P2 Project

Bag Packer

P2 project

Group B130

Software

Department of Computer Science
Aalborg University

The 24th of May 2012

Faculty Office for Engineering and Science
Strandvejen 12-14
Phone 96 35 97 31
Fax 98 13 63 93
http://tnb.aau.dk

Title:
Bag Packing

Project period:
P2, spring 2012

Project group:
B130

Participants:
Dag Toft Børresen Pedersen
Christian Jødal O'Keeffe
Niels Brøndum Pedersen
Aleksander Sørensen Nilsson
Mette Thomsen Pedersen
Rasmus Fischer Gadensgaard
Kasper Plejdrup

Supervisor:
Karsten Jahn

Finished:
The 24th of May 2012

Synopsis:

This report contains documentation of which problems some people might have packing their suitcases before travel. These chapters will lead up to a statement of the problem which will be used in the development of a solution.
The solution will be a program based on the programming language C#. A chapter describing how the structure of the program will be, and the reason behind the construction. A section about the reflections, when the program was being made, will also be made.
One of the last chapters will concern the testing and conclusion of the finished program. Through these tests, it was discovered what should be corrected in the program. This chapter will also concern the reflections on the product to see if it actually solves the problem or why it does not solve the problem.
The product of the project is a program which helps the user pack a suitcase efficiently, and through a 3D-image of the packed suitcases the user will see where in the suitcase each item must be put.

Front Page image:
Stuart Miles / FreeDigitalPhotos.net

Date: Aleksander Sørensen Nilsson

Date: Christian Jødal O'Keeffe

Date: Dag Toft Børresen Pedersen

Date: Kasper Plejdrup

Date: Mette Thomsen Pedersen

Date: Niels Brøndum Pedersen

Date: Rasmus Fischer Gadensgaard

Prolog

This project is made by Rasmus Fischer Gadensgaard, Christian Jødal O'Keeffe, Kasper Plejdrup, Aleksander Sørensen Nilsson, Niels Brøndum Pedersen, Mette Thomsen Pedersen and Dag Toft Børresen Pedersen from Denmark. The project began on the 8th of February 2012 and was finished on the 24th of March 2012, at the University of Aalborg, where the participants are studying Software.

All participants chose University of Aalborg because they like the PBL-model. PBL stands for Problem Based Learning and it is a unique learning method which in Denmark is only used at University of Aalborg and Roskilde University Centre.

The goal with this project is to give the group skills using the programming language C# (C-Sharp), and make them better at team-work. This project is about optimizing and analysis of algorithms.

Contents

1 Introduction

This project is based on the subject "Pak kufferten" (Pack the Suitcase) seen on Appendix A with a focus on helping users packing a suitcase properly. From this project a product should emerge that will deal with a problem within the subject matter. When the product is done it will be tested. Based on the test it will be possible to evaluate the product's capability at solving the problem.

It is a requirement that the program is made in the programing language C#. It is not a requirement that the program has a graphical user interface (GUI), but it is an option. A part of the project is to document a problem and analyze it, to find the reason behind the problem. When the analysis is done, the product should be designed to handle the problem and thus solve it.

This is the essential aspect of problem-based learning: To find a problem, then work on a solution, and hopefully gain some knowledge in the process. For this to be an effective model all of the group members must participate in the process and acquire knowledge by working with the project.

This project was in the project catalog, which can be seen in appendix A. The catalog was made by the semester coordinator, Hans Hüttel, and this project was chosen based on the expectations in the group.

2 Problem Analysis

This project is based on the idea that it can be hard for some people to pack a suitcase properly. It can also be hard to take into account that the suitcase must not exceed the maximum weight limit, and some people might get fined for packing a suitcase which is to heavy. What can be done to make it easier for people to pack their suitcases? To find the answer to this, and to find a solution to this problem, it is necessary to take a closer look at what problems occur when packing.

2.1 Problems with Packing a Suitcase

In the modern society, families or groups of friends tend to go on vacation to other countries to relax and enjoy their vacation [Baumgarten, 2012].

When going on a vacation or a business trip packing a suitcase is normally needed. The passengers need all their personal items and clothes with them. The length of the stay, the purpose of the trip, and the number of people traveling together affects the weight and size of the needed luggage. This can be a problem due to the different rules or limits to your luggage when using different means of transportation.

The amount of luggage depends on the amount of people on the trip, because everyone needs their own personal items, and their own clothes. This can be a problem if the persons have a lot of shared items which have to be spread in different suitcases and need to be found afterwards and it might also affect the total weight of the luggage.

There can also be a problem with too large or too heavy suitcases if you are traveling by plane since this can result in a fee. The size of the fee varies from airline to airline and can be found on the airlines website e.g. [SAS, 2012].

To avoid this fee it can be an advantage to pack the suitcases properly and limit the choice of items so the total weight does not exceed the allowed weight.

2.1.1 Packing

Because of the weight and size restrictions when traveling by plane it can sometimes be difficult to get all the items you want to bring, into a suitcase that is small enough and still does not exceed the allowed weight. It can sometimes even be necessary to acquire more suitcases for the trip or leaving some of the items you wanted to bring.

Bringing more luggage means the passenger has to pay more to get the extra luggage on the plane [SAS, 2012]. A way they might be able to avoid buying and paying for extra

suitcases is to pack the suitcases more that you normally would. This, on the other hand, increases the suitcase's weight. This should be thought through because too much weight can be unhealthy to carry around.

Fees Regarding Luggage by Flight

The increased weight means the suitcases might exceed the limit for weight and therefore trigger a fee for overweight luggage. It seems people often pack their luggage in a more compact way instead of taking extra suitcases with them on vacation. In general, people bring a lot with them on vacation and they might not have packed their luggage the most effective way [Bureau of Transportation Statistics (RITA), 2012].

Airline	**First quarter**	**Second quarter**	**Third quarter**	**Total**
Delta	$197,971,000	$226,291,000	$232,508,000	$656,770,000
American	$137,210,000	$156,114,000	$152,750,000	$446,074,000
US Airways	$120,925,000	$134,752,000	$128,761,000	$384,439,000
Continental	$76,304,000	$91,332,000	$94,301,000	$261,937,000
United	$66,245,000	$71,111,000	$74,758,000	$212,114,000

Table 2.1: This table contains the top 5 of the companies which have collected the most fees regarding luggage in 2011. The numbers are from [Bureau of Transportation Statistics (RITA), 2012]. The fourth quarter is not yet released and is therefore not shown. This statistic is only regarding U.S. airlines.

An American website for statistics, [Bureau of Transportation Statistics (RITA), 2012], shows the amount of fees given regarding luggage which have been registered at the U.S. airlines. Table 2.1 is a segment of the statistic table found at [Bureau of Transportation Statistics (RITA), 2012]. Through this it is possible to see that there are people who exceed the set of limits given by the airlines. A note regarding this source is that the size of the fee is a combination of the different rules and the related fees. Therefore the statistics do not give an accurate image of the problem regarding weight limit, but a more general image of the problems with luggage exceeding the given limits.

Different Transport Methods

The problem with packing luggage is mostly the same if it is by train, car, or flights. Though flights are the one transportation where it plays the biggest role for the traveler because it has the biggest economical consequences and therefore the rules are more strict. With train and car it is more or less up to the traveler how heavy the luggage is to be. There are restrictions for how big the suitcases are allowed to be on the train. For some trains the limits are 100 x 60 x 30 cm [DSB].

When going on vacation, and using the car as transportation, the size of the car sets the limit in size of the luggage, since the passenger cannot pay to get more luggage with them than fits into their car. The weight might also have a influence on the amount of luggage there can be in the car, because the car might not be able to drive well if the certain car is over the excess weight, since there are cars which can lift more than others.

Souvenirs on Vacation

There can be many good experiences and memories on a vacations. The memories tend to be bound to photos, items and souvenirs which make it easier to remember. Souvenirs

can have a certain value and can be used to fill the home with memories about the past experiences. With this in mind, it is important to make room for possible souvenirs or other things that simply had to be bought while away.

These souvenirs can be a problem to bring home. As earlier mentioned the weight and size of the luggage is a problem before the departure, therefore it will normally also be a problem on the trip home. This means that if a family packs just to the limits and then buy things and souvenirs on their vacation they will get into trouble when packing the luggage for the trip home.

