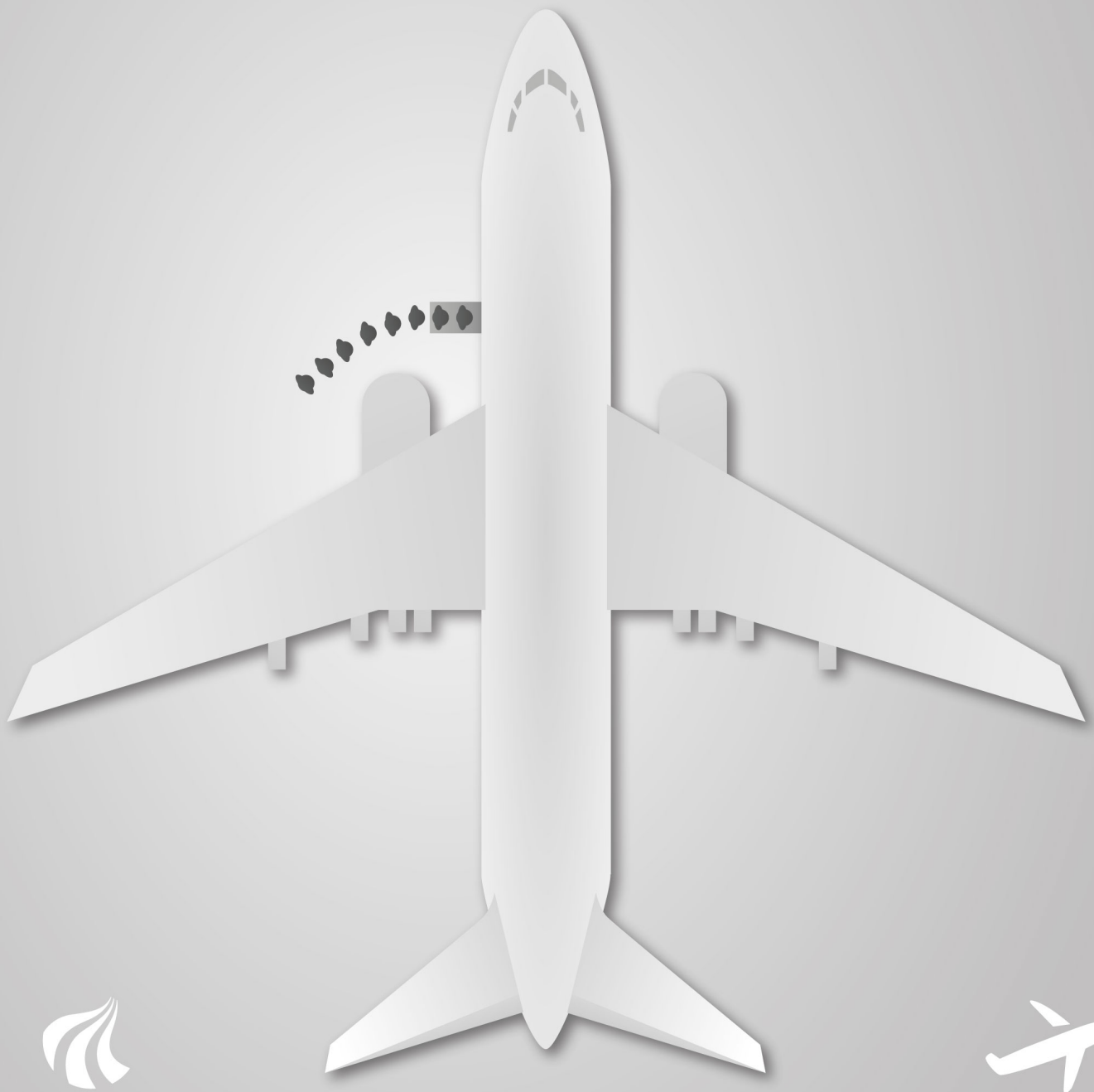
Boarding an Airplane

Fall Semester 2019 – P1 project by DAT1-A307 – Computer Science – Aalborg University



AALBORG UNIVERSITY
DENMARK

A REPORT BY DAT1-A307
AALBORG

AALBORG UNIVERSITET
STUDENTERRAPPORT

1. Semester
Computer Science
Strandvejen 12-14
9000 Aalborg
http://www.cs.aau.dk

Title:

Boarding an Airplane

Project:

P1-project

Project Period:

Fall Semester 2019

Project Group:

Dat 1 - A307

Participants:

Adam Moloney Stück
Benjamin Porsmose Pedersen
Casper Munk
Emil Gybel Henriksen
Niels Ulrik Gajhede
Tais Vemmelund Hors
Theodor Risager

Supervisor:

Ingo van Duijn
Sahar Sattari

Page Numbers: 48

Date of Completion: 18-12-2019

Abstract:

Through analysis of the subject "Boarding an Airplane" the problem statement "How does carry-on luggage affect the boarding time in a Ryanair Boeing 737-800 for different boarding procedures?" emerges. By using the knowledge gained through the problem analysis and the problem statement, a model is created. The model is used as a foundation to simulate boarding an airplane. The simulation outputs the time it takes for different boarding procedures to fully load the airplane with a group of passengers. Test cases are used to manually validate the program. To get an overview of the boarding times for the different procedures, test data has been created. The results from this data is visualized with graphs. It is then discussed what improvements and changes could be made to the model. In the end, it is concluded that the amount of passengers that bring carry-on luggage impacts boarding time for all procedures. It does however impact the slower procedures more heavily.

Preface

This is the report for the P1-Project from Aalborg University, made by group Dat1A307. The overall theme for the project is "A program that solves a problem". The goal of the project is to make a small C program of high quality, that solves a problem deduced from the problem analysis.

This report uses the "Vancouver method" for references, meaning for every reference a number is placed in the text. This number refers to the section, called "References". The list of references is sorted by the occurrence of the source in the report.

At the end of this report a glossary can be found.

A REPORT BY DAT1-A307
AALBORG

Contents

Chapter 1

Introduction

Airplane traffic is getting increasingly busier. This also means that the amount of people travelling by airplane is increasing as well. This can create a challenge for airlines, who will rapidly have to handle more passengers. A study shows that air traffic increased by 6% from 2017 to 2018.[1] And this number will most likely increase over the next couple of years.[2] One of the challenges airlines will have to face, is the balance between boarding efficiency and customer satisfaction.

The main objective for the report is to see how carry-on luggage can affect boarding time for a specified airplane, with the use of different boarding procedures. A simulation will be made for the procedures, which will then print out an end result of the fastest boarding procedures, so that decision-makers can use it for the airline.

In Chapter 2, the problem analysis for boarding an airplane is described and discussed. This chapter discusses which airplane to use for the simulation and also defines the initial structure of the simulation.

Modeling is an important part of making a simulation, and is described in Chapter 3. A model describes the rules of a simulation. The model used in this report, will describe how the environment (the airplane) and the agents (the passengers) behave. It will be discussed which factors are to be taken into account to model the boarding of an airplane, given the project's limited time frame.

In Chapter 4, the implementation of the model and its structure is described. The chapter includes code snippets from the implementation to show interesting or advanced sections of the implementation.

This leads to Chapter 5, where the implementation is validated through tests against the model and where data from the program is analysed. The tests handle different scenarios to cover a wide variety of scenarios in the implementation. After the validation of the implementation, the simulation's results regarding the problem statement are analysed. Three different boarding procedures are compared in the program. These are: a modified version of *Steffen*, *Back-to-Front* and *Random*.

Chapter 6 contains the conclusion where the problem statement is answered.

Chapter 2

Problem Analysis

The initial problem is described as: *create a simulation of boarding an airplane to measure boarding time, that can be used by decision makers at an airline to choose the appropriate boarding procedure.*

Before the problem domain can be analysed, the fundamental theory in simulation and modeling must be described, as the theory is used as a base for the analysis. When the relevant theory has been described, the analysis of boarding an airplane can be carried out and the problem statement and the specifications can be defined.

2.1 Simulations

"Simulation uses a model to develop conclusions providing insight on the behavior of real-world elements being studied. Computer simulation uses the same concept but requires the model be created through computer programming."[3, p2]

A simulation follows the rules defined by the model to output results. In an ideal simulation the output would be equal to what would be given in the real world. This means that the real world needs to be modeled so that the simulation can produce the same results.

Using computer simulation as a problem solving technique has several benefits. For example, computer simulation can be used to experiment on a pre-existing system without modifying the system itself. A new concept can be simulated before actually implementing it. Additionally, computer simulation allows for quick analysis. Once a system has been modeled, it can be simulated at speeds much faster than reality.[3, p4]

There are several different computer simulation methods but this project will only be focusing on Agent-Based Modeling as this project's system consists of a group of passengers and their behavior when boarding an airplane. Trying to simulate the passengers as a single unit would be complex and choosing a boarding procedure becomes impractical. To simplify the system the Agent-Based simulation method can be used.[4, p200]

Agent-Based Modeling is described as addressing *"simultaneous interactions of multiple agents to simulate, recreate, study, and predict complex phenomenon. The concept of agent-based modeling is that an overall behavior emerges through the micro-level interactions of individual agents. The primary assumption is that simple local behaviors generate complex height level behavior"*[3, p10]

2.2 Simulation Modeling

According to Pascal Cantot and Dominique Luzeaux in the book *System of Systems*[4, p57-66], a model is a representation of a system and is designed to solve a specific problem. In the following sections Pascal's and Dominique's definitions of a model is described.

A model can be either static or dynamic. A static model does not evolve over time, but keeps an identical system at all times. A dynamic model is the opposite and evolves over time. The evolution is defined through laws of evolution and events.

Another characteristic of a model is whether it is a deterministic or stochastic model. A deterministic model is defined as a model that gives the same output every time the same input is given. This means the model does not contain randomness. The stochastic model is defined as a model that can provide different outputs when simulated.

The qualities of a model can describe how well, the model is designed for its purpose. Overall there are four main areas of quality:

- Simplicity
- Validity
- Accuracy (As accurate as possible regarding the simulation goal)
- Efficiency (The most efficient model concerning the intended goal)

A simple model is simplistic in all aspects of the model. This will make it easier to understand and use the model for its purpose. A complex model can be difficult to handle on its own and impossible to handle within a bigger system or simulation.

An accurate model's output is close to or matches observations from the real world. For this reason, having data observations from the real world to test the model's accuracy is a great tool for designing and testing the model for mistakes.

