliche Umgebungen [6].

Mit der Hilfe des IDM lässt sich das Fahrverhalten in verschiedenen Umgebungen simulieren, ohne dass eine vollständig neue Gleichung benötigt wird. Tabelle 2 schlägt einige Werte für das Fahrverhalten im Stadtverkehr und auf der Autobahn vor. Um den Umgebungswechsel zu modellieren, ist es hier ausreichend die Wunschgeschwindigkeit zu ändern.

Das IDM lässt sich zudem so anpassen, dass auch menschliche Aspekte, wie Schätzfehler oder Reaktionszeiten, simuliert werden können [6].

3.2.3. Agentenbasierte Simulation

Wie bereits erwähnt, wird in einem agentenbasierten Modell jeder Verkehrsteilnehmer als Agent bezeichnet und verfolgt unabhängig von anderen Einflüssen seine eigene Strategie um ein bestimmtes Ziel zu erreichen. Das Ziel des Modells ist es, eine möglichst realistische, detaillierte und dynamische Verkehrssimulation zu realisieren.

Ein Agentenbasiertes Modell besteht dabei aus drei Komponenten. Die erste ist der Agent selbst. Dieser kann eine Vielzahl von Formen annehmen, wie etwa ein Fahrzeug, ein Fußgänger, ein Radfahrer, öffentliche Verkehrsmittel und noch viele mehr. Jeder Einzelne verfolgt dabei seine eigenen Ziele und Strategien. Die Umgebung ist eine weitere Komponente. Diese beinhaltet die Infrastruktur, wie etwa Straßen, Verkehrszeichen, Parkplätze und Ampeln. Die letzte Komponente sind die Interaktionen. Die Agenten reagieren auf ihre Umgebung und andere Agenten. Z. B. kann ein Agent auf ein Stoppschild oder auf ein vorausfahrendes Fahrzeug reagieren.

Ein Vorteil der Agentenbasierten Simulation ist, dass sich mit dessen Hilfe komplexe, dynamische, nichtlineare Verkehrsverhalten darstellen lassen. Eine solche Simulation kann dazu beitragen die Auswirkungen von Änderungen an Verkehrsstrategien, Infrastrukturänderungen und anderen Faktoren auf das Fahrverhalten der Agenten zu untersuchen. Im folgenden Abschnitt werden einige verfügbare Modelle vorgestellt.

Ein Beispiel für ein agentenbasiertes Modell ist Matsim. In diesem Modell hat jeder Agent seinen eigenen Individuellen Tagesplan und trifft Entscheidungen auf der Grundlage seines persönlichen Ziels und der aktuellen Verkehrslage. Es gibt jedoch einige Einschränkungen. So können Überholvorgänge oder der Spurwechsel nicht modelliert werden. Auch können Aktivitäten wie das Radfahren oder Laufen nicht simuliert werden. Da Matsim kostenlos und open-source ist, kann es beliebig auf andere Verkehrsszenarien angepasst werden.

Ein weiteres Beispiel ist Vissim. Dabei handelt es sich um ein kommerzielles Tool zur Darstellung einer detaillierten, multimodalen Verkehrssimulation. Das Tool kann somit verschiedene Verkehrsteilnehmer, wie etwa Autos, Busse, Straßenbahnen, Züge, Fahrräder und Fußgänger simulieren. Es handelt sich dabei um ein Zeitdiskretes Modell mit einem Zeitschritt von $0{,}1\ s$. Es ist möglich Straßen und Verkehrsinfrastrukturen in einem hohen Detailgrad darzustellen, was jedoch einen hohen Aufwand zur Erstellung der Szenarios bedeutet. Vissim besitzt ebenfalls ein Emissionsmodell und kann auch zur Ampelschaltungsoptimierung eingesetzt werden. Die Software wurde für unterschiedliche Fahrweisen in Deutschland und den USA eingestellt und wird zur Untersuchung verschiedener Aspekte des Verkehrssystems verwendet.

Sumo ist eine weitere quelloffene Software zur Verkehrssimulation und wird von mehreren Mitarbeitern des DLRs entwickelt. Ziel ist es eine Vereinheitlichung von Simulationsergebnissen und Modellen im Verkehrssimulationsbereich zu schaffen. Das Tool ist ebenfalls Zeitdiskret und arbeitet mit einem Zeitschritt von einer Sekunde. Dazu wird das Fahrzeugfolgemodell *Extended Intelligent Driver Model* benutzt, welches eine erweiterte Version des IDM darstellt [9]. Für das Pathfinding können unterschiedliche Algorithmen verwendet werden, wie z. B. *Dijkstra*, *A**, *ALT* oder *CH* (Contraction Hierarchies) [10]. Es wird ebenfalls multimodaler Verkehr unterstützt, der tatsächliche Weg von der Quelle zum Ziel wird jedoch nur für Autos simuliert. Bei anderen Verkehrsteilnehmertypen wird die Reisedauer abgeschätzt. Somit können nicht die Interaktionen zwischen allen Verkehrsteilnehmern berücksichtigt werden

3.3. Pathfinding

Als Wegfindung (engl. pathfinding) wird die Suche nach der kürzesten Route zwischen zwei Punkten bezeichnet. Es gibt viele Anwendungsbereiche, in denen eine Lösung für dieses Problem nötig ist, wie z. B. bei der Verkehrsplanung, der Routenplanung im Telefonverkehr, die Navigierung durch ein Labyrinth, der Roboterbahnplanung oder Videospiele [11].

Es sollen zunächst zwei Algorithmen verglichen werden: Greedy Best-First-Search und Dijkstras Algorithmus. Dijkstras beginnt an einem festgelegten Startpunkt und setzt die Kosten, um jeden Punkt zu erreichen, auf unendlich. Der Algorithmus betrachtet dann jeden der Nachbarknoten und aktualisiert die Kosten für den Weg vom Startpunkt zu diesem Knoten. Der Knoten mit den niedrigsten Kosten, der noch nicht besucht wurde, wird dann ausgewählt und dessen Nachbarknoten werden auf die gleiche Weise betrachtet. Der Prozess wird nun solange wiederholt, bis der kürzeste Weg zum Ziel gefunden wurde [12].

Der Greedy Best-First-Search Algorithmus funktioniert ähnlich, nur dass dort noch eine Heuristik zum Einsatz kommt, welche die Entfernung vom untersuchten Knoten zum Ziel schätzt. Anstatt den zum Startpunkt nächstgelegenen Punkt zu untersuchen, wird nun der zum Zielpunkt nächstgelegene Punkt ausgewählt. Der Greedy Best-First-Search Algorithmus findet jedoch nicht immer den kürzesten Pfad, ist jedoch sehr viel schneller als Dijkstras Algorithmus. Die folgenden Grafiken verdeutlichen den Unterschied zwischen den Beiden Algorithmen [12].

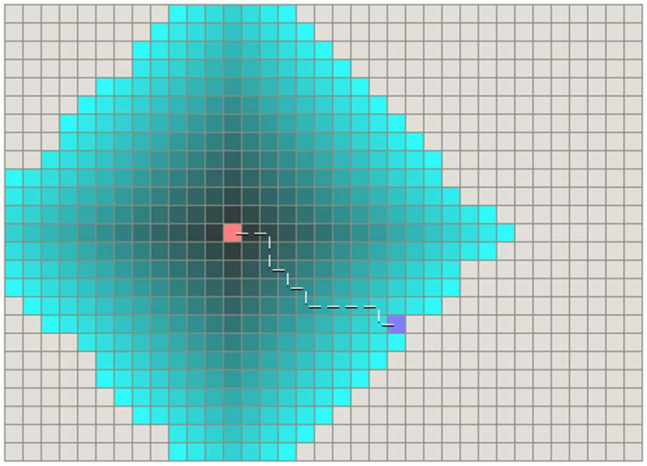

Abbildung 9: Wegfindung mit Dijkstras Algorithmus ohne Hindernis [12].

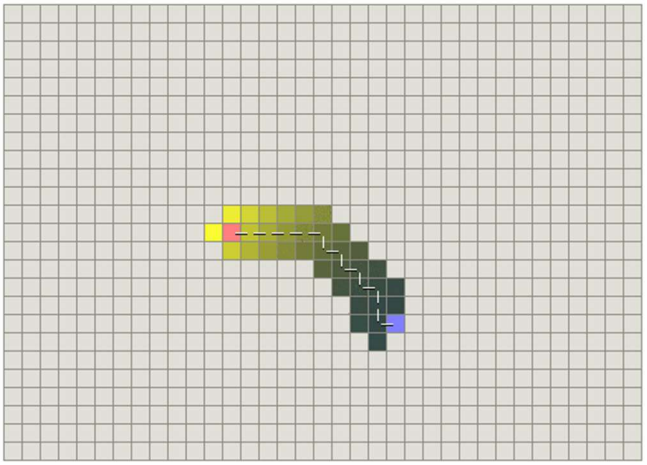

Abbildung 10: Wegfindung mit dem Greedy Best-First-Search Algorithmus ohne Hindernis [12].