Packing as a Family

When traveling as a family it is an option to pack clothing, accessories and other relevant items in a more effective way. This is done by using all the suitcases in the most effective way and thereby not take the ownership of the suitcases into consideration. This method means that the suitcases will not be sectioned by ownership but the packed items will be packed into different suitcases despite the ownership of the item to achieve the most effective way of packing. Original suitcases tend to be packed so the suitcases are sectioned by the owner of the items so it is easier to find the desired item. This method is mostly used because it makes it much easier to know where to find the desired item. When this method of packing is used, it will be more effectively packed, but it has to be kept in mind that the weight should be distributed evenly, so none of the suitcases are too heavily packed compared to the others.

Summing Up

So the problem is packing the luggage in the most effective way and spreading the weight in the available suitcases, without violating the different rules or limits to size and weight. It can also be a problem to pack the suitcases for the trip home due to the bought souvenirs or other items. The reason for the problem could be that people find it difficult to pack the luggage for a trip and therefore pack more than they actually need on the trip. When traveling as a family, it is more optimal to use all the family's suitcases, to fit all the items.

The consequences of luggage exceeding the weight limits at the airlines are that the traveler will have to pay a fee for the extra weight. Train passengers who exceed the size limits might not be allowed to have their luggage with them if the dimensions of the suitcases are to big.

2.2 Luggage Rules

This section will focus on the general rules regarding luggage when going abroad. This is to give a good idea about the different restrictions that can be encountered when using commercial transportation.

2.2.1 Luggage Table

Given below is a table (see Table 2.2) which displays the various limits for luggage in different transportations. The table will be used to give a general overview of the different limits for luggage for different types of transportation.

Type of luggage	Dimension limit	Weight limit
Check-in luggage(Airplane)		20-23 kg
Carry on(Airplane)	50-55 x 40 x 18-25 cm	5-8 kg
Luggage(Train)	100 x 60 x 30 cm	Within reason *
Check-in luggage (Cruise)	75 x 50 x 29 cm	30 kg
Hand Luggage (Cruise)	55 x 35 x 25 cm	Within reason *

Table 2.2: This table displays a summary of the different rules given below.
* There are no set limits, it just has to be portable and not be a bother for other passengers
Source for airline [Airline]. Source for train [Indian Railways]. Source for Cruise [MSC-cruise].

2.2.2 Charter Trips on Air Planes

Given below is the various restrictions when traveling by plane.

Checked-in Luggage

Check-in luggage is the luggage which will go in the planes' cargo hold.
Items not allowed:

- Explosives, including detonators, fuses, grenades, mines and explosive compounds
- Gasses, propane, butane
- Flammable liquids, including petrol, methanol
- Flammable solid matter and reactive, including magnesium, matches, fireworks, flares
- Oxidising and oxidised compounds and organic peroxides, including bleach, auto repair-kits.
- Toxic or contagious compounds, including rat poison or infected blood.
- Radioactive materials, including medical isotopes and isotopes for industrial use
- Corrosive compounds, including quicksilver, car batteries.
- Compounds from combustible systems, which have contained fuel.

Due to the volatile or dangerous nature of the items listed above, they have been deemed unsafe and thus not allowed on the plane without explicit permission from the airline [CPH].

Carry On

Carry on luggage is what the passenger is allowed to bring aboard in the cabin.
Special rules for carry on luggage:

- Liquids, perfume, gel and spray – max. 100 ml – equal to one deciliter pr. container. You are only allow to bring these containers (bottles, cans, tubes and, so on), if they are contained in a transparent plastic bag, which has to be closed (1 liter bag per passenger). The bag has to be resealable.

- Past security, wares can be purchased (including spirits, perfume and other liquids). Wares are handed out in sealed bags, these bags may only be opened after the final destination has been reached.

- It is now a requirement that you take off your overcoat, take laptops and other larger electronic devices out of the bag before the security check-in. [Airlines]

2.2.3 Rules on Trains

There are different rules depending on which train company you are using.

The Danish train company, DSB, has very few rules regarding the luggage the passenger are allowed to bring with them.

The only rule is that the passenger's luggage needs to be able to lie on the luggage rack or under the seat. The luggage must not be bothering other passengers or putting any other person on the train in danger [DSB].

Another example could be Indian Railways where the luggage is allowed to have different weight depending on which class the passengers travel. They have no other rules regarding luggage [Indian Railways].

2.2.4 Rules on Cruise Ships

On board a cruise ship the "rules" are not really rules, more like guidelines as they encourage the passengers to not exceed the limits. Furthermore, the passenger's luggage should be kept in their cabins during the trip [MSCcruise].

2.2.5 Summing Up

Based on the information above it can be concluded that traveling by plane is by far the strictest of the transportations described. Packing a suitcase for this type of trip posses the greatest headache for the travelers, because they have to take so many factors into account, so they do not violate the rules. And as it can be seen from Table 2.1 a lot of travelers do seem to have a problem following these rules. But as it can be seen on Table 2.2 it can be difficult to make a general rule to follow as commercial transportation all have different rules regarding luggage.

2.3 Luggage Allowance

Due to the hijacking and crashing of the airplanes into the World Trade Center on the 11th of September 2001, the airport security has increased dramatically. Some of the hijackers carried knives and box cutters and this led to an immediate restriction of any and all types of sharp objects. The reason the hijackers could get these weapons on board the plane, was lax security around for instance Swiss army knives and blades like a box cutter. Along with stricter rules for items allowed on the plane, a thorough check up of the security personnel hired by the airport has been issued. After the change, airports are no

longer allowed to hire their own security personnel due to a lack of discipline and training and in some cases hiring of personnel with a criminal background [Zielbauer, 2001].

On the 5th of October 2006 more regulations were introduced to prevent passengers from bringing liquids of too large a quantity on board (see section 2.2). To construct a bomb a certain amount of "liquid" is required for it to have enough power to be a threat, and studies have showed that several 100 milliliter containers stored in a 1 liter bag equals around 500 milliliters of liquids which in turn is not enough to make a bomb that can take down a plane. This restriction covers all types of liquids because the screening points at the security can not distinguish one liquid from another without the security personnel manually checking the various liquids, which would severely slowdown the whole process [EU-Kommisionen].

Due to these restrictions packing a bag is not as simple as it used to be. A lot of items are no longer allowed and thus it can be difficult to know what is allowed and what is not. As the restriction covers all sorts of liquids packing a simple toilet bag is time consuming.

2.4 Solutions on the Market

This section is used to research the market of packing program and thereby get an image of which solutions already exists on the market. Through the research it is also possible to determine, how the existing solutions help the users with the stated problem. Looking at existing solutions can be used to determine which features would be needed in the program for this project, and from these solutions get inspiration. The amount of lists and guides on the market is huge. These lists and guides offer help and provide tips for packing when traveling. Some of these lists and guides have been developed into solutions that are available for the customer to use.

There also exists programs, which have integrated algorithms to handle optimization of the packing, on the market that can be used.

2.4.1 App - Packing Pro

Packing pro is an app developed for the Iphone and Ipad that offers templates for check lists to the customer. Packing Pro uses a touch interface, which means that the user by using the finger can navigate.

Figure 2.1: This is a picture of 2 menus from the Packing Pro app [Top iPhone Application, 2011].

Packing pro is designed with a panel in the bottom of the screen which lets the user navigate through the menus. Packing Pro provides the user with a help menu that contains information on how to use the app. On Figure 2.1 an example of how a check list can look like is seen.

These templates are designed for different purposes regarding the customers gender, type of trip, and purpose of the trip. The customer can then load the wanted template for the given purpose. The users also have the possibility to create their own lists by adding things that should be remembered for the trip by themselves, The user can select an existing list and delete the objects which were found irrelevant by the user. The user can then check objects on the lists off as they get packed. Packing Pro is a management tool that helps the customer get an overview of all the things to remember. As the name implies(pro) the app has to be bought before it can be used [Top iPhone Application, 2011].

Packing Pro works as a checklist and helps the user remember what to pack, but it does not perform any packing of luggage content itself. So Packing Pro itself does not solve several of the found problems, for example people packing too heavy bags, but helps the user remember what to pack. So a feature to consider from this program is the check list function that gives the user an overview of things to pack. A function that would not be needed is the compatibility with iPhone/iPad operating system. It would be nice if the program made works cross platforms but it is not required to solve the problem.

2.4.2 Online Check/Tip List

The online check list works as a reminder when packing luggage. It also gives tips and tricks which could be considered when packing for the trip. There exists a lot of different websites offering this service for free. Some are posted by an organization and others by

a person on a forum. This means that all electronic devices such as computers, tablets, and smartphones which have access to the Internet can open the website address.

An example of this kind of website is the following source [Foster]. This website offers a list of 10 tips that can be helpful for the customers when they are packing for a trip. The website is purely text based and helps the user packing through the tips on the website. The site does not help with the actual pack, instead it helps with the planning of materials that the user might want to have on the trip. The website is designed with a menu on the left that lets the users navigate through the different content on the website.