A valid model is measuring the purpose and the needed precision of the model and validating whether or not the needed precision is presented in the model. The accuracy of the model looks at the absolute values from real observations, but a model can be valid, even though the accuracy is not precise. It should however be noted, that these qualities are often closely related.

An efficient model is one that contains the qualities previously described and it runs with the lowest amount of resources and ignores irrelevant information for faster results and performance.

One of the methods used to ensure the quality of the model is to use an appropriate level of decomposition on the system. The decomposition of a system splits the model into subsystems and sub components, which gives the model more detail. Choosing an appropriate level of decomposition for the goal of the simulation is essential in the modeling phase.[4, p72]

2.3 Boarding an Airplane

The act of boarding an airplane, can be defined in a variety of ways. The most commonly used definition, is that a group of people coming from a bus, gate or airport, need to get into their respective seats in an airplane. Only after this, can the airplane start flying. This is a very simple description of boarding, but gives an essential picture of what boarding is.

The specific definition of boarding is mostly a subjective term. Thus, to simulate boarding an airplane, it has to be clarified what boarding is defined as. Some airports and airlines define boarding, as when the passengers start the gating process, while some define it as when the passengers reach the aisle.[5]

2.3.1 Boarding Procedures

When an airplane is boarded by passengers, different boarding procedures can be used. A variety of procedures exists, some are fast while others offer better comfort for the passengers.[6]

A boarding procedure can be described as the order in which the passengers enter the airplane. This order is mostly influenced by the passengers' assigned seats. This means that the boarding procedure only acts outside the airplane.

From this definition it can be concluded that boarding procedures, in essence, are different ways of ordering the queue of passengers. Therefore, the procedure is not part of the actual simulation of boarding an airplane, but a separate part, acting before the simulation is run.

2.3.2 The Static System

This project will use the definition of boarding an airplane used by "Aircraft Boarding - Data, Validation, Analysis". This is done, to make the simulation validatable with the same collected data, as the project mentioned above. This leads to the definition being: [7]

The boarding procedure begins when the first passenger enters the front-end door on the plane and ends when the last passenger is seated and all the passenger's carry-ons is in the airplane's overhead compartments.

To get an understanding of the system in question, a static system analysis will be made. In the static system analysis, the following must be determined: [4, p72]

- Input and output
- Entities in the system
- Relationships between entities
- The system environment

The input should include a list of passengers, their assigned seat and whether or not they bring carry-on luggage with them on board. The output of the simulation should be the boarding time for each boarding procedure and which procedure was the fastest. The reason for including the results for slower procedures is that the airline might want to use a

non-optimal procedure, as its' benefits such as customer satisfaction for example, may outweigh the longer boarding time.

As mentioned earlier in section 2.1 on page 4, the passengers are described as agents. All the agents are entities boarding an airplane inside the simulation. These entities all have the same level of granularity. As all agents spawn outside the airplane, none of them are permanent entities and they all enter the system after it starts. This makes all the agents temporary entities. The temporary entities will all be affected by time, as the system will have the entities go through the boarding process, measured in time.

The system's entities impact the output of the system. As the system is defined as boarding an airplane, the airplane itself is an entity outside the system, but still shapes the seating area, which is the system environment. This is important to clarify, as the agents are not affected by the airplane's physical size, but by the size of its seating area. Lastly, the boarding process can be affected by the defined starting position. This position can vary depending on the airport facilities and airplane size. So the airport, where the airplane is located, is an entity in the system environment.

For the model to be valid, a look at all the entities and environments is required. These include: The airplane, the airport and the passengers.

2.3.3 Choosing an Airplane Type

The first component of the system to investigate is the airplane. The reason for this is its importance for the simulation. The type of airplane affects boarding time. Boarding time is primarily affected by the number of entrances, and the passenger capacity.

There are a lot of different airplanes that all have several layouts. This means that if a model is created to accommodate all these airplanes, the model would become complicated, broad and inefficient. This goes against the design principals stated in section 2.2 on page 4. Therefore, the model should only accommodate a single airplane.

When narrowing the model down to a single airplane, it should be the most relevant one. To do this an airline should be chosen, as they are the ones operating the different airplanes.

Choosing an Airline

Generally, airline business models can be sorted into seven different groups. These are defined by multiple sources and are applicable to most airlines in the world. These groups are:[8] [9, p5-13]

- Low Cost Carriers (LCC)
- Full Network Carriers
- Holiday Carriers
- Regional Carriers
- Traditional Carriers
- Integrators
- Hybrid Carriers

LCCs focus on providing only the essentials and offer close to no luxury at all. This means that their main selling point is their low ticket prices. To keep their tickets affordable, LCCs have to keep their own costs down. LCCs often use one or just a few types of airplanes. This is done to keep costs down. Some examples of LCCs include Ryanair and EasyJet.[9, p8]

This type of airline would gain the most from a simulation. The reason for this is that the LCC business model is based on cheap prices. They need to cut time in order to save money and a simulation will enable them to optimize their boarding times and reduce time wasted. Seeing as LCCs often use only one type of aircraft[9, p8], the simulation model would also be applicable to a large fleet, rather than just one airplane in a fleet of several different airplanes.[9, p7].

Europe is the region with the largest percentage of its airplanes operated by LCCs, and the percentage is growing.[10] This means there is a lot of potential in Europe. The largest LCC in Europe, would be the best representative for the whole LCC category, because as previously stated, most LCCs often use one type of aircraft, making the biggest LCC the most applicable airline.

In Europe, the largest LCC is Ryanair.[10] Ryanair's fleet consists of 450 Boeing airplanes from the 737-800 series.[11]

The Airplane

Now that the airplane has been chosen, it is possible to further design the model. To do this, the level of decomposition has to be determined. This is done in regard to how boarding was defined in section 2.3. As mentioned earlier, it can be determined that the model should only include the passenger seating area because the cockpit and toilets of the airplane are irrelevant for the simulation's goal.

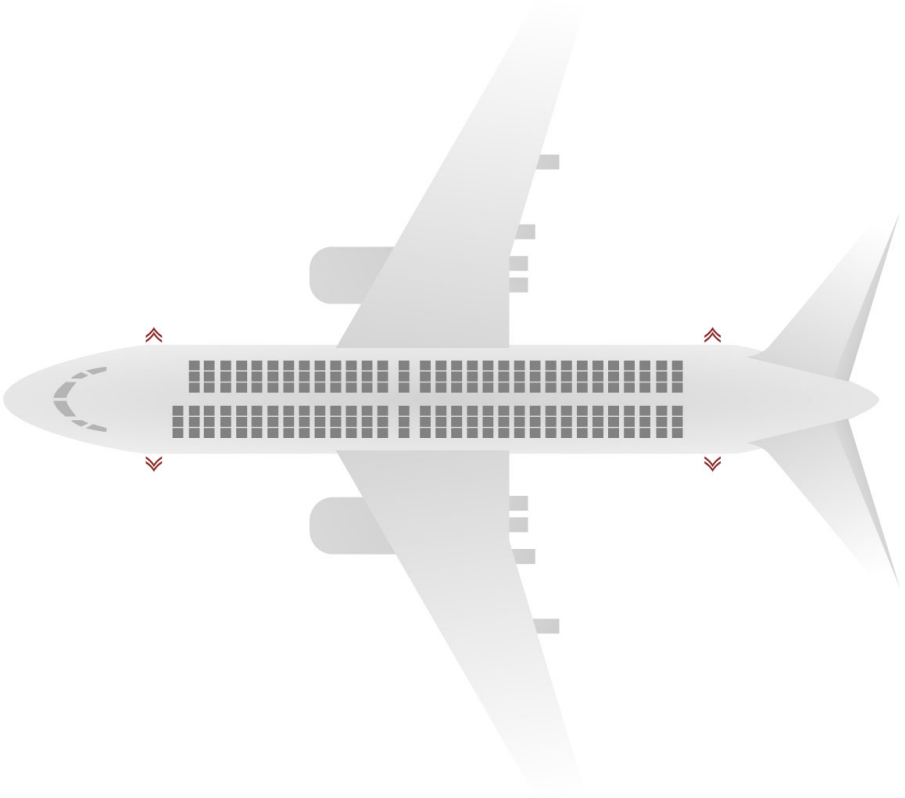


Figure 2.1: Model of a Boeing 737-800, showing passenger seats and entrance points

To determine whether the model should be static or dynamic, a look at the physical dimensions of the airplane is needed. An illustration of the airplane can be seen in figure 2.1. The Boeing 737-800 has a total length of 39.47 meters, the middle aisle is 0.51 meters wide and there are 189 seats distributed over 32 rows.[11, 12] These physical characteristics do not change from airplane to airplane or over time, meaning the environment, the airplane's interior, is static and therefore the model

should be static. In addition, the environment has no stochastic elements making it deterministic.

As mentioned previously, the airplane has 32 rows, but the last row is number 33. This is because many airlines respect the superstition of certain numbers. In Ryanair's case, they chose not to include the number 13 for row numbers, as many people believe this number causes crashes or is just unlucky in general.[13]

In order to make the model valid, some areas of the model should be discussed. These are:

- Seats
- Entrance

Seats come in two variations, standard seats and seats with extra legroom. The seats with extra legroom are located in the first row and the rows behind the emergency exits found in the middle of the airplane. Including these seat variations would make the model more accurate. Due to the fact the extra room will add more distance for every passenger walking by. However, implementing this would complicate the model unnecessarily, as the difference between standard seats and those with extra legroom is negligible.

The distance from the entrance to the first row is needed to make a valid model, since the test data defines boarding as starting at the entrance, as discussed in section 2.3. This distance has been measured as being $\sim$1.91 m according to the Boeing 737-800 design manual.[14, p59]

2.3.4 Airport Facilities

In order to determine which airport facilities should be included in the model, knowledge of where Ryanair operates, as well as the airport facilities they utilise when boarding, is required.

Ryanair operates in 239 airportsm and 86 bases in Europe and North Africa, with routes connecting 40 countries.[15]

Being a LCC, it is in Ryanair's interest to reduce the overall cost of aircraft turnarounds. This is done by only making use of basic services and tailoring their procedures thereafter. For example, Ryanair would opt to use the integrated ladders of their Boeing 737-800s and a second set of stairs at the rear end of the airplane, rather than using only a

Passenger Boarding Bridge (PBB).[16, p1] This requires the passengers to walk across the apron by foot or to be transported by an apron bus, if the airplane is parked at a remote stand. Some airports however, force airlines, Ryanair included, to use a PBB.

As mentioned in section 2.3, the defined environment of the model is the seating area of the airplane, therefore only the amount of entrance doors used is deemed relevant for the model. The simulation should take into account that Ryanair prefers to board from both ends, but also allow for scenarios where only one entrance door is available.