Abbildung 9 zeigt die Wegfindung mit der Hilfe des Dijkstra Algorithmus‘. Das pinke Quadrat ist der Startpunkt und das blaue Quadrat der Zielpunkt. Die türkisen Felder sind die, die vom Algorithmus untersucht werden. Abbildung 10 zeigt im Vergleich dazu die Funktionsweise des Greedy Best-First-Search Algorithmus‘. Hier sind die gelben Felder die, die untersucht wurden. Es fällt sofort auf, dass die Menge der untersuchten Felder sehr viel geringer ist. Beide Algorithmen finden hier einen der kürzesten Wege. Die folgenden Beispiele enthalten nun ein Hindernis im Weg [12].

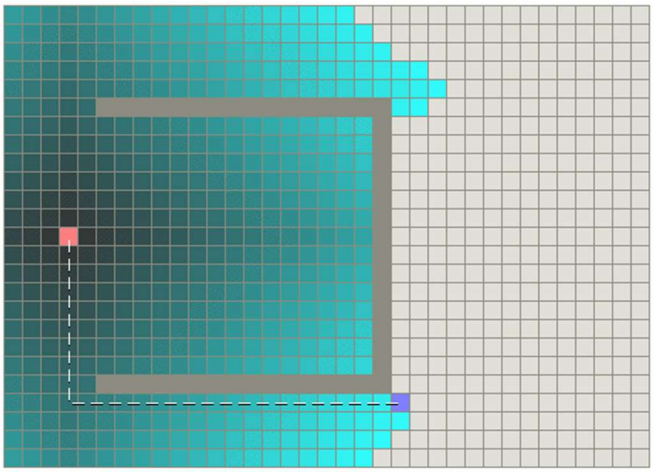

Abbildung 11: Wegfindung mit Dijkstras Algorithmus mit Hindernis [12].

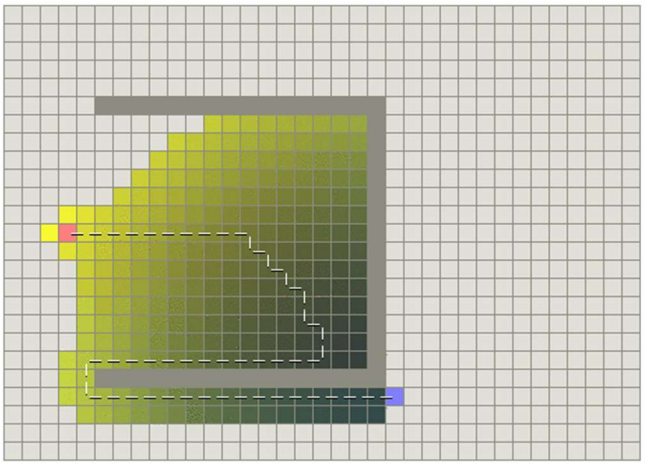

Abbildung 12: Wegfindung mit dem Greedy Best-First-Search Algorithmus mit Hindernis [12].

Wie zu sehen ist, untersucht der Greedy Best-First-Search Algorithmus wiederum weniger Felder, findet jedoch auch nicht den kürzesten Weg, da nicht die zurückliegenden Wegkosten betrachtet werden.

Der A* Algorithmus kombiniert nun beide Ansätze und kann damit immer den kürzesten Weg finden. Die folgenden Bilder verdeutlichen die Vorteile von A* [12].

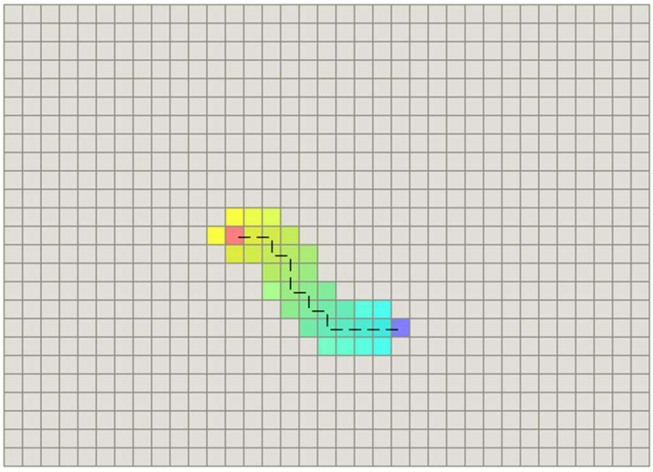

Abbildung 13: Wegfindung mit dem A* Algorithmus ohne Hindernis [12].

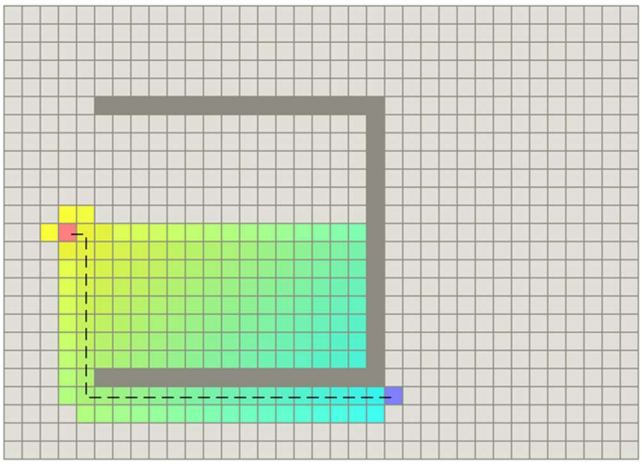

Abbildung 14: Wegfindung mit dem A* Algorithmus mit Hindernis [12].

Ohne Hindernis ist der Algorithmus genauso schnell wie Greedy Best-First-Search. Mit Hindernis ist der Algorithmus sogar schneller und findet den kürzesten Weg. Der Grund dafür ist, dass der A*-Algorithmus einerseits Knoten bevorzugt, die näher am Startpunkt liegen, andererseits aber auch Knoten bevorzugt, die näher am Ziel liegen [12].

4. Verkehrssimulation am Flughafen

In diesem Kapitel wird zunächst eine Anforderungsanalyse auf der Basis der theoretischen Grundlagen durchgeführt. Im Anschluss wird ein modularer Aufbau für das Programm vorgestellt. Schließlich wird die Entscheidung für die spezifische Programmiersprache begründet.

4.1. Anforderungsanalyse

In diesem Abschnitt werden nun auf der Basis der Erkenntnisse aus den Kapiteln 2 und 3 die benötigten Anforderungen formuliert.

Die Anforderungen werden zunächst tabellarisch aufgeführt. Daraufhin werden einige der Anforderungen näher erläutert oder Umsetzungsvorschläge gegeben. Die angegebene Priorität ist nur als Vorschlag anzusehen. Einige der Anforderungen sind Empfehlungen meinerseits. Eine Erläuterung, warum ich diese Anforderungen für wichtig halte, findet sich im Text unter den jeweiligen Tabellen.

4.1.1. Anforderungen an die Fahrzeugbewegungssimulation

Anforderungen an die Fahrzeugbewegungssimulation

Anforderungs-ID	Beschreibung	Priorität
1	Fahrzeugbewegung und -navigation	
1.1	Die Fahrzeuge bewegen sich entlang eines vorher definierten Straßennetzes.	Hoch
1.2	Die Fahrzeuge beachten die Verkehrsregeln.	Hoch
1.3	Die Fahrzeuge vermeiden Kollisionen.	Hoch
1.4	Die Fahrzeuge suchen den schnellsten Weg zum Ziel auf der Basis der gegenwärtigen Verkehrslage.	Hoch
1.5	Die Fahrzeuge sind dazu in der Lage Geschwindigkeitsbegrenzungen einzuhalten.	Mittel
1.6	Die Fahrzeuge können Verkehrsschilder und -signale erkennen und handeln dementsprechend.	Mittel
2	Fahrzeugtypen und -daten	
2.1	Die Simulation unterstützt verschiedene Fahrzeugtypen.	Hoch
2.2	Die Simulation stellt Daten für jeden Fahrzeugtypen bereit.	Hoch
2.3	Der Kraftstoffverbrauch der Fahrzeuge wird simuliert.	Mittel
2.4	Die Kapazität jedes Fahrzeugtyps wird berücksichtigt (z. B. die Ladekapazität der Gepäckwagen).	Mittel
2.5	Jeder Fahrzeugtyp ist dessen realem Vorbild visuell ähnlich.	Niedrig
3	Fahrzeuginteraktionen mit Flugzeugen	
3.1	Der gesamte Turnaround-Prozess, einschließlich Boarding, Entladen und Betanken, wird simuliert.	
3.2	Fahrzeuge warten auf ihrer Parkposition.	Hoch
3.3	Fahrzeuge fahren zur Warteposition, wenn ein Flugzeug im Anflug ist.	Hoch
3.4	Fahrzeuge fahren zur jeweiligen Serviceposition und starten den Service, sobald das Flugzeug steht.	Mittel
3.5	Fahrzeuge müssen den Taxiway vermeiden, wenn Flugzeuge sich dort bewegen.	Hoch
3.6	Die Simulation ermöglicht die Koordination zwischen den Fahrzeugen und der Flugzeugbewegung auf dem Rollfeld.	Hoch

Tabelle 3: Anforderungen an die Fahrzeugbewegungssimulation.