The online check/tip list in itself does not give the customer a solution to the packing problem. The websites instead help the customer plan the trip. The type of check/tip list used on [Foster] does not apply as a useful feature that could be used in the final program.

2.4.3 The E-Commerce Shipping Calculator

The e-Commerce shipping calculator is an advanced program which helps the customer pack large containers and calculates the price of the shipment. By typing the size, weight, location, and destination of the items which should be shipped, the program can calculate what the prize is going to be and generates a 3D-model of the container where the given items are placed in the best possible way so there is a minimum of wasted space. On their website [SolvingMaze] they offer a demo of their program. Their demo is web browser based and thereby should be accessible from computers connected to the internet. The demo is designed to have the dimensions of the container and the weight limit as input fields. Under the container there is a list of items where each item can have different dimensions and weights. To the right of this field the 3D-model is placed which will be generated.

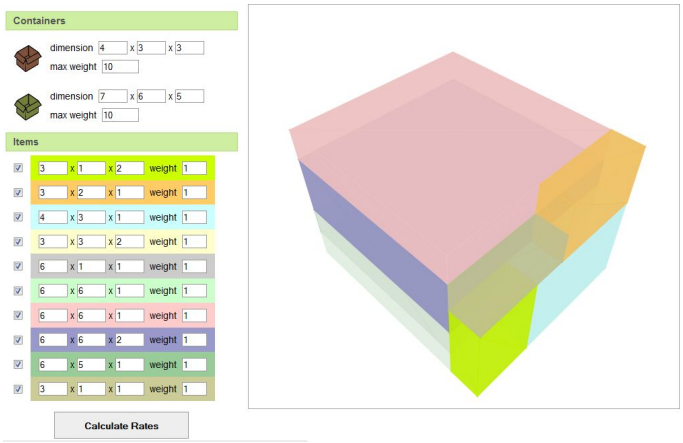

Figure 2.2: Screenshot of the demo running taken from [SolvingMaze]

The customer has to type all known data and press "Calculate Rates" and the program will then form a 3D-model, this can be seen on Figure 2.2. This product has a number of useful features which can be used in the final product. This solution can take an dimensions of the item and the weight and calculate the most optimal placement in the container. This can be related to packing your luggage for a trip.

2.4.4 Summing Up

The main object of this section is to look at the wanted features and recap them. Packing Pro provides the user with a sort of check list that can be checked off and thereby help the customer remember what has not been packed yet. Packing Pro allows the user to edit the check list which is needed if the user should be able to make a personal list.

This app does not help the user arrange the luggage content or take in consideration the size and weight of it. Thereby the app does not help people with the packing itself but rather which items should be remembered for the trip. The online check/tip list provide the user with advice for the trip and what to pack. Advice is great to get a general idea of what to pack but it still does not give a more effective way to pack. The e-Commerce shipping calculator is the solution with the most wanted features. One of the strong features which can be used is the ability to calculate the most effective way a container should be packed. An important side note is that the intentions is not to pack suitcases but the feature can be related to packing contents of a suitcase.

Included in product	**Solutions**	App - Packing / Packing Pro	Online check/tip list	The e-Commerce shipping calculator
Guide the user*		x	x	x
Distribute weight				x
Distribute space				x
Packing of items				x
Packing list		x		

Table 2.3: Table for the different products on the market and their features.
*The program should be able to guide the user through the different steps of the program.

Table 2.3 consists of features listed vertically and existing solution on the market listed horizontally. The crosses indicates when the particular product has the particular feature. The purpose of the table is to give an overview of the products and their features which were found essential to the problem. Table 2.3 shows that there are a lot of help regarding what to bring, but when it comes to packing it is only one of the selected solution that had this feature.

2.5 Problem Statement

In this section a problem statement will be formulated which will be used for developing a solution to the problem of this project. The problem statement is used to get a more

precise problem to work with. The solution is used in order to try solving the problem stated in the problem statement.

The problem analysis shows that there are two general types of programs on the market. One type is a form of packing list which tells the users what they might want to pack, the other type of program is a packing program which packs containers for the users and calculates the shipping cost to a designated location. These two types of programs only fulfill, parts of the criteria this project has put forward. With this in mind a problem statement has been formulated to help shape the solution for this project's problem.

- **How can a program be developed which helps the customer through the process of packing one or more suitcases the most effective way by size and weight?**

The meaning of this problem statement is to research and develop a program that in some way could handle the problem, but the user also plays a role in the problem. Therefore, the user must also be taken into account when the program is being developed. The reason for this is to make the program as user friendly as possible.

2.5.1 Sub Statements

The sub statements have been made to help find a solution to the problem statement. These sub statements describe some of the steps which need to be made in order to find the solution for the problem statement.

- **How can it be checked that the weight in the suitcases is evenly distributed and does not exceed the allowed weight or the volume in the bag?**

  The program will need to handle and solve calculations with weight and volume. Through these algorithms the program should find the most effective solution for the given data. It will also have to check that the weight of the solutions does not exceed the given limits or the volume of the luggage as found out in Section 2.2.

- **Which algorithms are needed to get the program to compute the most effective way to fit the items into the suitcases?**

  Some functions are needed for the program to find the most effective solution. The functions will use an algorithm for optimizing and thereby the program will fulfill its purpose.

- **How should the program communicate with the user and inform where the items are placed?**

  The program will have to be developed and tested so the customer can use the program to its full extent. Therefore it is important not to use advanced technical language or unexplained abbreviations.

From this problem statement there will be developed a list of requirements for the program, which it will need, to solve the given problem and be user friendly. The system requirements can be seen on section 4.1.

2.6 Method

The structure of this project will be based on Aalborg PBL (problem based learning). The Aalborg PBL is a method whereby the learning process lies in investigating a problem and trying to develop a solution for the given problem. The Aalborg PBL method also trains the students' ability to work together in a project group and gives them tools to handle the processes which go with working in a group.

The first stage of the project is the problem analysis in chapter 2, which purpose is to find and document that there is a problem to begin with. From the problem analysis a problem statement is formed and is used to produce a list of product requirements. The requirements are then used to designing and developing a product that should solve the problem stated in the problem statement. The design will be described in chapter 4. The development will also have its own chapter where the program and how the different functions are made will be described. This can be seen in chapter 5.

The testing phase will be described in chapter 6. The result of the testing will lead to improvements and a conclusion of the project. The conclusion will sum up the project and try to answer the problem statement. The conclusion can be seen in chapter 8. This is the main course of the project, when using the Aalborg PBL model. This project form is used because it finds and documents a problem and then, through the work with the problem, gives an estimated solution to the problem.

To document the problem, a lot of information is needed. The information is found through different sources such as; books, articles, websites, etc. When using information found through the Internet or other sources it is important to evaluate the used sources. This is done to filter out unreliable sources and thereby achieve a better and more trustworthy project. This process of evaluation is also known as source criticism and is generally used when using other peoples materials as documentation. Therefore, it is also a relevant method when using sources in the project work.

Theory 3

In this chapter a closer look will be taken on the theoretical aspect of writing a sorting program. With a good grasp on the different algorithms and the most effective way to pack items, the process of developing a program should be easier. A famous packing problem is the Knapsack problem, which is described in detail in section 3.1. A derivative of the Knapsack problem is the bin packing problem, which is described in section 3.2.

3.1 Knapsack Problems

The Knapsack problem is basically creating an algorithm which packs a list of items into a knapsack. Each item is assigned a weight and a value. The total weight of the items must not exceed the maximum weight capacity. At the same time, the knapsack must be packed so the summarized values of the items are as high as possible. There exists a vast amount of derivatives of the Knapsack problem. For example the 0-1 Knapsack problem [Kellerer, 2004], which dictates that each item can only have the status 1 or 0, which equals packed or unpacked. This means that each item can only be packed once, where in the regular Knapsack problem, items can be packed multiple times to maximize the total value of the knapsack. A Knapsack problem can be formulated as the solution to the following linear integer programming formulation:

$$\text{Maximize} = \sum_{j=1}^{n} p_j x_j \tag{3.1}$$

Equation 3.1 means: Maximize the total value (p) of items (j) in knapsack. Total number of items (n), the optimal solution vector (x).

$$\text{subject to} = \sum_{j=1}^{n} w_j x_j \leq c, \tag{3.2}$$

$$x_j \epsilon \{0, 1\}, j = 1, ..., n. \tag{3.3}$$

Equation 3.2 means: The total weight (w) of items (j) may not exceed the knapsacks capacity (c). Total number of items (n), the optimal solution vector (x).