2.3.5 Passengers

It has been determined that the passengers are going to be the agents of the model, further decisions have to be taken regarding the properties of the passengers. These decisions are whether they are to be deterministic or stochastic and in either case, to what level of decomposition it is relevant to look at passenger traits, in order to get as high a validity as possible.

The passengers of Ryanair are people, and human behavior is difficult, if not impossible, to model accurately. This is because of the many unknown and unique characteristics of humans. This makes the nature of a human model stochastic and difficult to validate. [4, p92]

Within the system of boarding an airplane, human behavior has an impact on boarding time. However the focus of the simulation is to output the quickest boarding procedure and not to simulate human behaviour. Including human behaviour would complicate the model and make the system very difficult to validate. For this reason, the passengers will be modeled to be deterministic.

The age of the passengers matters in the context of boarding an airplane, as the age of a given passenger determines their average walking speed. Ryanair's passengers range from all age groups, with the majority of them being between 25 and 34 years.[17, p. 7] But as the model will not include statistics of age and walking speed, this factor has been excluded and an average walking speed has been chosen to represent all passengers.

A passenger's choice of whether to bring a carry-on bag onto the airplane, can be assumed to prolong boarding process for the passenger, as they now not only have to get seated, but also have to place their bag into the overhead compartment. In addition to the one free carry-on that non-priority passengers are allowed to bring onto the airplane, parents

of infants are also allowed to bring one changing bag, which means that some people might bring two bags into the boarding process. Besides the use of carry-ons, there is also the possibility, that a passenger might have a need to bring medical equipment on board, which will also take up time and space.[18]

The context in which the passengers board the airplane, might also have an impact on their boarding speed. For example, whether they are in a hurry to get the flight going or not. Passengers on a leisure trip might have a more relaxed state of mind, whereas someone on a business trip would just want the airplane to get in the air and therefore board faster.[17, p. 8]

Another thing that may affect boarding time could be the socioeconomic status of the passengers. Ryanair appears to appeal to a wide variety of potential customers, though some groups are more underrepresented than others.[17, p. 8] As mentioned previously, the passengers' states of mind might be related to how fast they board. Someone who is in a bad mood, could make some kind of scene while boarding, hence lengthening boarding times. This could be a problem for Ryanair since 18% of the passengers reported having an unpleasant time boarding. Improving the passengers' satisfaction with the boarding process may therefore lead to faster boarding times.[19] But as mentioned earlier in section 2.3.5 on the preceding page, human behavior is hard to account for and is therefore not included in the model.

In the context of a simulation, the level of decomposition of the system, is chosen to be down to if they bring a carry-on or not. The reason for not including other parameters such as toddlers, is that only 0.2% of Ryanair's passengers are under the age of 18, and a negligible amount of those are between 8 days and 24 months old.[20] This amount is so minuscule that it would not improve the validity of the model and is considered to be an edge-case.[4, p72]

2.4 The System and its Model

2.4.1 Environment

The environment of the system is the seating area of the airplane, which can be seen in figure 2.2. As discussed in section 2.3.3, the model is a static and deterministic model of the Boeing 737-800.

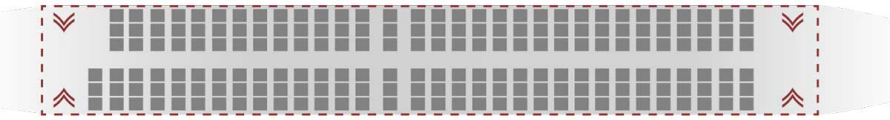


Figure 2.2: Illustration of the seating area of a Boeing 737-800

The model will include 32 rows of 6 seats, with an aisle in the middle. The first row only contains 3 seats.

The data set which will be used to validate the model defines boarding as starting at the entrance of the airplane.[7, p2] Therefore the distance between the entrance and the first row must be known. For the Boeing 737-800 this distance was measured to be $\sim$1.91 m.

2.4.2 Agents

In section 2.3.2 the input-parameters are formulated as the number of passengers and how many of those that bring carry-on luggage.

The passengers in section 2.3.5 are defined as deterministic. The passengers' parameters are average walking speed and carry-on stowing time.

Because agents are the only dynamic part of the model, a state diagram has been formulated to illustrate the agents behaviour, which can be seen in figure 2.3.

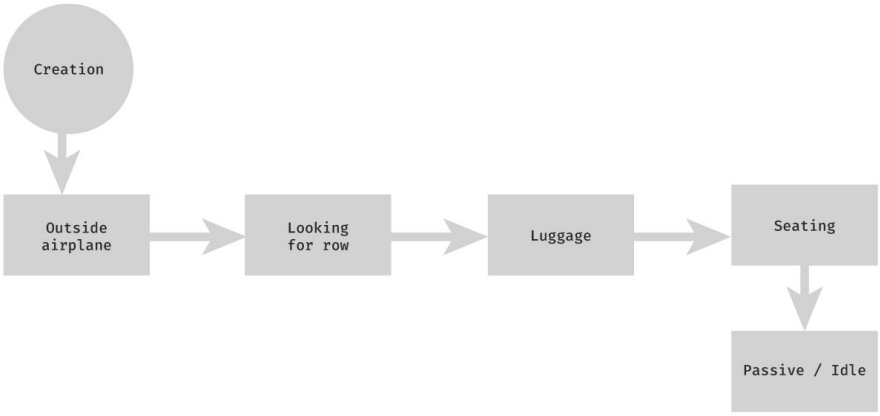


Figure 2.3: A state-diagram for passengers boarding an airplane

A passenger can be in five different states. First, all the agents are created and spawn outside the airplane and are therefore outside the environment. This state can be seen as the queue, leading into the airplane. Then, whenever there is room for an agent to get into the airplane, the agent first in line outside the airplane enters the *Looking for Row* state.

When an agent is looking for its row, it checks if its seat is in the current row. If this is not the case, the agent will move on to the next row. However, the agent can only move to the next row, if no other agent is standing in front of it in the aisle. When the agent finds its row, it enters the *Luggage* state, if it has luggage, and stows it in the overhead compartment.

Now the agent is ready to get to its seat. This is done in the *Seating* state, where the agent checks if another agent is blocking the path to the seat. An example of this could be, if the agent has a window seat and another agent has already gone through the boarding process and is seated in the aisle seat. It will then take the agent standing longer to get to its seat, because of the agent seated in the aisle seat.

2.5 Problem Statement

How does carry-on luggage affect the boarding time in a Ryanair Boeing 737-800 for different boarding procedures?

When someone brings carry-on luggage with them on an airplane, the boarding procedure can be expected to be longer than that of a similar boarding procedure, in which the passengers do not bring along any carry-on luggage, since an extra action needs to be performed. But how do vastly different boarding procedures compare to each other, when a given percentage of the passengers have carry-on luggage with them? This is a question that decision makers at cost-effective airlines most likely would be interested in an answer to, as it will show which boarding procedures are more effective at certain percentages of passengers with luggage.

2.5.1 Requirements

For the program to be successful, it has to fulfill these requirements:

1. A minimum of two different boarding procedures
2. The input is a given number of passengers with two parameters
   - Their seat number
   - Whether or not they bring carry-on luggage
3. It has to follow the behavior of the designed boarding model.
4. It outputs the boarding times for each boarding procedure as an ordered list starting with the shortest time first
5. Output is displayed as a formatted list

Nice to Haves

1. A visual representation of the simulation running
2. The option to choose between boarding with one entrance or two entrances

Chapter 3

The Model

3.1 Simulation

The overall simulation sequence from start to finish is represented in figure 3.1 below.

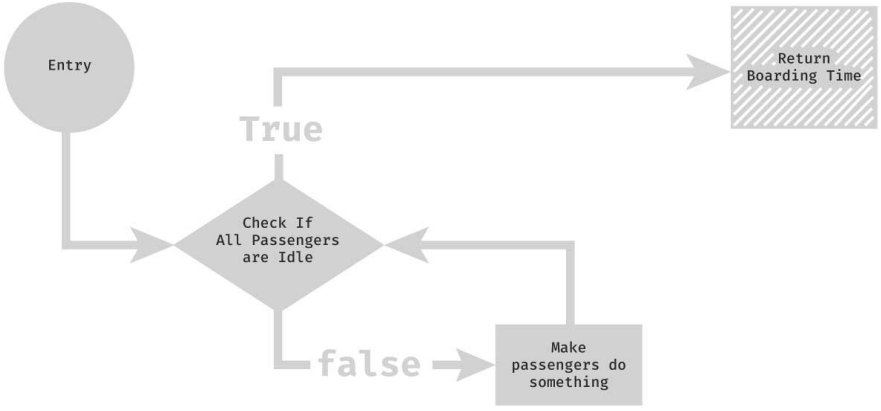


Figure 3.1: A diagram showing the model

When the simulation is run it will count the time until all passengers are idle. A passenger becomes idle when seated. In each iteration all passengers that are not idle will do an action, this could be entering the airplane or walking to the passenger's assigned seat.

3.2 Environment - Airplane

The following figure (3.2) represents the environment of the model.

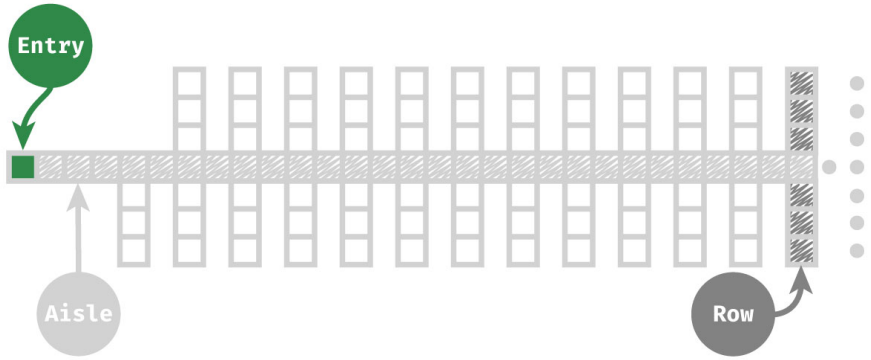


Figure 3.2: An abstract model representing the airplane

It is important to note, that the visual representation above is not the actual environment, as there are less seats on figure 3.2 than in the actual model. The intention is purely to show what is defined as a row, the aisle and the entry point.

The green square represents the position in which a passenger enters the airplane. The horizontal grey line represents the aisle of the airplane, while the vertical lines represents the rows of the airplane. In figure 3.3 below, the complete environment is illustrated.