Unter diesem Punkt findet man die Anforderungen an das Bewegungsverhalten der Fahrzeuge. Eine Möglichkeit diese Anforderungen zu erfüllen, ist eine agentenbasierte Verkehrssimulation, welche eine angepasste Version des IDM und zusätzlich den A* Algorithmus verwendet.

Die agentenbasierte Simulation ist für eine derartige Simulation geeignet, da die vielen unterschiedlichen Fahrzeugtypen (Anforderungen 2.1, 2.2, 2.4, 2.5) autonom handeln sollen, um den Service am Flugzeug durchführen zu können. Auch die Anforderungen 1.2 und 1.3 können durch eine agentenbasierte Simulation erfüllt werden, indem das Verhalten der Agenten so modelliert wird, dass diese auf ihre Umgebung reagieren. Die

Anforderungen 1.1, 1.5 und 1.6 sind Punkte, welche durch ein Verkehrsflussmodell erfüllt werden können. Eine erweiterte Version des IDM wäre für diese Aufgabe geeignet. Der A* Suchalgorithmus kann dazu verwendet werden, die schnellste Route auf Basis der gegenwärtigen Verkehrssituation zu finden (Anforderung 1.4). Dabei sollten auch etwaige Sperrungen berücksichtig werden. Der gesamte Turnaround Prozess soll simuliert werden (Anforderung 3.1). Dafür könnten die Interaktionen der Fahrzeuge mit den Flugzeugen durch eine Kombination aus ereignis- und regelbasierten Triggern simuliert werden. Fahrzeuge können so angewiesen werden auf ihrer Parkposition zu warten (Anforderung 3.2), bis ein Flugzeug im Landeanflug ist, woraufhin sich der Agent zur Warteposition begibt (Anforderung 3.3). Sobald das Flugzeug in Position ist, kann der Service starten (Anforderung 3.4). Zudem sollte ein Regelwerk integriert werden, welches verhindert, dass Fahrzeuge die Start- und Landebahn betreten, sobald ein Flugzeug startet oder landet. Durch die Verknüpfung der autonomen Fahrzeugagenten und der Flugzeugbewegung (Anforderung 3.6), kann so ein effizientes und realistisches System erstellt werden, welches die Bewegung auf dem Rollfeld simuliert.

In Kapitel 3.2.2 wurde bereits die mathematische Beschreibung des IDM erwähnt. An dieser Stelle soll eine mögliche Umsetzung vorgeschlagen werden. Um den Rechenaufwand einzuschränken, wird eine Zeitdiskrete Variante der Bewegungssimulation empfohlen. Die Größe des Zeitschritts liegt dabei für das IDM oft zwischen $0{,}1s$ und $0{,}2s$. Die zu lösenden Differentialgleichungen sind dieselben wie im Minimalbeispiel in Kapitel 3.2.2:

$$\frac{\mathrm{d}v_\alpha}{\mathrm{d}t} = f(s_\alpha, v_\alpha, \Delta v_\alpha) \tag{13}$$

$$\frac{\mathrm{d}x_\alpha}{\mathrm{d}t} = v_\alpha \tag{14}$$

Durch numerische Integration erhält man folgende Lösung [8]:

$$v_\alpha(t + \Delta t) = v_\alpha(t) + \dot{v}_\alpha(t)\Delta t \tag{15}$$

$$x_\alpha(t + \Delta t) = x_\alpha(t) + v_\alpha(t)\Delta t + \frac{1}{2}\dot{v}_\alpha(t)(\Delta t)^2 \tag{16}$$

Mit den Randbedingungen $x_\alpha(0) = 0$ und $v_\alpha(0) = 0$ und für den Fall, dass es kein vorausfahrendes Fahrzeug gibt, lässt sich damit der erste Zeitschritt bei $t = 0{,}1s$ wie folgt berechnen:

$$v_\alpha(0{,}1s) = a \cdot 0{,}1s \tag{17}$$

$$x_\alpha(0{,}1s) = \frac{1}{2}a \cdot (0{,}1s)^2 \tag{18}$$

4.1.2. Anforderungen an die Flugzeugbewegungssimulation

Anforderungen an die Flugzeugbewegungssimulation

Anforderungs-ID	Beschreibung	Priorität
4	Flugzeugbewegung	
4.1	Landungen werden gemäß Flugplan durchgeführt.	Hoch
4.2	Flugzeuge können landen und über den Taxiway zur Parkposition gelangen	Hoch
4.3	Die Simulation kann Wartezeiten bei mehreren Flugzeugen optimieren und den optimalen Pfad ohne Kollision finden.	niedrig
4.4	Flugzeuge müssen starten können	Hoch
4.5	Es werden die Auswirkungen von Wetterbedingungen auf die Flugzeugbewegungen berücksichtig.	Mittel
4.6	Flugzeuge müssen mit dem Pushback-Fahrzeug interagieren können, so dass der Pushback vom Fahrzeug durchgeführt werden kann.	Mittel
5	Flugzeugattribute	
5.1	Die Simulation unterstützt verschiedene Flugzeugmodelle.	Mittel
5.2	Die Simulation unterstützt verschiedene Airlines und deren Richtlinien.	Niedrig

Tabelle 4: Anforderungen an die Flugzeugbewegungssimulation.

Auch die Bewegung der Flugzeuge muss simuliert werden, da es direkte Abhängigkeiten zwischen den Fahrzeugen und den Flugzeugen gibt. Die Fahrzeuge sollen auf die ankommenden Flugzeuge reagieren und müssen z. B. auch beachten, keine Kollision mit den Flugzeugen zu verursachen. Die Flugzeugbewegungssimulation kann ebenfalls in Teilen durch eine agentenbasierte Simulation unter Einbeziehung spezifischer Algorithmen für die Flugzeugbewegung und die Abläufe beim Turnaround umgesetzt werden.

Anforderung 4.2 kann durch die Anwendung eines angepassten A* Algorithmus erfüllt werden. Für die Umsetzung von 4.3 ist ein zusätzlicher Algorithmus zur Kollisionsvermeidung und Optimierung der Pfade notwendig. Anforderung 4.1 könnte durch die Einbindung einer Schnittstelle mit Live-Positionsdaten oder durch einen vorher festgelegten Flugplan umgesetzt werden. Nach der Landung sollte der eben erwähnte Algorithmus übernehmen, um eine Interaktion mit dem System zu ermöglichen. Eine eventbasierte Steuerung kann dazu verwendet werden, den Start einzuleiten, sobald der Service abgeschlossen ist (Anforderung 4.4).

Um die unterschiedlichen Flugzeugmodelle, Airlines und deren Richtlinien zu unterstützen (Anforderungen 5.1, 5.2), können die individuellen Flugzeugmodelle als Agenten in der Simulation implementiert und diesen verschiedene Eigenschaften zugewiesen werden.

4.1.3. Anforderungen an das Flughafenlayout

Anforderungen an das Flughafenlayout

Anforderungs-ID	Beschreibung	Priorität
6	Import und Generierung des Flughafenlayouts	
6.1	Das Flughafenlayout wird aus einer Datei oder geographischen Koordinaten importiert.	Hoch
6.2	Das Straßennetzwerk besteht aus Knoten und Kanten mit verschiedenen Gewichtungen, abhängig vom derzeitigen Verkehr und der Länge der Straßen.	Hoch
7	Flughafenlayout und -simulation	
7.1	Es werden mehrere Runways unterstützt	Niedrig
7.2	Es werden Passagierbrücken oder/und Busse simuliert	Niedrig
7.3	Airlines können spezifischen Gates oder Terminals zugeordnet werden	Niedrig
7.4	Wichtige Flughafeneinrichtungen, wie z. B. Wartungseinrichtungen, Tankstellen werden berücksichtigt	Niedrig
7.5	Störungen und Verzögerungen im Turnaround-Prozess werden simuliert.	Niedrig

Tabelle 5: Anforderungen an das Flughafenlayout.

Die Erstellung eines detaillierten und funktionalen Flughafenlayouts ist eine Grundvoraussetzung für eine realistische Simulation der Bodenabfertigung. Hierfür werden verschiedene Aspekte, wie das importieren eines Straßenlayouts oder die Unterstützung verschiedener Einrichtung und Strukturen berücksichtigt.

Um eine möglichst realitätsnahe Darstellung zu erzeugen, soll das Flughafenlayout aus geographischen Koordinaten importiert werden (Anforderung 6.1). Idealerweise werden diese Daten von den Flughäfen bereitgestellt. Hier könnten auch weitere Daten hinterlegt sein, wie z. B. der Standort der Gates, anderer Flughafengebäude oder Tankstellen. Das Straßennetzwerk sollte dabei als Graph mit Knoten und Kanten modelliert werden (Anforderung 6.2), wobei jede Kante eine Straße und jeder Knoten eine Kreuzung repräsentiert. Die Gewichtung kann je nach Länge der Straße oder je nach Verkehrsaufkommen oder wegen anderer Faktoren variieren. Dies ermöglicht eine dynamische Routenplanung, welche aktuelle Verkehrsbedingungen berücksichtigt.