The set seen in equation 3.3, denotes the optimal solution set.

[Kellerer, 2004]
Another derivative is the bin packing problem, which will be described in the following section.

3.2 Bin Packing Problem

The problem consists of fitting objects of different sizes into bins of identical sizes [Rosen, 1991]. This could for example be fitting various packages into shipping containers. There are various approaches to solving the bin packing problem. Bin packing problem is focusing on bins instead of suitcases. The only difference is the size, but basically they are the same. Some of the popular methods will be described in the following section. To describe these packing algorithms, illustrations will be used. The illustrations only show how the packing algorithms work in one dimension - but it gives a basic understanding of the algorithms. Figure 3.1 is an illustration of the unpacked elements:

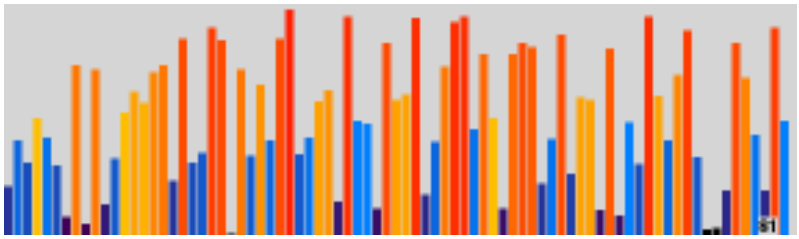

Figure 3.1: Initial elements (Source: [Arizona.edu]).

Given below are examples of different fitting methods used when packing bins.

3.2.1 First Fit (FF)

The First Fit algorithm creates a list of the objects needed to be fitted into bins. It then runs through the list, checking if an item can fit in each bin. If it cannot fit in the first bin, it will check if it can fit in the second bin and so on. If it does not fit in any bins, it opens a new bin, and fits the object there. Figure 3.2 is an illustration of the elements packed with the First Fit algorithm.

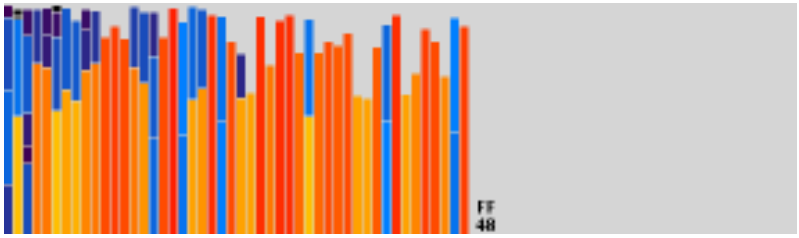

Figure 3.2: Elements after FF has been applied (Source: [Arizona.edu]).

3.2.2 Best Fit (BF)

The Best Fit algorithm is much the same as the first fit algorithm, except it does not fit the object into the first bin that can contain it, the algorithm compares it to each open bin, where the object fits. It will then place the object in the bin which will have the least

space left when the object is packed. Figure 3.3 is an illustration of the elements packed with the Best Fit algorithm.

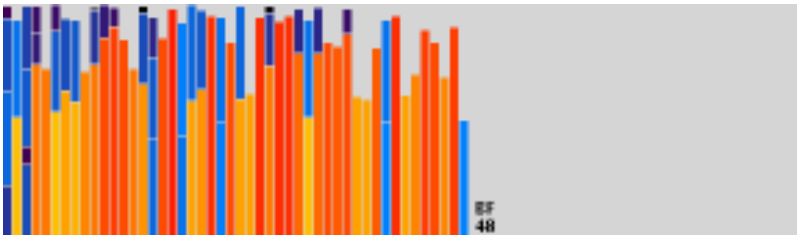

Figure 3.3: Elements after BF has been applied (Source: [Arizona.edu]).

3.2.3 Last Fit (LF)

This algorithm packs the object in the last open bin which has room for it. This algorithm is thereby the opposite of the First Fit algorithm. Figure 3.4 is an illustration of the elements packed with the Last Fit algorithm.

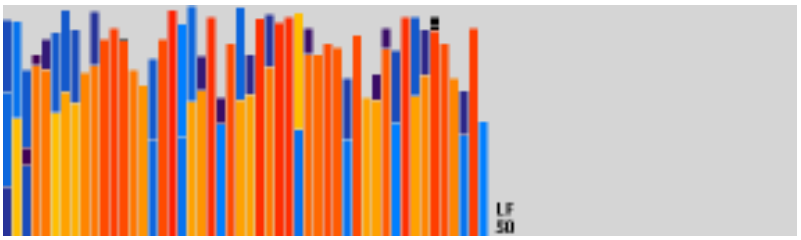

Figure 3.4: Elements after LF has been applied (Source: [Arizona.edu]).

3.2.4 Worst Fit (WF)

The algorithm checks all the bins, and packs the object in the bin which has most empty space. As its name suggests, this algorithm is the opposite of the Best Fit algorithm. Figure 3.5 is an illustration of the elements packed with the Worst Fit algorithm. As the figure shows, the worst fit algorithm is in fact more effective than its name might suggest.

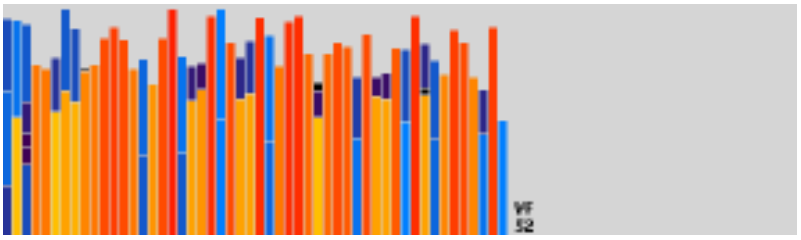

Figure 3.5: Elements after WF has been applied (Source: [Arizona.edu]).

3.2.5 Almost Worst Fit (AWF)

Similar to the Worst Fit algorithm, but the almost worst fit algorithm packs the object in the second most empty bin. Figure 3.6 is an illustration of the elements packed with the almost Worst Fit algorithm.

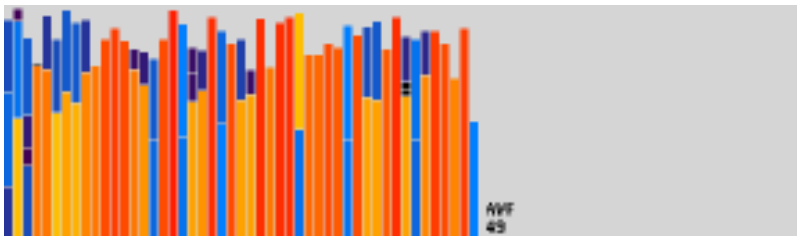

Figure 3.6: Elements after AWF has been applied (Source: [Arizona.edu]).

3.2.6 First Fit Decreasing(FFD)

The algorithms above are very ineffective because the biggest objects might be placed at the end of the list, and thus be packed in the end, where it is more effective to first pack these large objects. The First Fit Decreasing algorithm takes this into account and sorts the list before attempting to pack the items. This way the biggest items will be packed first.

3.2.7 Best Fit Decreasing(BFD)

Again this is the same as the Best Fit algorithm, but with the list being sorted before attempting to pack the objects. The list is sorted by size of the items, in decreasing order.

3.2.8 Summing Up

It seems that it is more effective to sort the lists by decreasing size before attempting to pack objects into bins. This way bigger objects are packed first, and the smaller objects can then be fitted around the bigger objects. However; in some situations it is necessary to use unsorted lists. For example in a factory with continuous production, it is never possible to have the complete list of objects, and thus never possible to sort the list. This theory can be used to form the algorithm of this project.

3.3 Comparison Between Knapsack Problem and Binpacking Problem

The main difference between the knapsack problem, in section 3.1, and the bin packing problem in Section 3.2, is that with the knapsack problem a list of items is given. Each item has a weight and a value, and must be fitted into a fixed size knapsack, so that the combined weight of the items is as low as possible, where the combined values must be as high as possible. In the bin packing problem, a list of items, each with given dimensions, each item must be packed into bins of a given capacity, so that a minimal number of bins are used. In this project the focus will be on the bin packing problem, because the goal is to pack items with given dimensions into suitcases (bins) of a given size. In other words, each item does not have a specific value - each item must be packed, which is why the bin packing problem has been chosen over the knapsack problem.

Design 4

In the design chapter the specification requirements, partner analysis, solution suggestions, program planning, sketches before programming, GUI description, and the packing algorithm are described and why those features are needed or wanted in the program. There is also a section detailing some features which would be nice to have if there is enough time to implement them in the program.