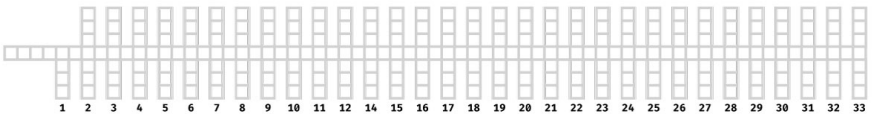


Figure 3.3: Illustration of the complete environment

There are twice as many squares on the horizontal line, as there are rows. In this case there are $2 \cdot 32$ squares horizontally, because there are 32 rows in the Boeing 737-800. In addition to the 64 squares, there should be added four additional squares because, of the length from the entrance to the first row. Since the distance is 1.91 m and each square is 40.64 cm the amount of squares is rounded down to 4. This can be seen in figure 3.3 and 3.2

Each square in the model, can only accommodate a single passenger at a time. This means, for example, that a passenger must wait behind another passenger, that is in the act of stowing luggage.

3.3 Agent - Passenger

3.3.1 Passengers

A passenger can be in five different states as illustrated in 3.4.

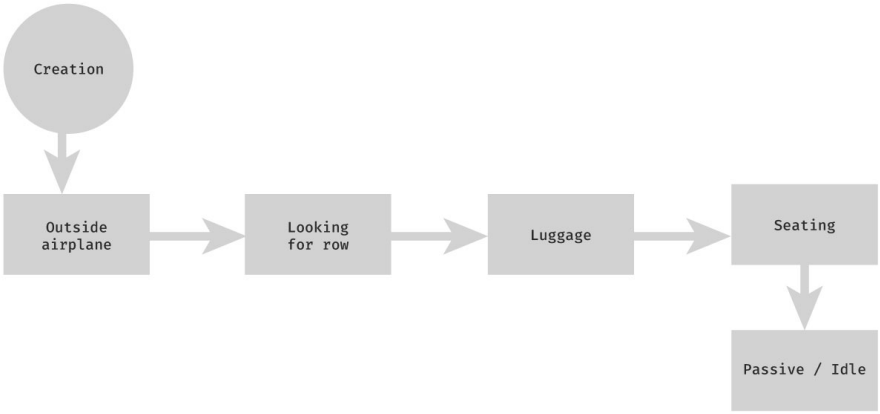


Figure 3.4: Model of the different states a passenger can be in

When an agent is created it is given the state *Outside Airplane*. When the agent enters the airplane it progresses to the next state: *Looking for Row*.

The *Looking for Row* state, is where the agent walks down the aisle looking for the row with its assigned seat. When the passenger has walked to the assigned row, the agent moves on to the next state which is *Luggage*. The *Luggage* state is where the agent places carry-on luggage in the overhead compartment, if they have any.

After the *Luggage* state, the agent enters the *Seating* state, where the agent moves from the aisle to its assigned seat. For the agent to get seated all other agents blocking the way to the seat, must stand up and move out into the aisle, before the agent can get to its seat. When the passenger is seated it moves to the last state: *Idle*. In this state the agent waits for all other agents to get seated.

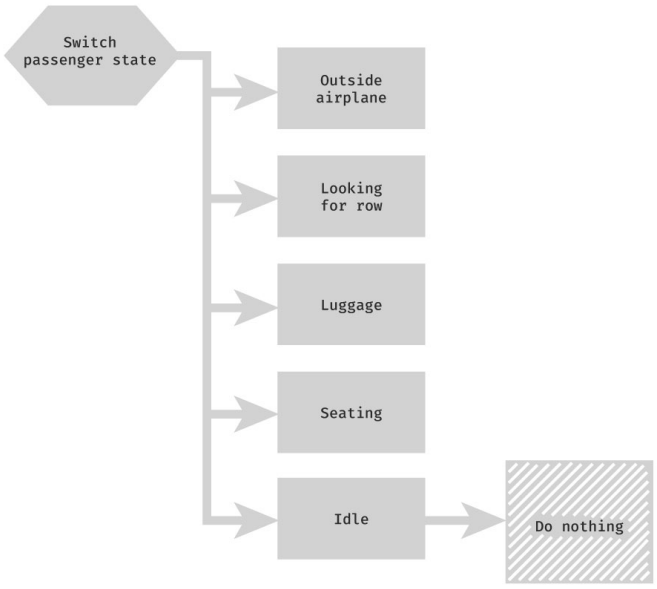


Figure 3.5: Model of Passenger Behaviour

The behaviour of the passenger is described in figure 3.5. Depending on the current state of the agent it will perform differing actions, in each iteration of the simulation loop, as described in section 3.3.2.

3.3.2 Passenger States

Outside Airplane

When the agent is in the *Outside Airplane* state it will check if it is the next agent to enter the airplane. If the first square in the airplane is accessible it will move into the airplane. Otherwise it will wait until it can enter the airplane.

A square is considered accessible if and only if the square is empty or the agent occupying it wants to and can move.

Looking for Row

Looking for Row is the first state the agents enters after entering the airplane. This state's purpose is to take the agent from the arrival point to the agent's row in the airplane. The state's logic is presented in figure. 3.6 on the following page.

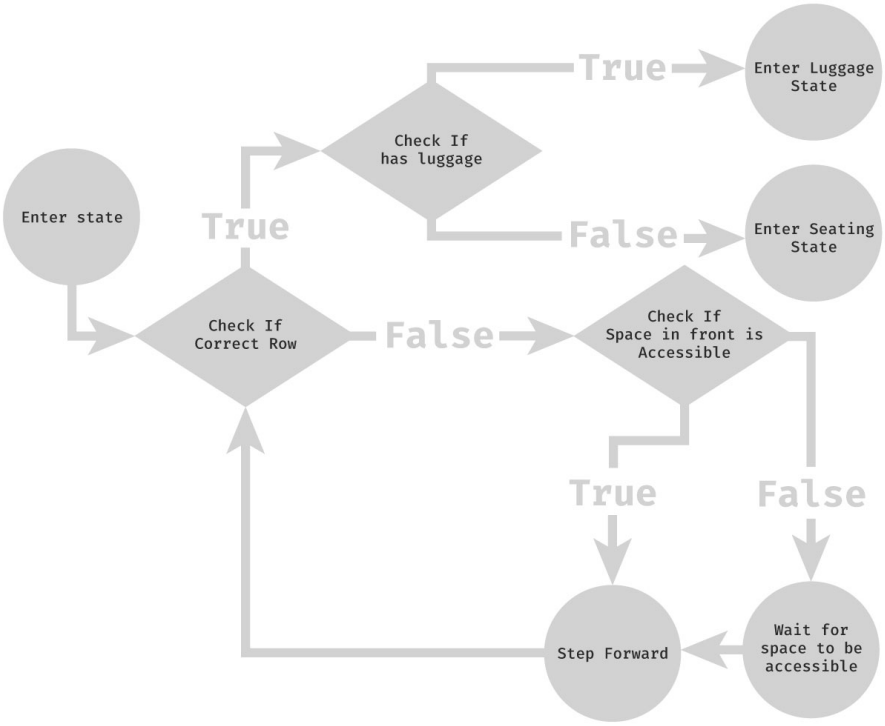


Figure 3.6: A diagram representing the *Looking for Row* state

When the agent is looking for its row, it will start by checking if the row it is standing at is the correct one. If this is the case, the agent will either change to the *Luggage* or *Seating* state, depending on whether or not the passenger has luggage. If the agent is not standing in the correct row, it will check if the square in front is accessible. If the square is accessible, the agent will move to the square. If not, the agent will wait for the square to become accessible, before it moves forward.

Luggage

Figure 3.7: A diagram representing the agent luggage state

In the *Luggage* state, the agent stows away their luggage. The next time the agent is updated, it moves on to the *Seating* state.

Seating

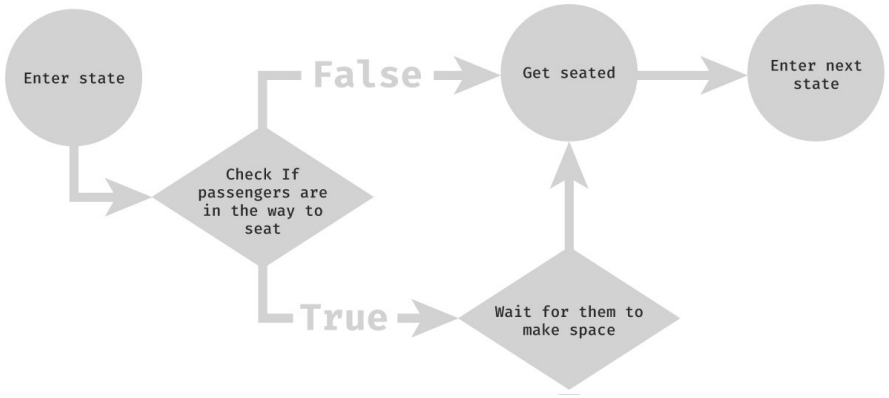


Figure 3.8: A diagram representing the agents seating state

In the *Seating* state, the agent checks if it can move from the aisle into the assigned seat without crossing any passengers. If this is not possible, the passenger must wait in the aisle until the seated agents make room for the passenger and only then can the agent walk to its seat.

To achieve this, an equation for how long a passenger should block the aisle, have been formulated instead of simulating passengers moving back and forth. This formula is described below.

$$T = 2(O + S(p)) - E(p) - 1$$

- p is the passenger seated closest to the aisle in the targeted row
- O is the number of passengers already seated in the targeted row
- $S(x)$ is a function that takes a passenger as input and returns the number of seats from the aisle to the passenger seated
- $E(x)$ is a function that takes a passenger as input and returns the number of currently empty seats from the aisle to the targeted passengers assigned seat-number
- T is the amount of ticks to wait

3.3.3 Idle

When the agent is in the *Idle* state, it is seated and will not do any more actions in the simulation.

3.4 Properties

In order for the model to become valid, properties must be set correctly. These properties are:

- Time to walk between two squares
- Stowing time

The time it takes to walk from one square to the next in the model, determines the speed of the agent and must therefore be found. As described in section 2.4.1 one seat is 81.28 cm wide, and since the model contains 2 squares per seat, one square is 40.64 cm wide. To determine the time to walk 40.64 cm, the walking speed of the agent must be determined.