Zudem müssen Teile des Flughafenmanagement und verschiedene Einrichtungen simuliert werden. Es sollte z. B. möglich sein, mehrere Runways hinzuzufügen (Anforderung 7.1), um verschiedene Flughäfen, Flugzeugmodelle und Betriebsabläufe simulieren zu können. Zudem sollte es eine Unterstützung von Passagierbrücken und Bussen geben (Anforderung 7.2), um die Bewegung von Passagieren zwischen Terminals und Flugzeugen zu simulieren. Außerdem sollte es für eine realistische Darstellung der Flughafenoperationen möglich sein, Airlines spezifischen Terminals oder Gates zuzuweisen (Anforderung 7.3). Dies erfordert ein Managementsystem innerhalb der Simulation, welche Flugpläne, Fluggesellschaftsanforderungen und Terminalkapazitäten berücksichtigt. Darüber hinaus sollen diverse Flughafeneinrichtungen, wie z. B. Wartungseinrichtungen oder Tankstellen in der Simulation berücksichtigt werden (Anforderung 7.4), denn dadurch können Wartungsarbeiten oder der Tankprozess simuliert werden, wodurch nicht nur das Verkehrsaufkommen erhöht wird, sondern auch Fahrzeugausfälle entstehen können. Zudem sollten diverse Störungen, welche in Kapitel 2.3 besprochen wurden, im Turnaround-Prozess simuliert werden (Anforderung 7.5).

4.1.4. Anforderungen an die Systemfunktionalität

Anforderungen an die Systemfunktionalität

Anforderungs-ID	Beschreibung	Priorität
8	User Interface (UI)	
8.1	Das Programm unterstützt das verändern des Bildausschnitts, wie etwa durch „ziehen“ des Bildes oder durch eine Zoomfunktion.	Hoch
8.2	Es existiert eine Möglichkeit zur Darstellung der Verkehrsdichte.	Niedrig
9	User Experience (UX)	
9.1	Die Simulation gibt Rückmeldung über den Status der Simulation (z. B. Fortschritt der Bodenabfertigung, Ankunft des nächsten Flugzeugs)	Niedrig
9.2	Eine Timescale-Funktion ermöglicht das Überbrücken von Wartezeiten.	Niedrig
9.3	Das Programm ist robust gegenüber Fehlern und zeigt angemessene Fehlermeldungen an.	Mittel
9.4	Die Simulationsgeschwindigkeit ist auch für große Flughäfen mit einer hohen Anzahl an Objektbewegungen ausreichend.	Mittel

Tabelle 6: Anforderungen an die Systemfunktionalität.

Im Folgenden werden die Anforderungen hinsichtlich des User Interfaces (UI) und der User Experience (UX) näher erläutert.

Die Implementierung der Änderung des Bildausschnitts (Anforderung 8.1) ist wichtig, um die Simulation im Detail betrachten zu können. Nur so kann beurteilt werden, ob die Simulation sich wunschgemäß verhält oder wo es Probleme gibt. Die Verkehrsdichte (Anforderung 8.2) könnte durch eine Farbcodierung umgesetzt werden, wobei z. B. eine hohe Verkehrsdichte mit einer dunkleren Farbe dargestellt werden könnte. Auf diese Weise könnten Engpässe schneller identifiziert werden.

Die Rückmeldung über den Status der Simulation (Anforderung 9.1) könnte durch ein Ereignis-gesteuertes Benachrichtigungssystem implementiert werden, welches Updates zu wichtigen Ereignissen, wie z. B. den Fortschritt der Bodenabfertigung oder Witterungsänderungen liefert. Eine Timescale-Funktion (Anforderung 9.2) soll es ermöglichen, Wartezeiten zu überbrücken, da die Bodenabfertigung ein längerer Prozess sein kann. Dadurch wird es einfacher, die entscheidenden Momente zu betrachten. Eine Implementierung könnte so aussehen, dass verschiedene Zeitfaktoren ausgewählt werden können. Anforderung 9.3 fordert eine robuste Fehlerbehandlung, die durch ein gut strukturiertes Fehlerbehandlungs- und Berichtssystem umgesetzt werden kann. Die Fehlermeldungen sollten genau beschreiben, um was für einen Fehler es sich handelt und Informationen darüber liefern, welche Komponente, welches Modul oder welches System betroffen ist. Zudem sollten die Meldungen verständlich sein, so dass auch Nicht-Entwickler diese nachvollziehen können. Um eine hohe Simulationsgeschwindigkeit auch bei großen Flughäfen mit einer hohen Anzahl von Objekten zu ermöglichen (Anforderung 9.4), sind effiziente Datenstrukturen und Algorithmen notwendig. Insbesondere bei der Pfadfindung und der Kollisionserkennung, sollte dies berücksichtigt werden.

4.2. Modularer Aufbau der Simulation

Für die Simulation ist ein modularer Aufbau vorgesehen, um die einzelnen Module warten und erweitern zu können, ohne dass der gesamte Programmcode betroffen ist. Außerdem erhöht Modularität die Verständlichkeit des Systems, da jedes Modul eine spezifische Aufgabe erfüllt. Des Weiteren ist es möglich einzelne Module für andere Projekte wiederzuverwenden.

Die folgende Grafik zeigt einen modularen Aufbau, basierend auf den genannten Anforderungen und enthält sieben unterschiedliche Module. Diese Module und deren Interaktionen werden nachfolgend beschrieben.

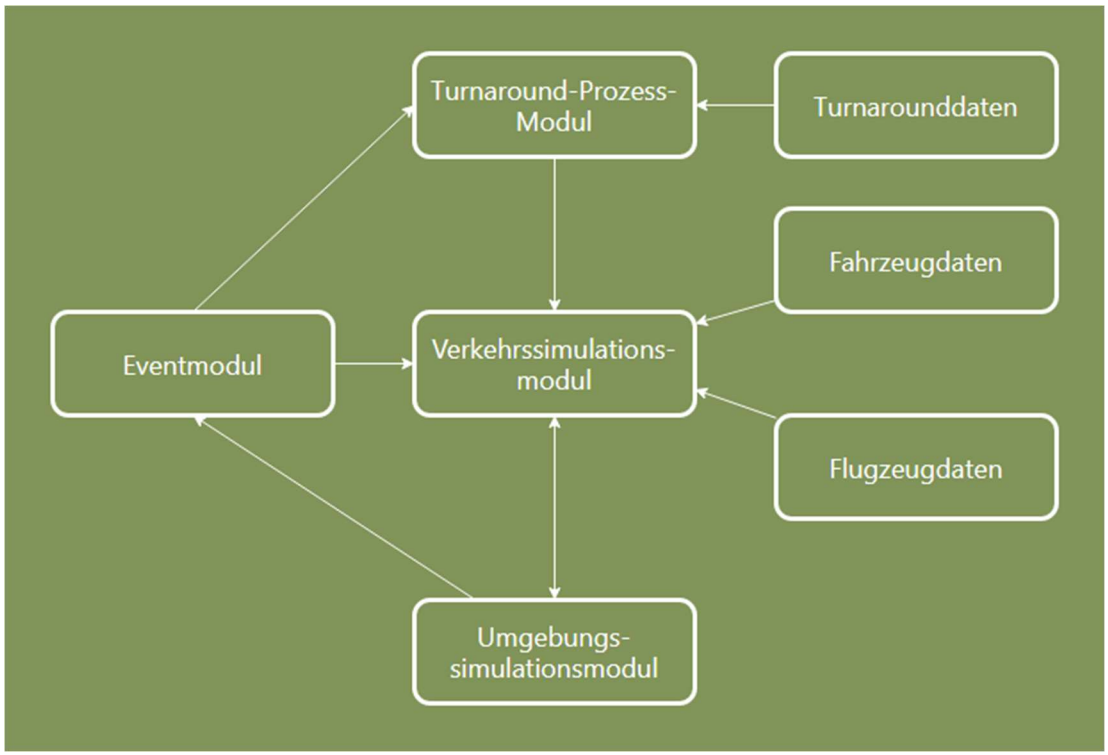

Abbildung 15: Grafische Darstellung des modularen Aufbaus der Simulation.

Fahrzeugdaten

Dieses Modul behandelt die Anforderungen des Abschnitts 2. Es verwaltet damit die verschiedenen Fahrzeugtypen und dessen Attribute, wie z. B. den Kraftstoffverbrauch oder die Ladekapazität.

Flugzeugdaten

Dieses Modul beinhaltet die Anforderungen des Abschnitts 5. Somit enthält es sämtliche Attribute der Flugzeuge, wie etwa die unterschiedlichen Flugzeugtypen und sendet diese Daten an das Verkehrssimulationsmodul, damit dieses die Bewegung des Flugzeugs auf dem Vorfeld umsetzen kann.

Turnarounddaten

Dieses Modul enthält alle Daten, welche für den Turnaround notwendig sind, wie etwa die unterschiedlichen Abläufe bei den verschiedenen Flugzeugmodellen.

Verkehrssimulationsmodul

Dieses Modul ist für die Anforderungen 1, 3 und 4 zuständig. Es nutzt eine angepasste Version des IDM und den A*-Algorithmus zur Fahr- und Flugzeugbewegungssimulation. Es hat Zugriff auf die Fahrzeug- und Flugzeugdaten und steuert damit die Bewegung der Agenten gemäß den Attributen der Fahr- und Flugzeuge. Die Daten über das Flughafenlayout erhält es vom Umgebungssimulationsmodul und sendet an dieses die Daten, welche zur Visualisierung der Fahrzeuge nötig sind.