4.1 Specification Requirements

Through the problem analysis (see chapter 2) it has been documented that there are strict rules regarding some forms of public transportation when going on vacation, see section 2.2. Based on this research a list of features has been composed, which the program must fulfill to meet the base requirements, see section 2.5. Furthermore another list has also been composed of some additional features that would make the program better and more user friendly. They are not needed for the basic requirements, but rather as improvements to make the program even more ideal for the user.

Since it sometimes can be hard to use a new program, it is necessary to have a guide or some instructions for the users to help them use the program the way it was designed. The guide should be short and well formulated so the users can understand what to do in the program with ease. It has chosen to make it a requirement that the program should be written in C#, since this is the language taught this semester. From the problem analysis it was concluded that some people have a tendency to pack their suitcase(s) too heavily and then getting fined when using transportation with limits on how much the luggage is allowed to weigh, see section 2.1.1.

This has been solved in the program in two ways. Firstly, the program pack the suitcases using weight optimization - it will, if more suitcases are available, try to spread the items even between the suitcases. Secondly, a function must be made that makes it possible for the user to set a max weight for the luggage. This can help on the problem that airplane companies have a maximum weight limit for luggage and will fine people carrying too heavy luggage. The reason that the program is not programmed to just follow a standard rule about how much the suitcase(s) are allowed to weigh is because different companies have different rules as seen on Section 2.2.

The program also needs to pack the best way according to the size of the suitcase(s) and item(s) so as many items as possible will be packed in the program, and that the program should not place any items outside the suitcase(s). This helps solving the problem that some people have packing their suitcase(s) so everything will fit in the suitcase(s), which

is documented in Section 2.1.1. To make the program more user friendly a list of all the items to be packed should be included and be able to be edited so the user can get a better overview on what they are asking the program to pack.

To help the user through the process of packing the suitcase(s) the program needs to show the user where to put the items in the suitcase(s). This is a very essential feature in the program, since if the user is not able to see or get a description of where to put the items, the program does not in any way help the problem.

Another essential thing for the program is that the users are able to run the program and pack the items in the lists while away on a trip. This is needed if the user, while away, buys anything new and/or throws anything away. This can be solved by making a save/load function so the users are able to save their lists on the computer and load them later. This will also enable more than one user to use the same program without having to start all over again with putting in data every time.

There are some features that are not essential for the program to work but will improve the program. One of these features is to handle changeable shapes of items e.g. a T-shirt or other forms of clothing. This makes the program able to pack more efficiently. This means that the program can handle solid, liquid and bendable shapes. But this may not be in the program from the beginning since this is hard to develop and implement. To help get a better packing and plan ahead, the program needs a list of different trip types which can help the user pack the luggage for a given type of trip. Another optional feature to have is to allocate space for possible souvenirs the user might buy on the trip. These features mean that the user does not need to check if there is room for the souvenirs before buying it.

4.1.1 Targeted Features

Given below are the essential features that the program must have. In the earlier section where each part of the given problem has been named and from them these features have been made.

- **Guide the User:**

  The program has a small "read me" file, or another form of guide, which will tell the user how to use the program.

- **Distribute Weight:**

  The program must be able to distribute the weight of all the items between multiple suitcases if there are more than one suitcase.

- **Distribute Space:**

  The program also needs to distribute the items by space. The whole idea behind this program and project is to make a program able to pack a suitcase the best possible way with as little as possible wasted space.

- **On the Road:**

  The program needs to be able to repack the suitcases at any given time. Since the user might buy more or throw something away they will have the possibility to update the list and get the program to pack the suitcase(s) with the new item(s).

- **Baggage Rules:**

  The program needs to have a feature which allows the user to set a weight limit, so the bag will not extend the rules about how much the suitcase(s) are allowed to weight.

- **Structure of Packing:**

  The program needs a way of showing the user where to put all the items to pack.

- **Packing list:**

  To make it easier for the user to know what will be packed an editable list will be included.

- **Save/Load function:**

  To save time a function to save and load lists of items and suitcases will be needed. This makes the program easier and faster to use, since the user will not have to put in all the data about every item or suitcase every time he/she wants to use the program. It also enables more users to use the same program since they can save their own lists.

4.1.2 Optional Features

These features, as mentioned above, are additional features that might be able to be implemented later.

- **Solid/liquid/bendable shapes:**

  The program does also take in account that items might be bendable, and therefore fit in other ways than solid items. For instance a T-shirt can be folded in many ways and thus can be considered a liquid form as it can fit almost everywhere.

- **Type of trip:**

  Depending on the nature of the trip different packing lists will be necessary because each trip might require unique items.

- **Space for Souvenirs:**

  The program could pack the items in a way, so there will be allocated place for the souvenirs to buy on the trip.

The target features are the most important of these two lists because the target features play the biggest role in the problem solving. The optional features are for when all the target features are covered and there are still time left to work on the program instead of writing on the report.

4.2 Partner Analysis

This section is an analysis about who and what kind of groups could use a product such as the one this project is aiming to make. Instead of a specific target group this analysis will focus on cases in which someone might need this product.

- **People who have problems packing a suitcase properly:**
  If someone do not know how to effectively pack their own suitcase, be it how to distribute the weight of the items or how to utilize the space.

- **Families or groups of people who share or do not mind sharing suitcases when going on a trip together:**
  When traveling in a group there is an opportunity to share space amongst each other. A program that can calculate this would be ideal to better utilize this increase in bags and thereby an increase in bag space.

Based on this analysis the general functions and design of the product is created to fit the cases described best. This will be described in greater detail in section 4.6.

4.3 Solution suggestions

This section will focus on different solutions to the problem about packing a suitcase. Each difficulty of the solution, and the and benefits will be explained. The last subsection a solution will be selected and the choice explained. These solutions have been chosen based on the mind map that can be seen in appendix B.

4.3.1 Application for Smartphones

This idea is to make an application that helps the user pack one or more suitcases. The user will need to type in the height, length, depth, weight and name of the items to pack. The application will then calculate whether all the items can be packed. Because the weight of the calculations might be too much for the average smartphone, the application will need a server to make the calculations. This requires the customer to have access to the internet on the phone to be able to use the program. At the last step the application will show the user where to place each item. This means that the customer will have easy access to the program everywhere the user might bring the smartphone as long as it has access to the internet. On the other hand, people without a smartphone would not be able to use the program. This solution is easy to bring everywhere because you rarely leave your phone. One of the technical problems to this solution is that the programming language C# in itself cannot be used to program applications for smartphones, though with a plug-in the code can be translated to another language such as Java (which is the language for applications on Android phones).

4.3.2 Extension for an Existing Program

A second solution could be to make an extension for a program that already exists. This extension should add the missing functions of the original program. It could be an extension to the many packing lists. In that case the program should be able to use the information from the lists to calculate how to most efficiently pack all the items and afterwards show how to pack them. An example of this solution could be an extension to the e-Commerce Shipping calculator 2.4.3. This extension should be able to pack smaller items like a suitcase and not a container. A problem with making an extension for another program is if the other program is written in a programming language that is not compatible with a program written in C#. Another problem is writing the program if the other program is not open source and therefore the permission of the company could be needed before the coding can begin. If the existing program is not open source the permission of the company, and cooperation will be needed in the making of an extension

of their product. On the other hand it is possible to make a program that focuses more specifically on what is missing in the other program and therefore covers more of the important features. So this solution would need the permission of company to make use of their code if it is not open source, and it would have to be determined if the program would be able to work together with an extension written in C#. The solution would be able to cover more of the problem since the existing program would already have some of the features needed and the extension could cover even more.

4.3.3 Program for the Computer

A third solution could be a program for the computer. The user will need to supply the program with the height, length, depth, weight and name of the items to pack, and the program will then calculate if all the items can be packed and where in the suitcase. After the calculation the program will show where to put all the items, by showing where the individual items need to be on a 2D or 3D figure of the suitcase. Making a program for the computer means the customer will need to bring his computer on the trip if he wants to use it on-the-road, but it will not need Internet since the computer is strong enough to make all the calculation on its own. This solution requires some time to learn 3D editing if the displayed figures of the suitcase should be in 3D and this should be taken into account in the time schedule. This solution makes it is easy to test as the code will be self written and it does not need Internet to work, but on the other hand the customer needs to bring the computer if the person wants to use it on-the-road.