In section 2.3.5 it was found that the avg. age of Ryanair's passengers is between 25 and 34 years old. Healthline.com reports that the average walking speed of an adult in this age group is 1.34 m/s to 1.43 m/s.[21] In this report the the walking speed will be the average which is 1.39 m/s. This means that the agent will use 0.29 s per square. But because, this speed is not measured when walking down a narrow aisle crammed in between other passengers, this number will be set back to 60 % (this should be looked at when comparing to real-world data). So, the resulting time per square is 0.56 s. However, this number is just an approximation to simulate this lowered walking speed and does not represent reality.

The average luggage stowing time, according to the German Aerospace Center, is 13.9 s.[7, p6] and $13.9/0.56 = 24.8214$. Therefore, stowing luggage equates approximately to moving one square forward 25 times.

Chapter 4

Implementation

The implementation of the model has been created in the programming language C99. The implementation's code has been broken down into several sub-functions, following the concepts of top-down programming. Each of the sub-functions have a specific task for the implementation. The different sub-functions handle:

- Input reading and formatting
- Ordering passengers by boarding procedure
- Simulation
- State: Looking for row
- State: Luggage
- State: Seating

In the model, the time it takes to walk from one square to the next is 0.56 s. In the implementation this represents 1 tick with a time-step of 0.56 s. This means that stowing luggage takes 25 ticks.

In the following sections the different functions and their relation to the model will be presented.

4.1 Input Reading and Formatting

This section handles input for the simulation. Its purpose is to read a *.txt* file and convert the data into a format the program can understand and process. Underneath is an example of what this *.txt* file could look like.

```
1A 1
1B 1
1C 1
1D 1
1E 0
1F 1
2A 1
END OF FILE
```

Listing 4.1: Example input file

The file consists of a number of lines equal to the number of passengers boarding the airplane. In the example above the amount of passengers is 7. Each line represents a passenger, which includes the assigned seat-number and whether the passenger brings carry-on luggage. The first number is the passenger's assigned row and the following letter is its seat. The last number is a boolean value, which represents whether or not the passenger brings a carry-on. As seen in the *.txt* file above, where line 1 represents a passenger with carry-on luggage, assigned to Row 1 Seat A.

The input file is converted and inserted into the struct shown below:

```
struct passenger {
    point currPos;
    point seatPos;
    state currState;
    int ticksToWait;
    int hasLuggage;
};
```

Listing 4.2: The passenger struct used in the program

The *currPos* is the current position of the passenger inside the airplane when the simulation is running. *seatPos* is the position of the assigned seat. *currState* represents the state of the passenger. *ticksToWait* is the amount of ticks the passenger has to wait before the passenger can do the next action. *hasLuggage* is whether or not the passenger has luggage.

4.2 Ordering the Passengers by Boarding Procedure

This section handles the passenger's order before boarding. This order is determined by the boarding procedure used. This implementation includes the following 3 boarding procedures:

- Random
- Back-to-front
- Steffen modified

The function for ordering passengers called from *main.c* is shown below. This function takes the list of passengers which should be reordered in accordance to the chosen procedure.

```
1 void QueuePassengers(passenger* passengers, int numPassengers,
      boardingProcedure procedure)
```

Listing 4.3: Function prototype called from *main.c*

4.2.1 Random

The *Random* procedure is straight forward. It shuffles the passenger queue uniformly at random by using the Knuth shuffling algorithm. This is done to simulate passengers getting into the queue in no particular order.

```
// Shuffles the passengers randomly
void randomQue(passenger *passengers, int numPassengers) {
    knuthShuffle(passengers, 0, numPassengers);
}

void knuthShuffle(passenger *passengers, int start, int end) {
    for (int i = start; i < end; i++)
        swap(passengers, randIndex(i, end), i);
}
```

Listing 4.4: Function which reorders passengers according to the *Random* procedure

In this implementation the shuffling is done using *Knuth shuffling*. Which swaps passengers uniformly at random.

4.2.2 Back-to-Front

The *Back-to-Front* procedure is very similar to the *Random* procedure but instead of shuffling all passengers at once, the passengers are divided into two groups. Both groups are then shuffled individually.

```
void backToFrontQue(passenger *passengers, passenger
    *sorted_passengers, int numPassengers) {
    int mid = numPassengers / 2;
    knuthShuffle(sorted_passengers, 0, mid);
    knuthShuffle(sorted_passengers, mid, numPassengers);
    CopyArray(passengers, sorted_passengers, numPassengers);
}
```

Listing 4.5: Function for reordering the passengers according to the *Back-to-Front* procedure

4.2.3 Steffen Modified

The *Steffen Modified* procedure orders passengers from back to front in four different groups. Each groups consists of every second row from one side of the airplane, as shown in figure 4.1 on the next page.

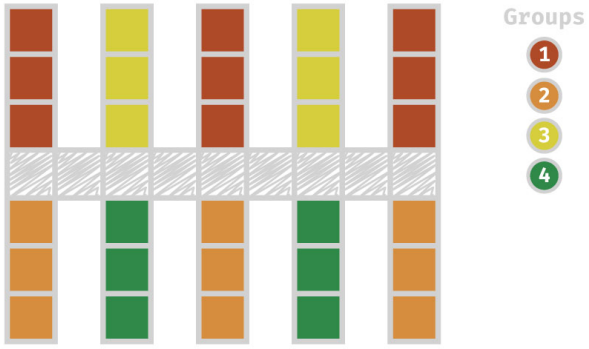


Figure 4.1: Illustration showing the grouping *Steffen Modified* procedure utilizes

Shown below is the implementation of the *Steffen Modified* procedure. This code consists of three for-loops running through each group described by figure 4.1. The second loop runs through each row descending by two, while the third loop runs through each seat in that given row. In the inner loop, it looks for a passenger assigned for that seat, if a passenger is found, the passenger is added to queue.

```
1  void steffenQue(passenger *passengers, passenger
       *sorted_passengers, int numPassengers)
2  {
3      int group, row, seat, check, placeInQue = 0, p;
4
5      // For all groups (Even A-C, UnEven A-C, Even F-D, UnEven F-D)
6      for (group = 0; group < 4; group++) {
7          // For every second Row
8          for (row = (ROWS - (group/2)); row > 0 ; row -= 2) {
9              ...
10             // For each seat from A-C or F-D
11             for (seat; seat > check; seat--) {
12                 p = binarySearch(sorted_passengers, seat, row,
                       numPassengers);
13                 // If a passenger with this Row and Seat is found
14                 if (p != -1) {
15                     passengers[placeInQue] = sorted_passengers[p];
16                     placeInQue++;
```

Listing 4.6: Functions used for reordering the passengers according to the *Steffen Modified* procedure

4.3 Simulation

This section looks into the simulation of the passengers behaviour. Firstly, all passengers must be reset and have their variables set to their starting values.

4.3.1 The Passenger Simulation Loop

```
1  float runSimulation(passenger *pArr, int pArrSize) {
2      int tick = 0, seatedPassengers = 0;
3
4      // Keep ticking until all passengers are seated
5      while(seatedPassengers < pArrSize) {
6          for(int i = 0; i < pArrSize; ++i) {
7              // Update Passenger
8              ...
9          }
10         tick++;
11     }
12
13     return tick * TIMESTEP_IN_SECONDS;
14 }
```

Listing 4.7: Function that runs the simulation

The function above, is the one handling the actual simulation loop. It keeps count of how many iterations (*ticks*) the simulation runs through before returning the total boarding time. When the function returns, it multiplies the *tick* with the time each iteration takes in seconds (*TIMESTEP_IN_SECONDS*), because this converts the iterations into total boarding time. This is needed, because the simulation does not run in real time, but as fast as the computer can run it.

In each iteration of the loop all passengers are updated. Passengers are updated differently depending on what their current state is. This can be seen in the following code snippet.

```
void updatePassenger(passenger *pArr, int pArrSize, int i) {
    switch(pArr[i].currState) {
        case LookingForRow:
            stateLookingForRow(pArr, i);
            break;
        case Luggage:
            stateLuggage(pArr, i);
            break;
        case Seating:
            stateSeating(pArr, pArrSize, i);
            break;
        ...
    }
}
```

Listing 4.8: Function that updates the passengers

In this implementation the passengers are updated from front to back. This means, that the first passenger to enter the airplane will also be first to be updated. This ensures that all passengers can move simultaneously, because everyone in front of a given passenger **p** has already been updated by the time it is **p**'s turn to update.

The function only returns when all passengers become *Idle*. To keep track of this, the local variable *seatedPassengers* is used. Every time a passenger enters the idle state, this variable is incremented by 1. When this number reaches the number of passengers in the simulation, the function returns the boarding time.

All the states; *Looking for Row*, *Luggage* and *Seating* will be discussed in the following sections.

4.3.2 State: Looking for Row

In the *Looking for Row* state, the passenger **p** steps forward if no one is blocking them. The function *getPassengerAhead* is used to determine if **p** is free to step forward. If there is no passenger ahead, **p** can move forward. Alternatively, if there is someone ahead, **p** can only move forward if there is at least one empty square between **p** and the passenger ahead. This logic can be seen in the following code snippet. This process is repeated until **p** reaches their assigned row. **p**'s state is then set to either *Luggage* or *Seating* depending on whether or not they have carry-on luggage.

```
void stateLookingForRow(passenger *pArr, int i) {
    ...

    // Get index of the passenger in front (-1 if none)
    int infront = getPassengerAhead(pArr, i);

    // Is there a passenger infront?
    if(infront == NO_PASSENGERS_AHEAD) {
        // Move forward
        pArr[i].currPos.x += STEP_DISTANCE;
    } else {
        // Is the distance between two passengers is >= 2 Steps
        if(pArr[infront].currPos.x - pArr[i].currPos.x >=
            STEP_DISTANCE * 2) {
            // Move forward
            pArr[i].currPos.x += STEP_DISTANCE;
        }
    }
}


/* Returns the index of the passenger ahead of the passenger
    with index pIndex.
   Returns -1 if no passenger is ahead. */
int getPassengerAhead(passenger *pArr, int pIndex) {
    ...
}
```

Listing 4.9: Logic for the *Looking for Row* state

4.3.3 State: Luggage

In the *Luggage* state the passenger has to stow away any luggage. The code snippet below, follows the model as described in section 3.3.2.

```
void stateLuggage(passenger *pArr, int i) {
    // If the passengers has no luggage or has stowed it away
    if(pArr[i].hasLuggage == STOWED_AWAY_LUGGAGE ||
        pArr[i].hasLuggage == NO_LUGGAGE) {
        pArr[i].currState = Seating;
        return;
    }

    // Set the passenger has luggage to stowed away
    pArr[i].hasLuggage = STOWED_AWAY_LUGGAGE;
    // Set time to wait
    pArr[i].ticksToWait = LUGGAGE_STORE_TICKS;
}
```

Listing 4.10: Function that handles logic for the *Luggage* state

4.3.4 State: Seating

In the model, the *Seating* state is described such that a passenger blocks the aisle until all passengers in the way to their seat have made room. Below are the functions used to implement the equation introduced in section 3.3.2. *getClosestToAisle* is equivalent to p in the formula, and *countPassengersInRow* is equivalent to $E(x)$.

```
void stateSeating(passenger *pArr, int pArrSize, int i)
int countPassengersInRow(passenger *pArr, int pArrSize, int pI)
int getClosestToAisle(passenger *pArr, int pArrSize, int pI)
```

Listing 4.11: Functions that handles logic for the *Seating* state

4.4 Putting it all Together

In this section the main function is described. This function combines all the program's sub-functions, runs the simulations with several different boarding procedures and prints the results, ordered from fastest to slowest.