Turnaround-Prozess-Modul

Dieses Modul ist dafür zuständig, den Ablauf der Bodenabfertigung durchzuführen. Es erhält die Turnarounddaten und kann dementsprechend den Bodenabfertigungsprozess je nach Flugzeugmodell durchführen. Zudem sendet es Signale an das Verkehrssimulationsmodul über den Fortschritt der Bodenabfertigung, so dass das Verkehrssimulationsmodell am Ende des Vorgangs die Kontrolle über die Bodenfahrzeuge übernehmen kann.

Umgebungssimulationsmodul

Dieses Modul beinhaltet vor allem Anforderungen der Kategorien 6 und 7. Es stellt das Flughafenlayout, die Einrichtungen und die Infrastruktur bereit. Es interagiert mit dem Verkehrssimulations- und dem Eventmodul, womit diese Zugriff auf die Flughafendaten haben.

Eventmodul

Dieses Modul ist für alle dynamischen Ereignisse zuständig. Darunter fallen der Flugplan, Wetterveränderungen, Notfälle, Verspätungen im Flugverkehr sowie Verzögerungen bei der Bodenabfertigung. Es sendet Events an das Verkehrssimulationsmodul und das Turnaround-Prozess-Modul, welche daraufhin auf die Ereignisse dementsprechend reagieren können.

4.3. Programmiersprachen und Frameworks

Um eine hohe Leistung der Simulation zu gewährleisten ist die Wahl der richtigen Programmiersprache von großer Bedeutung. Zudem sollte die Programmiersprache die Entwicklung eines modularen Aufbaus unterstützen, um die definierten Anforderungen zu unterstützen.

Um die Auswahl zu ermöglichen, wurden einige beliebte Programmiersprachen und Frameworks ausgewählt. Es sollen die Programmiersprachen Python, C++, C#, Java, C++ mit QT Framework und die Unity Engine verglichen werden. Hauptargumente für die Auswahl sind, dass diese Programmiersprachen Objekt-Orientiertes Programmieren unterstützen und sehr beliebt sind, was eine große Unterstützung und eine gute Dokumentation garantiert [11]. Im Folgenden sollen die in Betracht gezogenen Technologien kurz vorgestellt werden.

Python ist für seine Einfachheit und Lesbarkeit bekannt. Es ist eine umfangreiche Standardbibliothek vorhanden und es gibt zahlreiche Bibliotheken, welche den Funktionsumfang erweitern. Wegen der einfachen Syntax wird es häufig für das rapid prototyping verwendet. Da es eine interpretierte Programmiersprache ist, kann es den Code direkt ausführen, was die Fehleranalyse vereinfacht. Interpretierte Sprachen haben jedoch den Nachteil, dass die Ausführungsgeschwindigkeit niedrig ist, weshalb Python nicht in Anwendungen verwendet wird, in denen die Geschwindigkeit ein entscheidender Faktor ist [12].

C++ ist eine viel verwendete, leistungsstarke Programmiersprache, die für ihre Effizienz und Flexibilität bekannt ist. Bei C++ handelt es sich dabei um eine kompilierte Programmiersprache, was den Vorteil einer hohen

Performance bringt, da der Code in Maschinencode kompiliert wird. Zudem ist in C++ eine hardwarenahe Programmierung möglich, was zusätzlich Ausführungsgeschwindigkeit erhöhen kann. Zudem hat der Programmierer die vollständige Kontrolle über die Speicherverwaltung. Dies ist jedoch auch ein Nachteil, da es keine automatische Speicherbereinigung gibt, sodass der Benutzer selbst darauf achten muss, dass kein Speicherleck entsteht. Aus diesem und anderen Gründen, wie z. B. die Verwendung von Pointern, gilt C++ auch als komplexe Programmiersprache [13].

Java ist eine weit verbreitete, plattformunabhängige Sprache mit einer großen Community und einer großen Anzahl von Bibliotheken. Der Syntax ist einfach zu verstehen und entspricht in etwa einer vereinfachten Version der C++ Syntax. Ein weiterer Vorteil ist, dass Java-Applikationen auf nahezu jeder Plattform lauffähig sind. Zudem gestaltet Java die Speicherverwaltung einfacher, da es eine automatische Speicherbereinigung enthält. Java ist ebenfalls eine kompilierte Programmiersprache, es kompiliert jedoch in Java-Bytecode, welcher wiederum bei der Ausführung von der Java Virtual Machine (JVM) in Maschinencode übersetzt werden muss. Aus diesem Grund muss die Software JVM auf dem auszuführenden Gerät installiert sein und die Performance reicht nicht an die von C++ heran [14].

C# ist ebenfalls weit verbreitet und gilt als einfache und schnell zu erlernende Programmiersprache. Der Syntax ist ähnlich zu Java, unterstützt jedoch fortgeschrittene Programmierkonzepte, wie z. B. *lambda expressions*. Es gibt, wie auch in Java, eine automatische Speicherbereinigung, welche allerdings etwas effizienter arbeitet. Bei C# handelt es sich ebenfalls um eine Programmiersprache, welche in Bytecode kompiliert wird. Damit einher geht auch die im Vergleich zu C++ schlechtere Performance. Insgesamt gilt C# jedoch als performanter als Java [15].

Die Unity Engine basiert auf C# und ist vor allem für die Entwicklung von 3D Spielen konzipiert. Die Engine besitzt eine Vielzahl von Werkzeugen und Funktionen welche die Entwicklung deutlich beschleunigen oder vereinfachen können. Ein weiterer Vorteil ist der umfangreiche Asset Store, welcher die Entwicklungszeit erheblich verkürzen kann. Es ist zudem eine große Community vorhanden, die Unterstützung zur Verfügung stellt. Ein Nachteil der Unity Engine kann jedoch sein, dass diese bei komplexen oder leistungsintensiven Anwendungen Probleme hat. Die Engine ist eher für kleine bis mittlere Projekte vorgesehen und braucht verhältnismäßig viel Speicher. Zudem erreicht diese nicht die Flexibilität, wie die native Verwendung von C# oder C++ [16] [17].

QT ist ein weit verbreitetes Framework für C++, welches insbesondere für plattformübergreifende Anwendungen und grafische Benutzeroberflächen verwendet wird. Es erweitert dafür C++ um zusätzliche Klassen, insbesondere zur Entwicklung des GUIs. Der Syntax orientiert sich an C++, wobei dieser auch um neue Funktionen, wie das Signal-Slot-Konzept, erweitert wird. Zudem gibt es eine umfangreiche Dokumentation und eine große Community. Ein möglicher Nachteil ist, dass QT eine gewisse Einarbeitungszeit benötigt, auch wenn bereits Erfahrungen mit C++ gemacht wurden, da es ein sehr umfangreiches Werkzeug ist. Die Performance ist zudem stark von der Optimierung abhängig [18].

Die Wahl ist letztendlich auf C++ mit QT als Framework gefallen. C++ bietet eine hohe Leistungsfähigkeit, welche für eine rechenaufwändige Simulation notwendig ist, und das QT-Framework ist eine bewährte Plattform zur Erstellung einer grafischen Benutzeroberfläche. Die umfangreiche Dokumentation erleichtert zudem die Einarbeitung in das Framework. Zudem verwenden auch andere Verkehrssimulationen C++ oder C++ mit QT als Framework. Zwei Beispiele hierfür wären die Verkehrssimulationen SUMO [19] oder Aimsun Next [20]. Obwohl die Wahl von C++ in Kombination mit dem QT-Framework viele Vorteile bietet, gibt es auch einige Einschränkungen und Herausforderungen. C++ erreicht zusammen mit QT eine hohe Komplexität und erfordert eine gewisse Einarbeitungszeit. Bei der Programmierung muss zudem auf die Speicherverwaltung geachtet werden, da diese manuell durchgeführt werden muss. Ein weiterer Nachteil sind die Kosten. Zwar gibt es eine kostenlose Version, für diese muss das Projekt jedoch unter einer Open Source Lizenz veröffentlicht werden.

5. Ergebnisse

In diesem Kapitel wird dargestellt, welche Programmierarbeiten an der Simulation bereits durchgeführt werden konnten. Viele der gestellten Anforderungen konnten in diesem ersten Prototyp noch nicht umgesetzt werden, da

der Programmieraufwand für die zur Verfügung stehende Zeit zu groß war. Im Folgenden wird erläutert, welche Anforderungen erfüllt werden konnten und welche Schwierigkeiten auftraten.