4.3.4 Choice of Solution

When understanding the three solutions it is possible to determine, which of the solutions solves the described problem best. To determine the best solution it is needed to look at the pros and cons of each solution. The pro about the first solution is that it is easy for the user to bring the program. The cons are that you will need Internet to use the program and this can be expensive on a vacation. Another con is that this solution requires time to learn of how to write an application for a smartphone. The pro about the extension for another program is that since some code is already written, there is more time to be more specific in what the original program is missing. On the other hand it can be a problem if the original program is written in a programming language which is not good at working together with a program written in C#. It is also a problem if the original program is not open source, because the company who owns the program then needs to give permission to use their code. If the original program is not open source it can be a problem to get this permission.

The last solution is a program for the computer. A pro about this solution is that it does not need Internet to run, since the computer is strong enough to make the calculations on its own. Another pro is that since no code from others is needed there is no problem about different programming languages needing to work together. It is also easier to make the program exactly as needed, since it is being made from scratch. A con is that some people do not want to bring their computer on a trip and can therefore not use the program while away. So when looking at all the solutions the choice is going to be the last solution about a program for the computer, since it is the solution with the strongest pros and fewest cons. It can be a problem on the go, but every solution has a problem "on the road" - the app will need internet access, and both the programs need a computer. Many people bring a computer on a vacation, so the problem is not that big.

4.4 The Planning of the Packing Algorithm

The algorithm used for packing items in this project is based on the known theory of the Bin Packing Problem (see section 3.2). The algorithm is a mix of elements from the "First Fit Decreasing" and the "Best Fit" method. These have been combined to make the algorithm used in this project. The algorithm is inspired by [Dube and Kanavathy]. The following sections will describe the used code. The algorithm will try to place all the items in the luggage so it is packed most optimally in both size and weight.

4.4.1 Optimization by Weight

The optimization of the weight is done, with the goal to prevent the luggage exceeding the weight limit if it could be distributed differently. Furthermore it would be preferable for the user, that the weight is evenly distributed among the luggage. To optimize the distribution of the weight, the average weight per luggage is calculated. It is calculated as seen on equation 4.1, where N = Number of items.

$$\text{Average Weight} = \frac{\sum_{i=1}^{N} Items[i]_{weight}}{N} \tag{4.1}$$

It is possible to distribute the weight evenly in the suitcases, when the optimal weight for each suitcase (the average weight calculated by equation 4.1) is known. The program will try to distribute the weight equally, but not if it will mean that the luggage cannot be packed. Therefore the weight distribution is an optimization goal, but not as important as the volume of the luggage. This part of the optimization is a Best Fit, because it finds the best placement in the luggage for an optimal weight distribution.

4.4.2 Description of the Algorithm

The algorithm uses the First Fit Decreasing (FFD) method to pack the items in the suitcases. This means that the algorithm at first sorts the items by size as described in Section 3.2. It will start by packing the largest items first, which will give a better result for the packing [Arizona.edu]. When the list is sorted it will then also sort the list of suitcases by size. The algorithm can now go to the actual packing.

The general algorithm is:

1. Sort the items by size.
2. Check if the item can be in the first suitcase, while total weight of the suitcase does not exceed the average weight per suitcase.
3. Check if the item can fit in the suitcase, else check the next.
4. If the item cannot be fitted in any suitcase, exclude it from the list, and notify the user.

This is a very general overview of the algorithm, and does not explain the process of the algorithm in sufficient details. To explain the algorithm, a flowchart can be seen on figure 4.1.

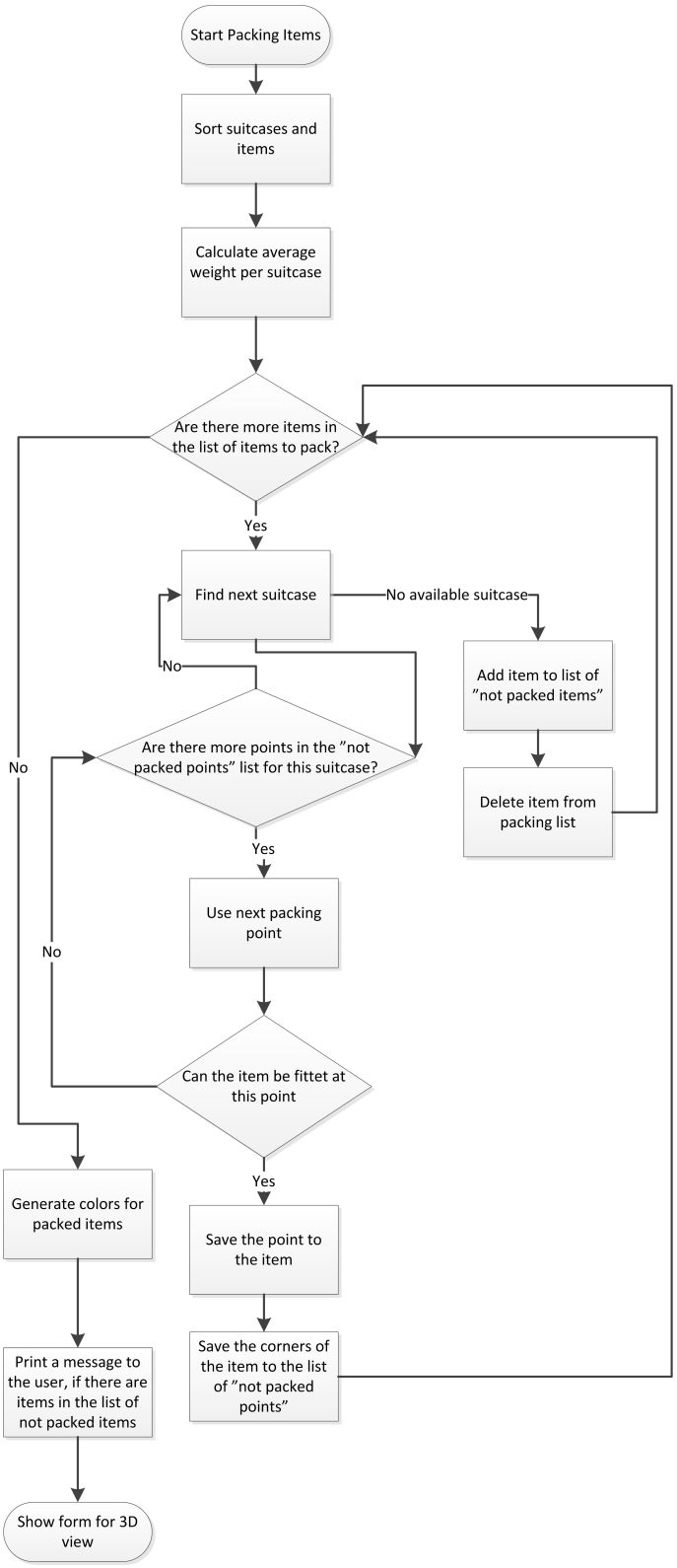

Figure 4.1: The flowchart for the packing algorithm

As seen on the flowchart 4.1, the first process is to sort the suitcases and items. They are sorted by size, so the biggest items are stored in the biggest suitcases. This will ensure the most effective way of packing, because it will fill the items, which are the hardest

to fit first. The next step for this algorithm is to call the function, which will calculate the average weight per suitcase for an even weight distribution. The algorithm can now start the actual packing of the items. It will run a loop, checking all the items in the list of items to pack. In the function "find next suitcase", the algorithm should find next available suitcase, checking for optimal weight distribution. To pack the item, each suitcase has a list of possible packing points, called "not packed points". This list should contain the points in the suitcase, where the next item can be fitted. This will ensure that the items are packed as effectively as possible.

The items will always be packed in a corner of another item, so there will be no leftover space between two items. This list of not packed points is individual for each suitcase, and if no item has been packed yet, the point [0;0;0] will be used. If the item cannot be fitted in the suitcase, it will try the next suitcase. In the case that the item cannot be fitted in any suitcase, it will add the item to the list of not packed items, and delete the item from the list of items to pack. This will make it possible to separate the items which are packed and those not packed. If the item could be packed the program will save the point to the item, so the program knows the position of the item further on. The corners of the item will be saved on the "Not packed points" list, and the next item will be packed. When all the items are packed, the algorithm will call the method to generate the colors for the items as seen in section 5.4. The items may have a color, but no item must be the same color as the item next to it, so they will not be mixed for the user. If some items could not be packed, they will be shown to the user, and the form to show the suitcase in 3D will appear as described in section 5.6.

4.4.3 Summing Up

This is the theory of the algorithm to pack the items in the suitcase. The algorithm is built from the most basic elements of the packing theories. This algorithm is the base for the packing program in this project, and the mix of First Fit Decreasing and Best Fit ensures an equal distribution of weight and space, and the "not packed points" list is used to make sure that as little space is wasted as possible.