The function starts by reading the passenger input file and converts it into something the program can use (as described in 4.1 on page 24).

```
// Initial setup
FILE *passengerSource = fopen("passengerText.txt", "r");
passenger *pArray = gather(passengerSource);
int pAmount = getPassengerAmount(passengerSource);
```

Listing 4.12: Reads and formats input file

Next, the function orders the passengers in queue appropriately for the boarding procedure in question. The function then proceeds to run the simulation and finally saves the time it took to board the airplane. This process is repeated for each boarding procedure.

```
// For each Boarding procedure
for (int i = 0; i < BOARDINGALGORITHMS; i++) {
    // Order Passengers
    QueuePassengers(pArray, pAmount, (boardingProcedure) i);
    // Calculate Boarding Time
    boardingCalculations[i].time = runSimulation(pArray, pAmount);
    boardingCalculations[i].procedure = i;
}
```

Listing 4.13: Behavior for running the simulations and storing the results

Finally, the function sorts the returned boarding times in ascending order and prints them it in a formatted list.

Chapter 5

Validation and Analysis

5.1 Validating the Implementation

To test the validity of the implementation it is tested against manual calculations of the model. These tests are separated into cases. These cases check the implementation for different types of calculations and the aim is that they together cover almost every case of input and output.

The manual calculations are done using an illustration of the environment, where agents are updated according to the model. This is illustrated in figure 5.1

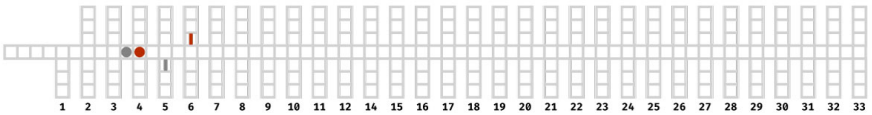


Figure 5.1: Illustration used for the manual calculations, each passenger has its own color

Case 1:

The first case tests the amount ticks it takes for one passenger without luggage to get seated (become idle). Figure 5.2 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger without luggage needs to be seated at the start of the airplane.

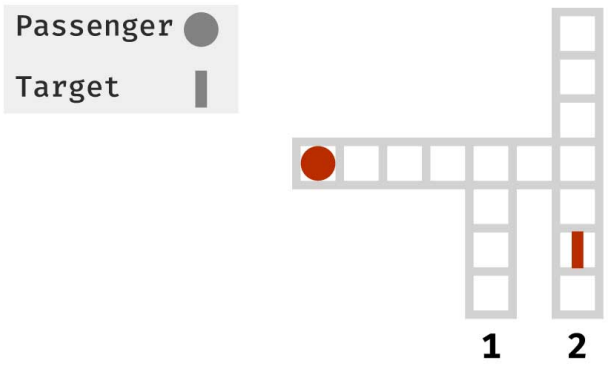


Figure 5.2: Case 1 scenario

Passenger moves 6 squares forward	+6
Passenger changes to seating state	+1
Passenger moves to assigned seat	+1
Passenger changes to idle state	+1
Total ticks:	9
Program output from same scenario:	9

Case 2:

The second case tests the amount of ticks it takes for a passenger with luggage to get seated. Figure 5.3 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger with luggage moves to a seat that is located at the start of the airplane.

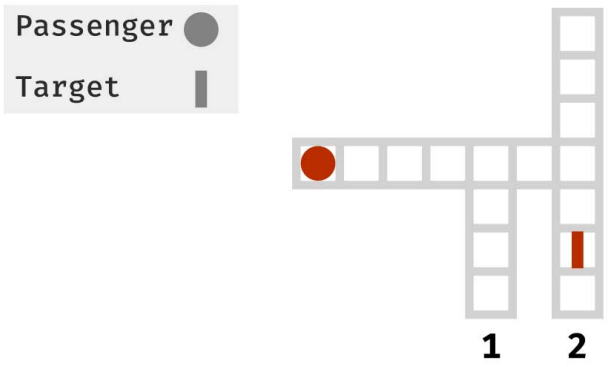


Figure 5.3: Case 2 scenario

Passenger moves 6 squares forward	+6
Passenger changes to luggage state	+1
Passenger stores luggage away	+25
Passenger changes to seating state	+1
Passenger moves to assigned seat	+1
Passenger changes to idle state	+1
Total ticks:	35
Program output from same scenario:	35

Case 3:

The third case tests the amount of ticks it takes for one passenger with luggage to get seated at the back of the airplane. Figure 5.4 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger with luggage needs to be seated at the end of the airplane.

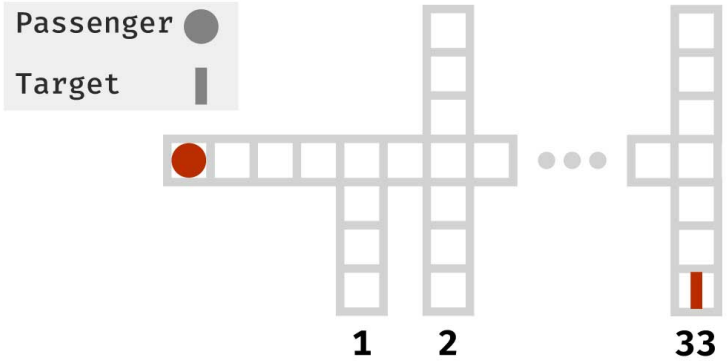


Figure 5.4: Case 3 scenario

Passenger moves 66 squares forward	+66
Passenger changes to luggage state	+1
Passenger stores luggage away	+25
Passenger changes to seating state	+1
Passenger moves to assigned seat	+1
Passenger changes to idle state	+1
Total ticks:	95
Program output from same scenario:	95

Case 4:

The fourth case tests the amount of ticks it takes for two passengers, one with luggage and one without, to get seated. Figure 5.5 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger must wait behind another passenger to get to their assigned seat.

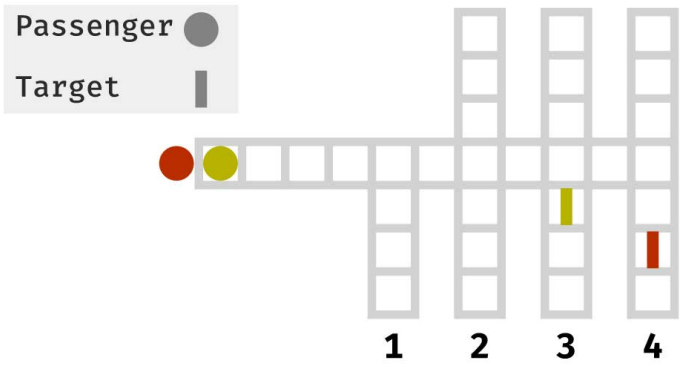


Figure 5.5: Case 4 scenario

Passengers move 8 squares forward	+8
Yellow changes to luggage state	+1
Yellow stores luggage away	+25
Yellow changes to seating state	+1
Yellow moves to assigned seat	+1
Yellow changes to idle state	+1
- Red moves 1 square forward	
Red moves 2 squares forward	+2
Red changes to seating state	+1
Red moves to assigned seat	+1
Red changes to idle state	+1
Total ticks:	42
Program output from same scenario:	42

Case 5:

The fifth case tests the amount of ticks it takes for two passengers without luggage to get seated. Figure 5.6 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger must wait behind another passenger without luggage to get to their assigned seat.

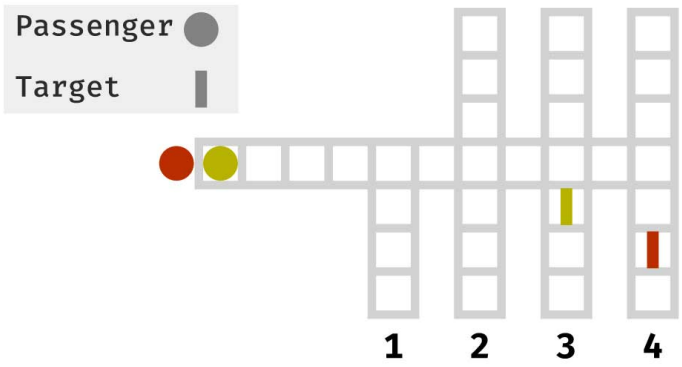


Figure 5.6: Case 5 scenario

Passengers move 8 squares forward	+8
Yellow changes to seating state	+1
Yellow moves to assigned seat	+1
Yellow changes to idle state	+1
- Red moves 1 square forward	
Red moves 2 squares forward	+2
Red changes to seating state	+1
Red moves to assigned seat	+1
Red changes to idle state	+1
Total ticks:	16
Program output from same scenario:	16

Case 6:

The sixth case tests the amount of ticks it takes for two passengers without luggage to get seated. Figure 5.7 represents this test case. This case is done to test if the simulation correctly handles a situation where two passengers without luggage need to be seated at the start of the airplane.

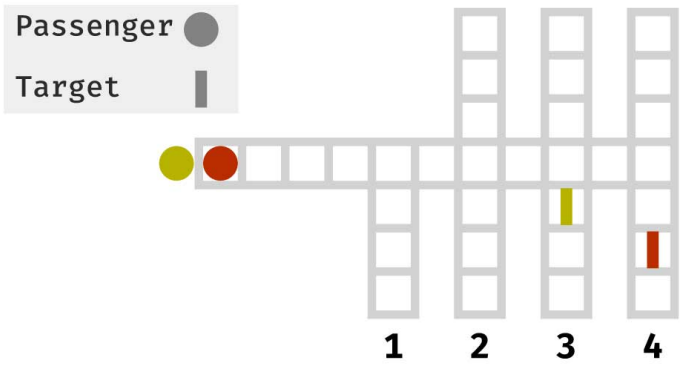


Figure 5.7: Case 6 scenario

Passengers move 9 squares forward	+9
Red moves 1 square forward	+1
- Yellow changes to seating state	
Red changes to seating state	+1
- Yellow moves to assigned seat	
Red moves to assigned seat	+1
- Yellow changes to idle state	
Red changes to idle state	+1
Total ticks:	13
Program output from same scenario:	13

Case 7:

The seventh case tests the amount of ticks it takes for three passengers without luggage to get seated. Figure 5.8 represents this test case. This case is done to test if the simulation correctly handles a situation where a passenger needs to cross another passenger to get to its target seat. This case also checks if the simulation correctly handles a situation in which there are seated passengers that are not in the way of a passenger's target seat.