Als erstes sollte ein einfaches Benutzerinterface erstellt werden. QT bietet verschiedene Möglichkeiten zur Umsetzung. Das fängt damit an, dass man sich zwischen den Frameworks Qt Widget und Qt Quick entscheiden muss. Eine Entscheidung zwischen den Beiden Frameworks ist wichtig, da diese unterschiedliche Anforderungen erfüllen und unterschiedlich zu bedienen sind. Beide setzen auf ein grundlegend anderes Prinzip bei der Umsetzung des UIs. QT Widget basiert dabei vollständig auf C++, während Qt Quick auf QML setzt, das auf JSON basiert und zusätzlich JavaScript unterstützt. Qt Quick eignet sich besonders für plattformübergreifende Projekte und ist ein neueres Produkt. Auch die grafische Darstellung soll performanter sein, da die Basis moderner programmiert wurde. QT Widget wird vor allem für Desktopanwendungen eingesetzt ist ein seit vielen Jahren etabliertes Framework. Aufgrund der fehlenden Notwendigkeit, die Simulation auf mobilen Geräten zu implementieren, und der simplen Anforderungen an die grafische Darstellung, fiel die Entscheidung auf QT Widget. Die Erstellung der Benutzeroberfläche war schnell erledigt, da der Editor diesen Vorgang stark vereinfacht. Der Editor ermöglicht es, die einzelnen UI-Elemente in das Programmfenster zu ziehen und dort anzuordnen. So wurde ein UI erstellt, das aus einer Seitenleiste für die Bedienelemente und einem freien Bereich für die Darstellung der Simulation besteht.

Abbildung 16: Darstellung des Benutzerinterfaces.

Die freie Fläche ist hier Teil des *Graphics View Frameworks.* Dieses Framework ermöglicht die Darstellung einer Vielzahl von 2D-Objekten. Die dargestellte Szene wird dabei von einer *QGraphicsScene* bereitgestellt, die als Container für die Objekte dient. Das Framework unterstützt auch zusätzliche Funktionen ohne großen Programmieraufwand, wie z. B. das Weiterleiten von Ereignissen an die Objekte oder das Selektieren von Objekten. Problematisch war hier zunächst, wie man vom Programmcode aus auf die einzelnen Elemente zugreift. Dies lag vor allem an der Komplexität der Datenstruktur, da die einzelnen Elemente Objekte sind, die Teil des „UI“-Objekts sind. Mit diesen Informationen und der korrekten Syntax konnte der Zugriff jedoch schließlich erreicht werden.

Im nächsten Schritt wurden die Flughafendaten importiert. Diese werden nicht nur für die visuelle Darstellung, sondern auch für das Pathfinding benötigt. Abbildung 17 zeigt einen Auszug aus der Datei. Die Punkte sind nummeriert und haben jeweils zwei Koordinatenpaare unterschiedlicher Koordinatensysteme. Die Linien haben verschiedene Attribute wie z. B. Straßentyp oder Fahrtrichtung. Zusätzlich sind noch einige Punkte mit Attributen wie „VEHICLE_PARK“ oder „VEHICLE_WAIT“ versehen, die den Park- bzw. Wartebereich für Fahrzeuge kennzeichnen.

```
//Points
//Point_No;'POINT';Easting;Northing;Latitude;Longitude
P3;POINT;3564144.76209;5943372.04247;53.61780824733;9.96819071327327
P4;POINT;3564757.70606;5943372.02725;53.617732961423;9.97745274248377
P5;POINT;3565100.5283;5943372.02753;53.6176905540372;9.9826314525521
P6;POINT;3565963.73383;5943372.02823;53.6175827540932;9.99567449064419
//Lines
//Line_No;Name;SubName;Type;WTC;Weight_Class;Direction(BOTH/FORWARD/REVERSE);Status(OPEN/CLOSED/...);Speed(kts);List of Point_No,,,
L2243;Name;Unternname;STANDLINE_TRIPLE;H;F;BOTH;OPEN;;P752,P801,P802,P803,P804,P805,P806,P807,P808,P809,P810,P811,P800
L2034;Name;Unternname;RWYCENTERLINE;H;F;BOTH;OPEN;100;P46,P57
L3159;Name;Unternname;ROADNETWORK;H;F;BOTH;OPEN;;P1364,P1442
L1834;Name;Unternname;TAXIWAY;H;F;BOTH;OPEN;15;P44,P94
//VehicleParking
//VEHICLE_PARK_No;Name;'VEHICLE_PARK';Status(OPEN/CLOSED/...);;Point_No
K0;Name;VEHICLE_PARK;OPEN;;P1227;
K1;Name;VEHICLE_PARK;OPEN;;P1224;
K2;Name;VEHICLE_PARK;OPEN;;P1221;
//VehicleWaiting
//VEHICLE_WAIT_No;Name;'VEHICLE_WAIT';Status(OPEN/CLOSED/...);;Point_No
W0;Name;VEHICLE_WAIT;OPEN;;P1120;
W1;Name;VEHICLE_WAIT;OPEN;;P1121;
W2;Name;VEHICLE_WAIT;OPEN;;P1122;
```

Abbildung 17: Ausschnitt aus der Datei mit den Flugzeugdaten.

Für den Import der Flughafendaten wurde eine Klasse erstellt. Die Daten wurden zunächst unbearbeitet zeilenweise in 2D-Arrays gespeichert, da das Pathfinding und die visuelle Darstellung des Flughafens unterschiedliche Anforderungen an die Datenstruktur haben. Mit diesem Schritt wurde die Anforderung 6.1 erfüllt. Da es jedoch wahrscheinlich ist, dass sich die Datenstruktur ändert, muss die Importfunktion in Zukunft voraussichtlich angepasst werden.

Ein Button wurde hinzugefügt um diese Daten laden zu können. Dies ist interessant, da hier zum ersten Mal *Signale* und *Slots* verwendet werden, die kein Standardfeature von C++ sind. *Signale* und *Slots* sind ein Mechanismus in Qt, mit dem unterschiedliche Objekte und Klassen miteinander kommunizieren können. Beim Betätigen des Buttons erfolgt der Funktionsaufruf und die Daten werden per Signal an den Slot der *QGraphicsScene* gesendet. Dort wurde eine Klasse für die Punkte und eine Klasse für die Straßen erstellt, in denen die empfangenen Daten verarbeitet werden. An dieser Stelle werden auch die WGS84-Koordinaten ein ein kartesisches Koordinatensystem transformiert, da dieses für die Darstellung in einer zweidimensionalen Umgebung besser geeignet ist. Die Herausforderung dabei war der komplexe Übergang von einem sphärischen in ein ebenes Koordinatensystem, ohne die Genauigkeit der Flughafenkoordinaten zu beeinträchtigen. Deshalb wurde hier eine externe Bibliothek eingesetzt, welche diese Aufgabe erfüllt. Ein weiteres Problem war, dass Qt eigene Dateitypen verwendet, sodass die Dateitypen der C++ Standardbibliothek nicht kompatibel mit den Qt eigenen Funktionen war. Da die Daten nach dem Import im Dateityp der Standardbibliothek vorlagen, mussten diese zunächst konvertiert werden. Qt bietet hier jedoch eine Lösung an, womit die Daten konvertiert werden konnten. In der Klasse für die Straßen wurde nun auch definiert, wie die Straßen dargestellt werden sollen. Hier wurden auch die unterschiedlichen Attribute genutzt. So konnte für die Darstellung der Straßen und der Darstellung der Mittelstreifen ein anderes Design gewählt werden. Für jede einzelne Straße wird nun ein Objekt erzeugt, das dann als *Item* der *QGraphicsScene* hinzugefügt wird. Für Debugging-Zwecke wurde die Möglichkeit genutzt ein Objekt zu markieren und dessen Daten in der Konsole auszugeben. Mit der Qt eigenen Lösung konnte auch hier schnell eine Möglichkeit zur Umsetzung gefunden werden.

Nach dem Laden der Daten ergab sich eine Darstellung des Flughafens, wie sie in Abbildung 18 zu sehen ist. Bei dieser Darstellung war es leider nicht möglich Details des Straßennetzes zu erkennen, weshalb eine Möglichkeit zur Änderung des Bildausschnitts implementiert werden musste. Eine Zoomfunktion zu implementieren war schwieriger als erwartet, hauptsächlich aufgrund der Schwierigkeiten mit der Koordinatenskalierung und der Einbindung des Mausrads. Die Skalierungssyntax war nicht offensichtlich, und die Einbeziehung der Mausradinteraktion war kompliziert, da sie einen tieferen Zugang zur Qt-Ereignisschleife erforderte. Hier wurde auf eine externe Bibliothek zurückgegriffen. Die Implementierung einer Funktion zum Verschieben des Bildausschnitts war hingegen einfach. Dies konnte allein durch das Setzen eines Attributs bei der *Graphics View* im UI-Editor umgesetzt werden. Abbildung 19 zeigt die Darstellung des Flughafens mit der Verwendung der Zoomfunktion. Mit der Implementierung dieser Funktionen wird zudem die Anforderung 8.1 erfüllt.

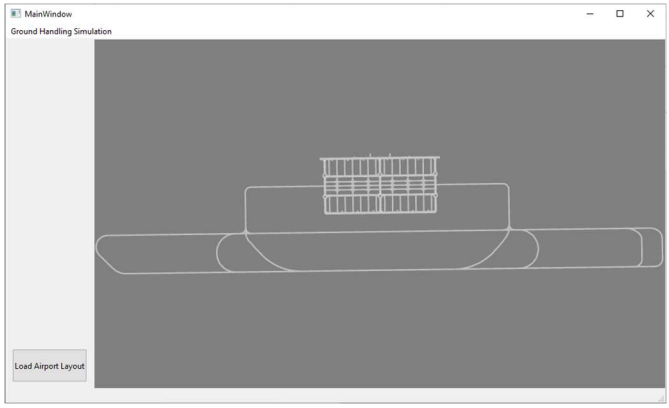

Abbildung 18: Darstellung des Flughafens nach Importieren der Daten.