4.5 Program Planning

The purpose of this section is to plan how the program should work and describe the flow of the program. The program is illustrated in the flowchart seen on fig 4.2 to give an overview of the whole program. A flowchart is a useful tool when programming because it explains the structure of the program.

To give a more precise explanation of a program, the flowchart can be formed into a pseudo code which is a level of abstraction above real code. A pseudo code is used as a schematic for the program and thereby gives some foresight into problems that can be encountered when writing the actual code. The pseudo code can be seen on Figure 4.3. Thus planning ahead and designing the program so a minimum amount of code errors and unexpected problems occur. The program planning will be used to make it easier to develop the program, and help make a better product in terms of structure.

When the program starts, it should show the main window. Here the user can load saved lists, manage the item list or the suitcase list, the instructions are shown, there is a button that will show information about the program and there is also a button that will start the packing of the suitcase. In the "manage" windows the user is able to clear the list, add new items, edit items, delete items, and save the list. If one or both of the lists

are empty when the user clicks the "Start packing"-button the program will inform the user that there needs to be at least one item and one suitcase for the program to be able to pack the suitcase(s) and item(s). After managing the list the user is asked to click the "Start packing"-button (see See section 5.1).

The program then runs the algorithms to place the items in the most efficient way regarding volume and weight. The program will also check that the suitcases do not exceed the weight limit set by the user. When the program successfully place an item, the item will be marked as packed. If the program cannot fit the item in any of the accessible suitcases the item will be marked as not packable. If the program reaches the point where all items have gone through the process, it should then inform the user that the process is done and inform how the user has to pack the suitcases and report if there were any items which could not be packed. At the end of the program the user will be able to see a 3D-view of the suitcase and be able to select which suitcase to show. The user will be able to zoom, rotate and drag the suitcase. A list will show all the items in the suitcase and if selected, they will be marked in the suitcase.

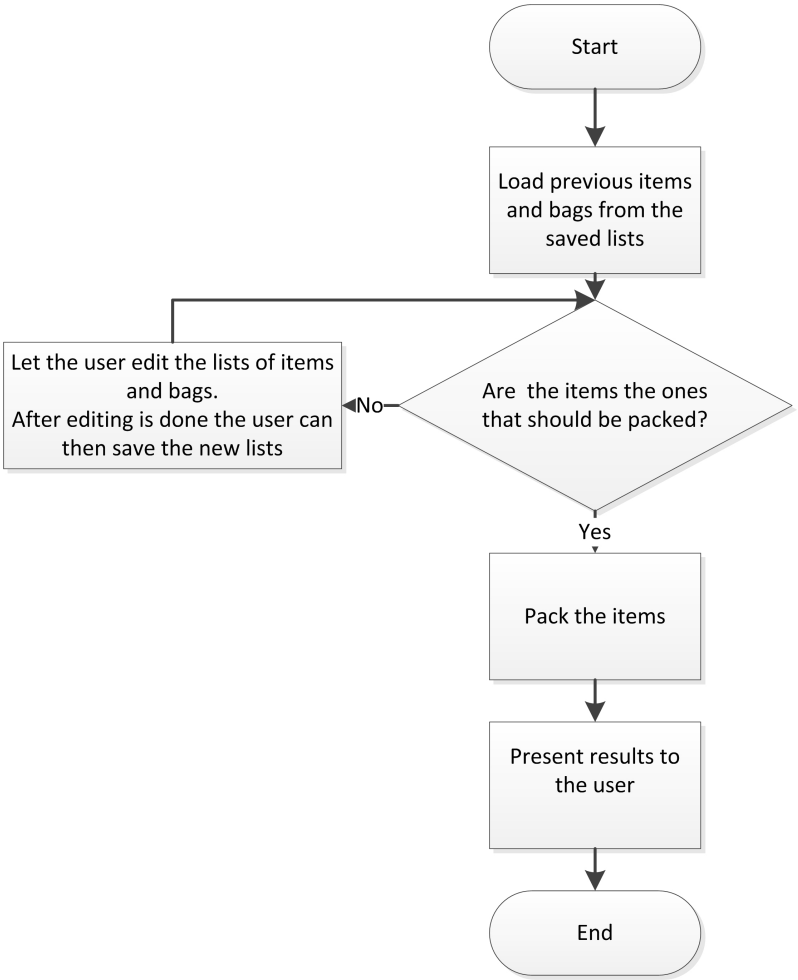

Figure 4.2: This is the flowchart of the program

Hereby the general structure of the program has been formed and can be described by a flowchart, seen on figure 4.2. The arrows show the direction of the flow in the program. Some of the arrows also have small labels indicating due answers to the decisions. This

flowchart can then be used as a schematic for the developing of the program and thereby a better structure of the program can be achieved.

A pseudo code has been made from the flowchart on Figure 4.2, which is one of the last steps before actually writing the program. This will give a more structured view of the events in the flowchart. The pseudo code can be seen on Figure 4.3.

Program start

If lists were used in previous start up:
- Load previous lists

Else
- Make blank lists

If the user wants to change the luggage:
- User selects list(s) of choice
- and/or
- edit selected list(s)

If there are items to pack:
- Pack the items
- Present 3D-model

End program

Figure 4.3: The pseudo code of the program

What happens in the pseudo-code on Figure fig:pseudocode, is first the program starts. The program will then check if there exists a previously saved list of items, and a saved list of suitcases. If these lists do not exist, the program should initiate a new, blank list. The user can also choose to load another previously saved list, or edit the loaded list. At last, the user can choose to pack the suitcase. The program will check if there are any items in the list of items, and if there is, pack the suitcase and present the 3D-model.

4.6 Sketches Before Programming of GUI

This section will describe the first thoughts of the Graphical User Interface (GUI). This is not how the final GUI is designed, but the first sketches. The final GUI was improved by tests and own experience. The final GUI can be seen in Section 5.1. Designing the program required some sketches and the first sketch looked like Figure 4.4. It shows some text boxes where the user must type the dimensions of the suitcase, so the program knows how much space it has to deal with, and that the measurements are in the metric unit system.

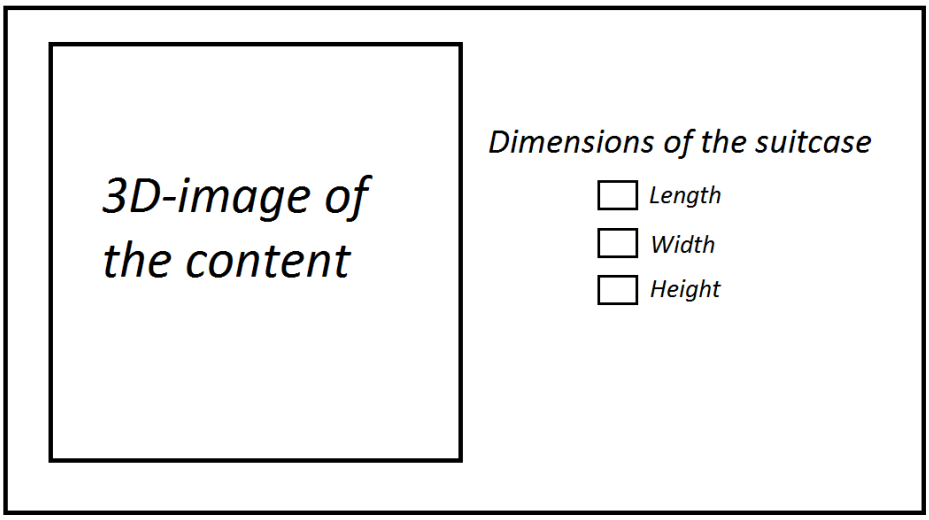

Figure 4.4: Initial sketch of the program

It has been discussed whether there should be a 3D image or a 2D image of the suitcase, where the user can see where the items are placed. It was decided that the best way to show the packing, is by a rotative 3D image, if possible. The point with the boxes was that the measurements of a specific item would be shown. A similar form would be made for items as well.