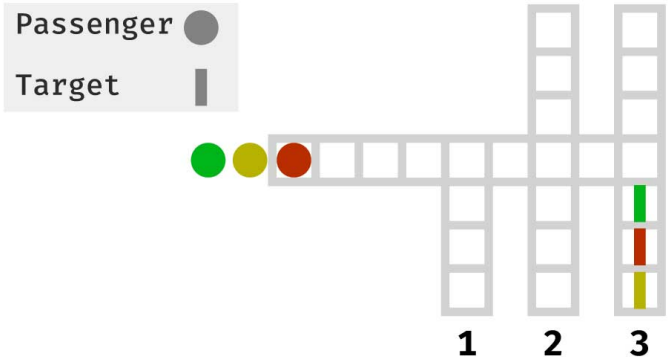


Figure 5.8: Case 7 scenario

Passengers move 8 squares forward	+8
Red changes to seating state	+1
Red moves to assigned seat	+1
Red changes to idle state	+1
- Yellow and Green moves 1 square forward	
Yellow changes to seating state	+1
Yellow waits until Red moves out of the way	+5
Yellow changes to idle state	+1
- Green moves 1 square forward	
Green changes to seating state	+1
Green moves to assigned seat	+1
Green changes to idle state	+1
Total ticks:	21
Program output from same scenario:	21

Case 8:

The eighth case tests the amount of ticks it takes for three passengers without luggage to get seated. Figure 5.9 represents this test case. This case is done to test if the simulation will correctly handle a situation in which a passenger needs to cross more than one passenger to get to their assigned seat.

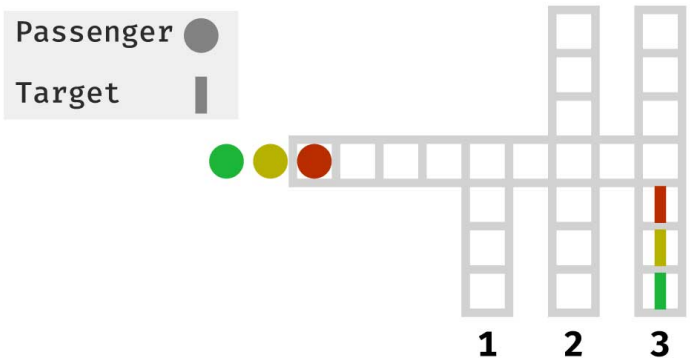


Figure 5.9: Case 8 scenario

Passengers move 8 squares forward	+8
Red changes to seating state	+1
Red moves to assigned seat	+1
Red changes to idle state	+1
- Yellow and Green moves 1 square forward	
Yellow changes to seating state	+1
Yellow waits until Red moves out of the way	+4
Yellow changes to idle state	+1
- Green moves 1 square forward	
Green changes to seating state	+1
Green waits until Yellow and Red move out of the way	+6
Green changes to idle state	+1
Total ticks:	25
Program output from same scenario:	25

5.2 Data Analysis

This section will discuss the results from the simulation. All the graphs show the relationship between the percentage of passengers bringing carry-on luggage and boarding time.

The results have been generated by running the simulation on a data-set generated by 2001 different sets of passengers. Where each increment of 1 % is an average of 20 tests in the data-set. This is done to make sure that the test scenario is better represented.

These results are plotted onto their respective graphs and a trend-line has been drawn, to show the trend.

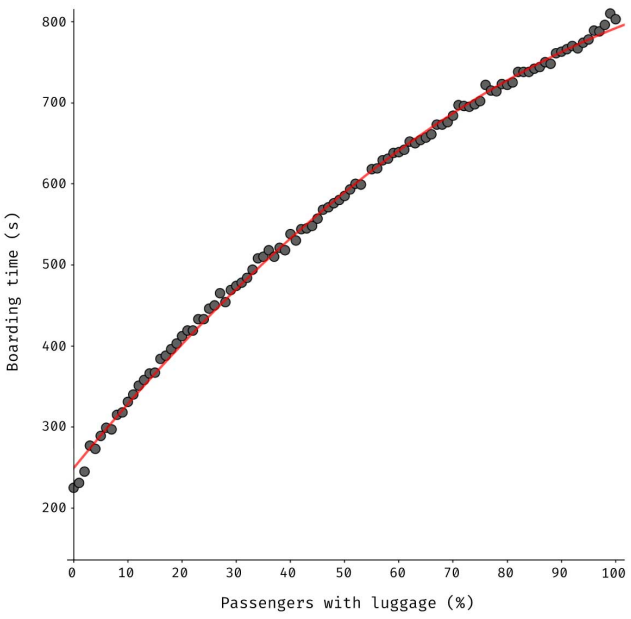


Figure 5.10: Results for boarding method *Random* with walk speed set to 0.834 m/s and stowing time at 20.85 s

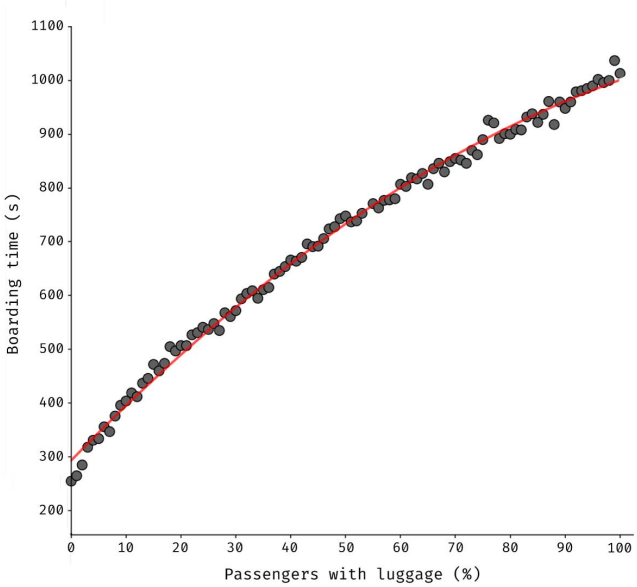


Figure 5.11: Results for boarding method *Back-to-Front results* with walk speed set to 0.834 m/s and stowing time at 20.85 s

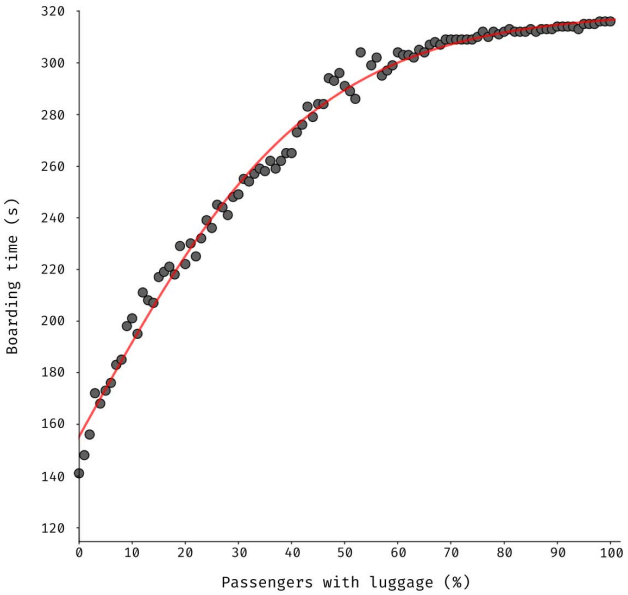


Figure 5.12: Results for boarding method *Steffen Modified results* with walk speed set to 0.834 m/s and stowing time at 20.85 s

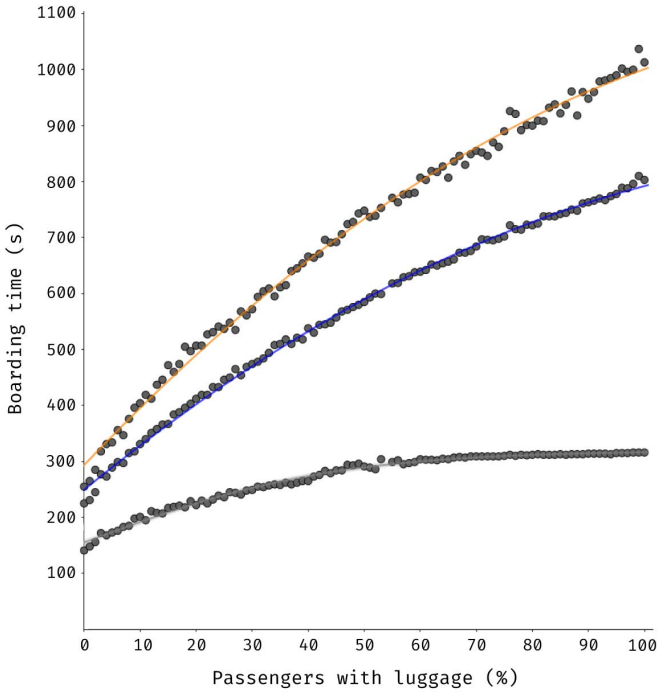


Figure 5.13: All procedures with *Random* in blue, *Back-to-Front* in orange and *Steffen Modified* in gray
with walk speed set to 0.834 m/s and stowing time at 20.85 s

Chapter 6

Discussion

6.1 The Model

In section 5.1 the implementation is tested. This is done to make sure the program correctly implements the model described in section 3.1.

From these tests, it can be concluded that the program ran identically to the manual calculations of the model in all cases, and that the implementation correctly follows the model. However, it should be noted that there might be some edge-cases that are not covered.

The data compiled in section 5.2 indicates that a Ryanair Boeing 737-800 can be boarded in 585 s, when using the *Random* procedure with 50 % passengers with carry-on luggage.

Comparing this with the data gained from another study from the German Aerospace Center[7, p4]. In the study it takes 988.5 s to board the airplane. It should be noted that the data is assumed to be using the *Random* procedure with 50 % bringing carry-on luggage.