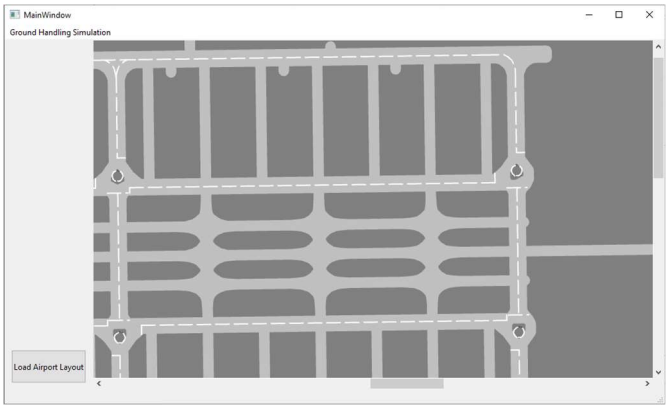

Abbildung 19: Darstellung des Flughafens mit Zoomfunktion.

Als nächstes wurde die Funktion implementiert, ein Fahrzeug von einem Punkt im Straßennetz zu einem beliebigen anderen Punkt zu schicken. Dazu musste zunächst der A* Pathfinding Algorithmus implementiert werden. Die importierten Daten müssen dafür aufbereitet werden. Der A*-Algorithmus benötigt einen gewichteten Graphen, der einem Knoten-Kanten-Modell entspricht. Die Knoten entsprechen dabei den Punkten in den Daten. Verbunden werden diese über die in den Daten definierten Linien. Implementiert wird der Graph als eine *Adjacency List*, welche mit einer $n \cdot 2$ Matrix verglichen werden kann. In der ersten Spalte befindet sich der betrachtete Knoten. In der zweiten Spalte befinden sich die angrenzenden Kanten, bzw. in diesem Fall Objekte der Klasse „*WeightedEdge*“. In diesen Objekten sind verschiedene Daten gespeichert, wie der Endpunkt der Kante, der Kantentyp, bzw. Straßentyp und die Gewichtung der Kante. Die Gewichtung der Kante entspricht hier der Distanz zwischen den beiden Knoten. Der Algorithmus basiert auf einer Implementierung von Amit Patel [12], die jedoch an die Daten angepasst werden musste. Die Umsetzung verlief dabei ohne größere Probleme. Wenn dem Algorithmus nun ein Start- und ein Endpunkt übergeben werden, gibt er eine Liste von Punkten zurück, die den kürzesten Weg zwischen dem Start- und Endpunkt darstellen. Diese können nun genutzt werden, um ein Fahrzeug zu navigieren. Mit der Implementierung des gewichteten Graphen wird zudem die Anforderung 6.2 teilweise erfüllt. Es fehlt noch eine Erweiterung um die aktuelle Straßenlage.

Es wurde eine Klasse „*Vehicles“* erstellt. Diese enthält Daten über die Fahrzeuge, wie die maximale Geschwindigkeit, der Fahrzeugtyp und wie das Fahrzeug dargestellt werden soll. Hier fiel die Wahl zunächst auf ein einfaches Quadrat, da eine andere Form das Problem mit sich brächte, dass das Fahrzeug je nach Fahrtrichtung seine Orientierung ändern müsste. Bei einem Quadrat kann dies vorerst vernachlässigt werden. Zusätzlich wurde eine Klasse „*VehiclePools*“ erstellt. Hier wird das bereits erwähnte Attribut „VEHICLE_PARK“ verwendet. Punkten mit diesem Attribut wird ein Fahrzeugpool zugeordnet. Dabei kann definiert werden, von welchem Fahrzeugtyp wie viele Fahrzeuge vorhanden sind. Derzeit sind hier nur „Standard“-Fahrzeuge eingetragen. Mit der Erstellung der Klasse „Vehicles“ ist zudem die Anforderung 2.1 erfüllt.

Zuletzt musste die Bewegung der Fahrzeuge implementiert werden. Aus Zeitgründen konnte hier nur eine einfache Animation erstellt werden, die einer gleichmäßigen Bewegung auf dem kürzesten Weg entsprechen sollte. Qt bietet zu diesem Zweck ein *Animation Framework,* welches dafür gut geeignet ist. Es basiert darauf, einem Objekt einen Start- und Zielpunkt zuzuweisen. Die Geschwindigkeit wird entsprechend der Animationsgeschwindigkeit angepasst. Um eine konstante Geschwindigkeit für alle Pfade zu erhalten, wurde die zurückgelegte Strecke für jeden Pfad berechnet und die Animationsdauer an diesen Wert angepasst. Schwierig wurde es, die Animation auf einem bestimmten Pfad ablaufen zu lassen, da eigentlich nur eine Bewegung zwischen zwei Punkten vorgesehen ist. Qt enthält eine Funktion, die es ermöglicht, einen Punkt auf einem Pfad zu ermitteln, der einen bestimmten Prozentsatz der gesamten Pfadlänge vom Anfang des Pfades entfernt ist. Diese Funktion wurde mit der Positionsaktualisierung und der verstrichenen Zeit kombiniert, um eine Bewegung auf dem Pfad umzusetzen. Zu Demonstrationszwecken, wurde eine Linie hinzugefügt, welche dem gewählten Pfad entspricht. Die folgenden Bilder zeigen den Pfad zu vier verschiedenen Punkten.

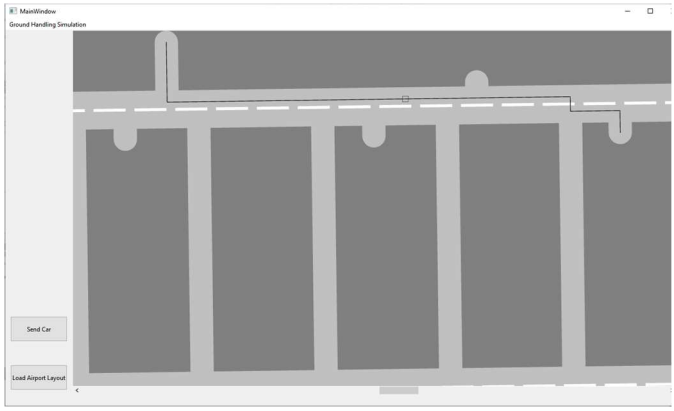

Abbildung 20: Weg zu erstem Punkt.

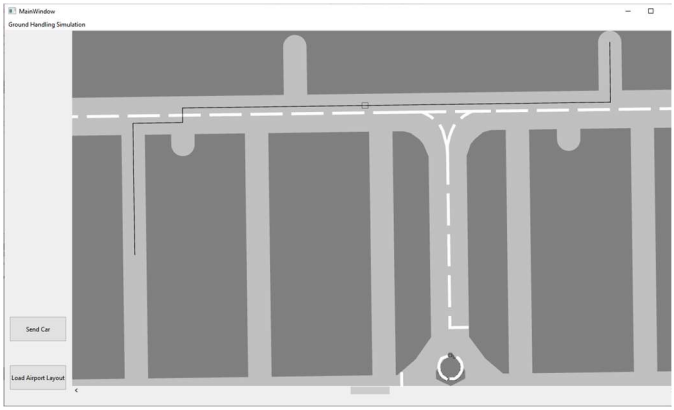

Abbildung 21: Weg zu zweitem Punkt.

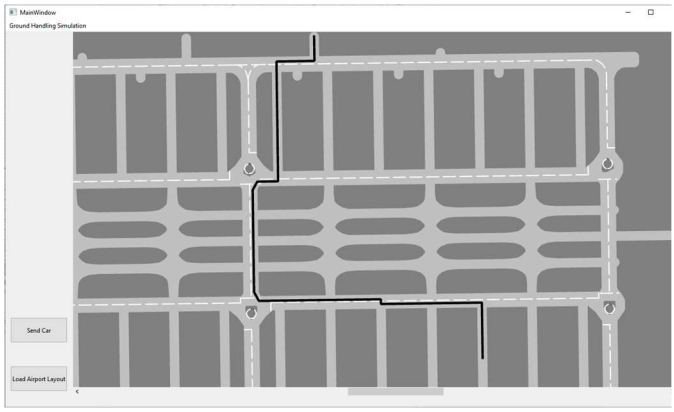

Abbildung 22: Weg zu drittem Punkt.

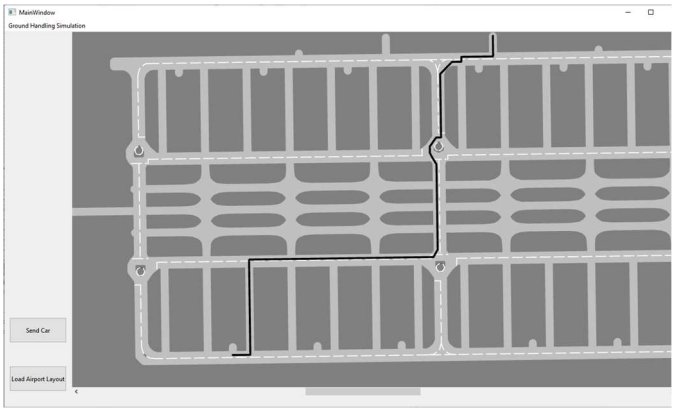

Abbildung 23: Weg zu viertem Punkt.