The the data from this report is 45 % faster than the study from the German Aerospace Center. Some of this can be attributed to the fact that the other study's data did not have all passengers board at the same time, but passengers were allowed to arrive at different times. This means that the recorded data takes more time than what it would take, if all passengers were ready to board at the start of the boarding procedure.

Because the purpose of this report is not to simulate boarding an airplane accurately, but to find out how carry-on luggage impacts boarding-times, the results may change when correcting the model to the real

world. However, the general tendencies will still show that more carry-on luggage corresponds to a longer boarding-time. Although, in the future, validating the model would be of high priority.

6.2 The Data

The problem-statement states that the purpose of this report is to measure the difference in boarding-time, when an airplane is boarded with different percentages of passengers bringing carry-on luggage.

To achieve this a lot of tests are run as described in section 5.2. The data from these simulations show that an increase in the percentage of passengers bringing carry-on luggage, results in an increase in total boarding-time. However, depending on which boarding procedure is used, the percentage increase differs. This can be seen in figure 6.1.

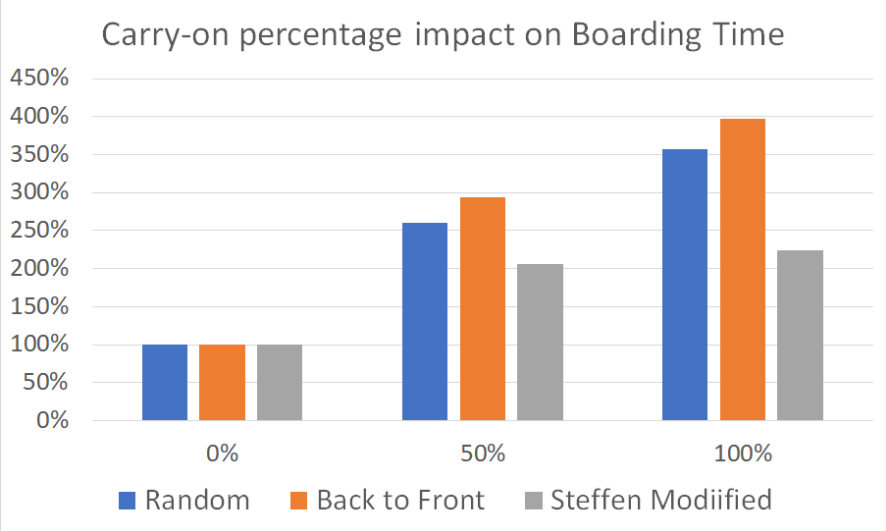


Figure 6.1: The percentage changes when the percentage of passengers bringing carry-on luggage goes from 0 % to 100 %

This difference can be attributed to the average wait-time a passenger experiences in the aisle, as bringing carry-on luggage in this simulation only changes the time a passenger will block the aisle before getting seated. Therefore, all procedures that reduce the time each passenger has to wait in line, result in an overall lower percentage increase in boarding time.

Therefore the boarding procedures resulting in the smallest increase in boarding time, are the procedures that are better at spacing out passengers. Thereby, lowering the amount of passengers queuing up because of other passengers stowing luggage or getting seated.

6.3 Putting it into Perspective

For future work on this project, the analysis could be extended to include not just how boarding time is affected by carry-on luggage when an airplane is full, but also for scenarios with variations in passenger amounts. The data gathered shows how carry-on luggage affects boarding time when boarding 189 passengers. In addition to this, the impact of the amount of passengers boarding should also be measured.

To further increase the relevance of the simulation for Ryanair, boarding from both ends of the airplane, could be implemented into the simulation. This is because Ryanair boards from both ends whenever possible.

Because agents are a central part of the simulation adding more human traits, such as passengers passing each other in the aisle or the flight personnel helping the passengers, might give a more accurate representation of the real world, than what the simulation/model currently does. This would be part of validating the model and will require comparing the output of the model with real-world data and changing possible deviations.

Chapter 7

Conclusion

This report set out to find out how to better inform decision-makers at an airline with the help of simulations. This led to a model being created of the Ryanair Boeing 737-800. This specific airplane was chosen because it represents 100 % of Ryanair's total fleet as well as Ryanair being the biggest Low Cost Carrier in Europe.

In the problem statement the question emphhow does carry-on luggage affect the boarding time in a Ryanair Boeing 737-800 for different boarding procedures?, was asked. Carry-on luggage can have an impact on the overall boarding time, because of the extra time required to stow carry-on luggage away.

From the data created by running the simulation, it can be concluded that there is a substantial increase in boarding time, when the percentage of passengers bringing carry-on luggage rises. Depending on the boarding procedure used, this increase differs a lot. In the case of the *Random* procedure this increase can go as high as 350 %, whereas when using *Steffen Modified*, the increase was only 225 %.

More factors could have been included for a more accurate model and representation of the real world. For example, a more precise description of how humans behave while boarding an airplane or the fact that Ryanair would use both ends for boarding if possible. Another factor, could be how the data created by the simulation, relates to real world data and whether or not this data can be used by the airlines.

The final conclusion is that the percentage of passengers bringing carry-on luggage impacts boarding time on a large scale. How much the boarding time is affected depends on the boarding procedure used, where faster procedures are less impacted by carry-on luggage.

References

[1] Eurostat. *Air transport statistics*. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php/Air_transport_statistics.

[2] IATA. *2036 Forecast Reveals Air Passengers Will Nearly Double to 7.8 Billion*. URL: https://www.iata.org/pressroom/pr/Pages/2017-10-24-01.aspx.

[3] Ma. Glaizel Gajardo, Ej Phillip Maliwanag, and Mark Reden Pacaon. "Simulation Definition". In: *Chapter 1: Introduction to Computer Simulation* (2009). Source does not specify the year of creation, pp. 2, 4, 10. URL: https://www.academia.edu/31226957/Chapter_1_Introduction_to_Computer_Simulation.

[4] Pascal Cantot and Dominique Luzeaux. *Simulation and Modeling of Systems of Systems*. E-book ISBN: 9781118616659. John Wiley & Sons, Incorporated, 2011, pp. 200, 57–66, 72, 92.

[5] The Flying Carpet. *The Fastest Way To Fill An Airplane*. URL: http://the-flying-carpet.com/.

[6] Cailey Rizzo and Cailey Rizzo. *Every Airline Has a Different Boarding Process - Here's What You Need to Know*. July 2017. URL: https://www.travelandleisure.com/airlines-airports/airline-boarding-groups.

[7] Michael Schultz. *Aircraft Boarding - Data, Validation, Analysis*. Report 4-8. Institute of Flight Guidance under the German Aerospace Center, 2017, pp. 2. URL: http://www.atmseminarus.org/seminarContent/seminar12/papers/12th_ATM_RD_Seminar_paper_116.pdf.

[8] Michal Swoboda. *Common Airline Business Models*. Accessed on 04/11/2019. http://airlinebasics.com/common-airline-business-models/, Feb. 2013.

[9] Reichmuth. Johannes. *Analyses of the European air transport market*. Report 5-14. German Aerospace Center, 2018, pp. 5–13. URL: https://ec.europa.eu/transport/sites/transport/files/modes/air/doc/abm_report_2008.pdf.

[10] Centre for Aviation. *Europe's LCC fleets continue to grow*. Accessed on 04/11/2019. https://centreforaviation.com/analysis/airline-leader/europes-lcc-fleets-continue-to-grow-ryanair-leads-wizz-air-has-most-orders-418772, June 2018.

[11] *Ryanair fleet information*. 2019. URL: https://corporate.ryanair.com/ryanair-fleet/.

[12] *Boeing 737-800 layout*. 2019. URL: https://www.seatguru.com/airlines/Ryanair/Ryanair_Boeing_737-800.php.

[13] David Emery. *Why Is Friday the 13th Considered Unlucky?* Oct. 2017. URL: https://www.liveabout.com/why-is-friday-the-13th-considered-unlucky-3298238.

[14] "Boeing 737-800 design manual". In: *http://www.boeing.com/* 59 (2013), p. 59. URL: http://www.boeing.com/assets/pdf/commercial/airports/acaps/737.pdf?fbclid=IwAR3Fb00NvyvX%20jZPqK6mji9qqdubwilDwbYROOqDHiY_QMpRqQT7lbZh6h7Y.

[15] *Ryanair Network*. Accessed on 06/11/2019. 2019. URL: https://corporate.ryanair.com/ryanair-facts-and-figures/.

[16] "Improvements to ground handling operations and their benefits to direct operating costs". In: *https://www.fzt.haw-hamburg.de/pers/Scholz/ALOHA/ALOHA_PUB_DLRK_09-09-08.pdf* (2009), pp. 1. URL: https://www.fzt.haw-hamburg.de/pers/Scholz/ALOHA/ALOHA_PUB_DLRK_09-09-08.pdf.

[17] Ryanair. *Exterior Branding*. Accessed on 04/11/2019. https://www.ryanair.com/doc/advertise/ExteriorBranding.pdf, Nov. 2007, pp. 8.

[18] Ryanair. *New Bag Policy From November Will Cut Check Bag Fees & Reduce Boarding Delays*. Accessed on 04/11/2019. https://corporate.ryanair.com/news/new-bag-policy-from-november-will-cut-check-bag-fees-reduce-boarding-delays/, Aug. 2018.

[19] Ryanair. *Rate my trip*. Accessed on 04/11/2019. https://corporate.ryanair.com/customer-care/rate-my-trip/, Sept. 2019.

[20] Ryanair. *Travelling with Infants*. Accessed on 08/11/2019. https://www.ryanair.com/ee/en/useful-info/travelling-with-children/travelling-with-infants.

[21] Emily Cronkleton. *What Is the Average Walking Speed of an Adult?* Mar. 2019. URL: https://www.healthline.com/health/exercise-fitness/average-walking-speed.

[22] "SURFACE MOVEMENT GUIDANCE AND CONTROL SYSTEM". In: *faa.gov* (1996), pp. 2, section 5, subsection "a". URL: https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/23193.

Glossary

apron A defined area on an airport intended to accommodate aircraft for purposes of loading or unloading passengers or cargo, refueling, parking, or maintenance.[22]. 11

entity Pascal Cantot, and Dominique Luzeaux defines it as: *"The entities of a system are the objects or components by which the system is defined."* [4, p72]. 6, 7

granularity Cambridge Dictionary defines it as *"a lot of small details included in information, making it possible for you to understand very clearly what is happening"* In our case these details are needed in the definition of the system. 7

tick One iteration of a loop. 23

time-step Determines the amount of time each iteration takes. 23