Wie zu sehen ist, berücksichtigt der Routing-Algorithmus nicht, dass auf der rechten Straßenseite gefahren werden muss. Generell werden noch keine Verkehrsregeln beachtet. Der Taxiway wird nicht befahren, da dieser im Graphen nicht mit dem übrigen Straßennetz verbunden ist. An dieser Stelle konnte auch der Fahrzeugpool getestet werden. Bei fünf Fahrzeugen im Pool konnten auch nur maximal fünf Fahrzeuge losgeschickt werden. Zu Testzwecken wurde das Limit entfernt und es konnte auch bei einer großen Anzahl von Fahrzeugen kein Leistungsabfall festgestellt werden. Es konnte beobachtet werden, dass der Algorithmus zuverlässig einen Weg zum festgelegten Ziel findet. Die Anforderungen 1.1 und 1.4 wurden damit teilweise erfüllt, müssen aber noch um weitere Funktionen erweitert werden.

6. Zusammenfassung und Ausblick

Das Ziel der Arbeit war es, ein Konzept für eine Verkehrssimulation der Bodenabfertigung zu erstellen. Dazu wurden zunächst die Grundlagen der Bodenabfertigung beschrieben. Des Weiteren wurden Verspätungen und Variationen im Ablauf der Bodenabfertigung betrachtet. Daraufhin konnten verschiedene Simulationsmodelle und deren mathematische Beschreibung dargelegt werden. Zusätzlich wurden agentenbasierte Modelle und Routingalgorithmen erläutert.

Auf Basis der Theorie konnte eine Anforderungsanalyse durchgeführt werden. Die identifizierten Anforderungen sind dabei keineswegs final. Es ist zu erwarten, dass die Anforderungen in zukünftigen Arbeiten in weitere Anforderungen unterteilt werden und neue Anforderungen entstehen. Außerdem konnte das Programm in mehrere Module unterteilt werden. Zum Schluss wurde ein erster Prototyp der Simulation erstellt. Es kann ein Flughafen mittels Datei importiert werden und Fahrzeuge können mittels Wegfindungs-Algorithmus einen Weg von einer Position zu einer anderen fahren.

Für zukünftige Arbeiten sollte der Fokus zunächst auf der Implementierung des IDM liegen, um das Verkehrsgeschehen realistischer abzubilden. Dazu gehört auch die Implementierung von Verkehrsregeln, wie z. B. das Rechtsfahrgebot, wofür die Daten jedoch gegebenenfalls erweitert werden müssen. Außerdem sollte die grafische Darstellung verbessert werden. Danach wäre ein Beginn der Implementierung der Bodenabfertigung möglich. Dazu müssen unterschiedliche Fahrzeugtypen erstellt werden und es müssen Daten über den Ablauf der Bodenabfertigung hinterlegt werden. Zudem muss eine Methode gefunden werden, wie die Bodenfahrzeuge an ihre entsprechende Serviceposition am Flugzeug fahren sollen, sobald diese auf dem Vorfeld am Flugzeug angekommen sind. Ich denke, das IDM ist hierfür nicht geeignet, da hier keine Pfade existieren, auf denen sich die Fahrzeuge bewegen könnten.

Qt hat sich im Verlauf der Programmierung bisher als richtige Wahl herausgestellt. Das UI konnte schnell erstellt werden und auch bei der Darstellung des Flughafens oder der Fahrzeuge kam es zu keinen größeren Problemen. Es wurde hier jedoch auf die Open-Source Variante gesetzt, weshalb überlegt werden muss, ob das Projekt als Open-Source weitergeführt werden oder eine Portierung auf die lizensierte Version vorgenommen werden soll.

Bei der Programmierung stellte sich jedoch heraus, dass das verwendete *Animation Framework* nicht für die Implementierung des IDM geeignet ist. Hierfür müsste z. B. auf die Funktion QTimer zurückgegriffen werden, mit der eine Funktion in einem bestimmten Zeitintervall ausgeführt werden kann. Eine erste Implementierung halte ich in ein bis zwei Monaten für möglich. Problematisch ist hier die Bestimmung des Abstands zum vorausfahrenden Fahrzeug. Qt hat aber auch eine Funktion zur Kollisionserkennung, die dafür verwendet werden könnte.

Die Implementierung der Verkehrsregeln und Kollisionsvermeidung schätze ich ebenfalls auf einen Zeitraum von ein bis zwei Monaten. Dazu müssen die Fahrzeuge in der Lage sein, auf ihre Umgebung und damit auch auf andere Fahrzeuge zu reagieren, was mit einem gewissen Programmieraufwand verbunden sein kann.

Eine Verbesserung der grafischen Darstellung sollte nicht sehr aufwändig sein. Eine Umsetzung könnte innerhalb eines Monats möglich sein. Eventuell sind zusätzliche Daten für die Darstellung von Flughafengebäuden etc. erforderlich.

Die Umsetzung der Fahrzeugbewegung am Flugzeug schätze ich als aufwändiger ein. Zunächst muss eine geeignete Lösung für die Bewegung zur Serviceposition gefunden werden. Außerdem müssen viele Daten eingegeben werden, wie z. B. verschiedene Fahrzeugtypen, deren Daten und deren Position beim Service. Auch der Pushback ist eine Herausforderung, da hier eine Interaktion zwischen Fahrzeug und Flugzeug erforderlich ist. Ich rechne hier mit einem Arbeitsaufwand von vier bis sechs Monaten.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] K. Straube, "Operationelles Konzept für einen Ground Handling-Disponenten," 2013.

[2] A. Schlegel, Bodenabfertigungsprozesse im Luftverkehr, 2010.

[3] Eurocontrol, "Performance Review Report of the European Air Traffic Management System in 2021," 2022.

[4] Airbus, "A320 - AIRCRAFT CHARACTERISTICS - AIRPORT AND MAINTENANCE PLANNING," 2022.

[5] Airbus, "A330 - AIRCRAFT CHARACTERISTICS - AIRPORT AND MAINTENANCE PLANNING," 2022.

[6] Treiber and Kesting, "Verkehrsdynamik und -simulation," 2010.

[7] J. Dallmayer, "Simulation des Straßenverkehrs in der Großstadt," 2013.

[8] A. Kesting, M. Treiber and D. Helbing, "Agents for Traffic Simulation," 2008.

[9] D. Salles, S. Kaufmann and H.-C. Reuss, "Extending the Intelligent Driver Model in SUMO and Verifying the Drive Off Trajectories with Aerial Measurements," 2020.

[10] German Aerospace Center (DLR), "Routing - SUMO Documentation," DLR, 2023. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Routing.html. [Accessed 31 05 2023].

[11] X. Cui and H. Shi, "A*-based Pathfinding in Modern Computer Games," 2011.

[12] A. Patel, "Introduction to A*," [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. [Accessed 28 Juni 2023].

[13] Stackscale, "Stackscale," [Online]. Available: https://www.stackscale.com/blog/most-popular-programming-languages/. [Accessed 19 Juni 2023].

[14] TechVidvan, "TechVidvan - Python Introduction," [Online]. Available: https://techvidvan.com/tutorials/python-tutorial/. [Accessed 2023 Juni 20].

[15] TechVidvan, "TechVidvan - Introduction to C++," [Online]. Available: https://techvidvan.com/tutorials/introduction-to-cpp-tutorial/. [Accessed 20 Juni 2023].

[16] TechVidvan, "TechVidvan - Java Introduction," [Online]. Available: https://techvidvan.com/tutorials/pros-and-cons-of-java/. [Accessed 20 Juni 2023].

[17] Codeguru, "Codeguru - C# vs Java," [Online]. Available: https://www.codeguru.com/csharp/c-sharp-vs-java/. [Accessed 21 Juni 2023].

[18] Unity Technologies, "Unity," [Online]. Available: https://unity.com/de. [Accessed 21 Juni 2023].

[19] Vention, "Vention - The pros and cons of the Unity game engine," [Online]. Available: https://ventionteams.com/blog/unity-pros-and-cons. [Accessed 21 Juni 2023].

[20] The QT Company, "QT Framework," [Online]. Available: https://www.qt.io/product/framework. [Accessed 21 Juni 2023].

[21] Eclipse, "Github - Sumo," [Online]. Available: https://github.com/eclipse/sumo. [Accessed 22 Juni 2023].

[22] Aimsun, "Aimsun Next Users Manual," [Online]. Available: https://docs.aimsun.com/next/22.0.1/UsersManual/ScriptIntro.html. [Accessed 22 Juni 2023].

Abkürzungsverzeichnis

A

ASU ..... *Air Starter Unit*

G

GPU ..... Ground Power Unit

I

IATA ..... International Air Transport Association
IDM ..... Intelligent Driver Model

T

TRT ..... Turn Round Time