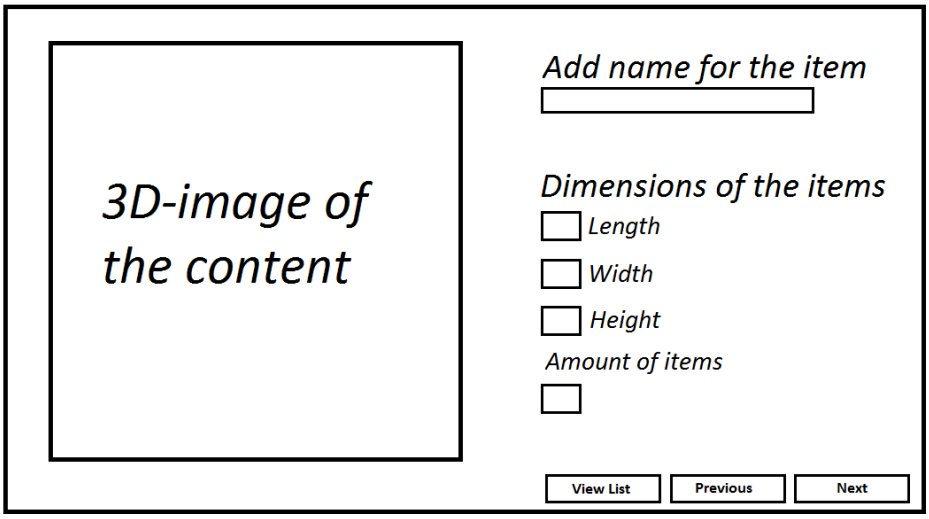

Figure 4.5: Second sketch, now with added control buttons

Figure 4.5 shows how the idea was before the coding of the program began. The idea was that the suitcase would be shown and then one item would be drawn at a time. The user would then be able to see where each item should be put in the suitcase according to the program.

The difference between this and the previous sketch is that now there is a list with

all the items. It is also now possible for the user to add names and amounts of specific items. The point was that the program would continuously update the 3D-viewer while each item was added to the list.

Figure 4.6: Sketch of the part where the user can add items

Figure 4.6 is an example of how the packing list should look like, and what functions it should contain. There is "Add More Items"-button and a "Start Packing"-button. This form is opening when the user presses "View Item List" in the previous form.

At last the final sketch for the GUI was decided. The 3D image viewer was replaced with the list of added items in the manage items and manage suitcases, because it gave a better overview of what the user already had added to the program. This can be seen on Figure 4.7.

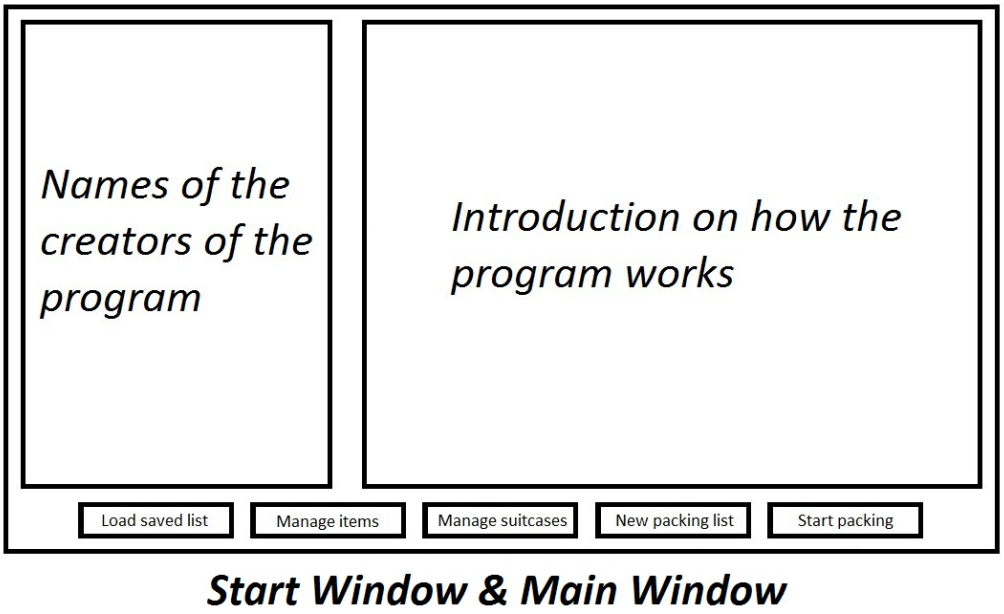

Figure 4.7: Sketch of the main window

What was changed from the first sketch of the main window was that more buttons were added. The two new buttons made it possible to load saved lists and start packing without needing to make changes to anything in the lists or making a new list. These changes were made to give more usability to the program.

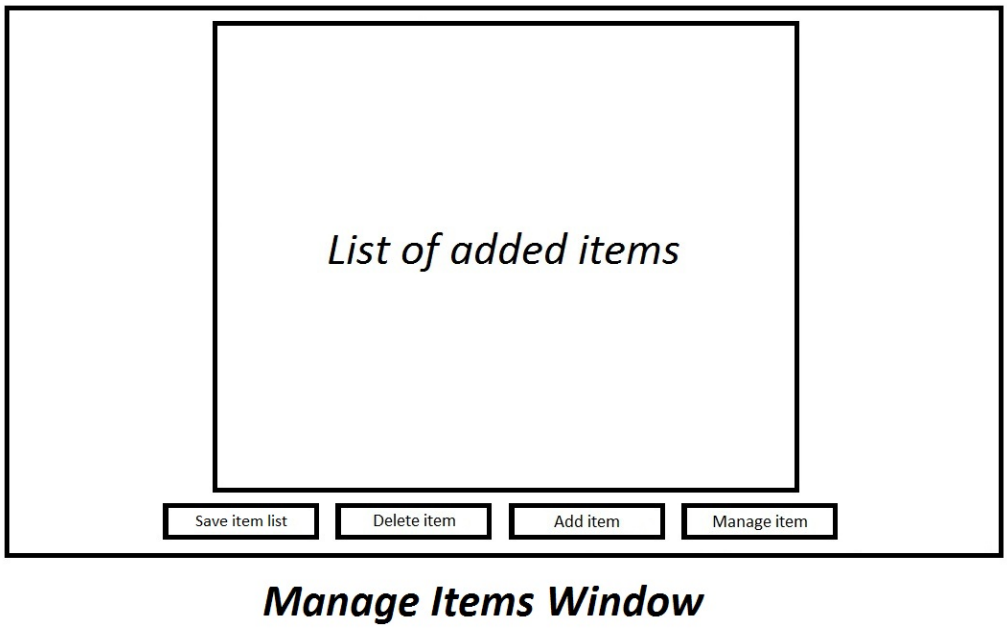

Figure 4.8: Updated version of the window where the user can manage items

Figure 4.8 is the sketch of the window which opens when the "Manage Items"-button at main form is clicked. The buttons are tools for adding, deleting or managing the items

on the list while the "Save item list"-button lets the user save the list so it can be used another time and returns them to the main window.

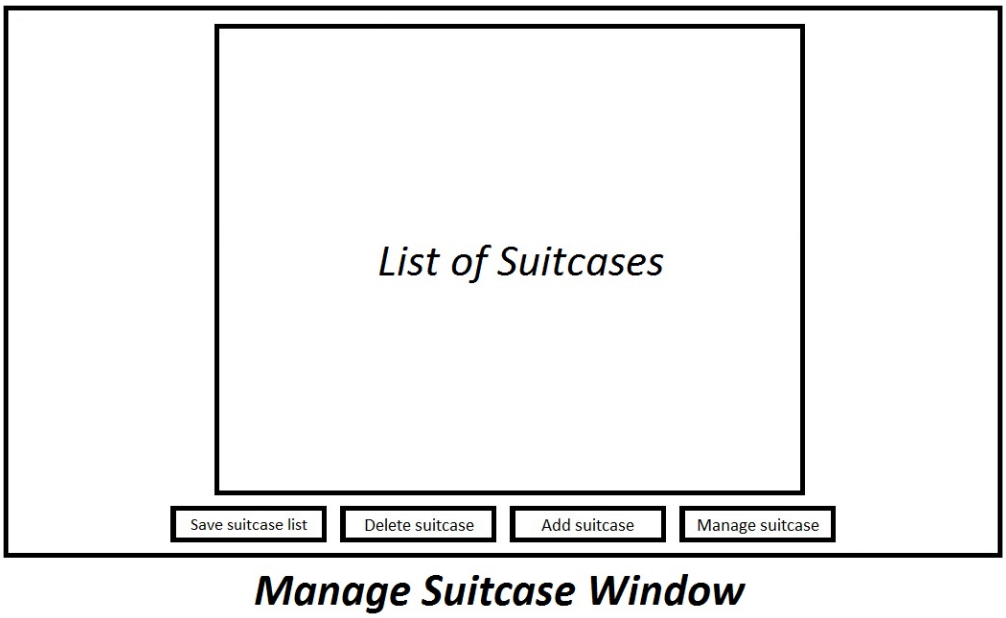

Figure 4.9: Window where the user can manage suitcases

The window seen on Figure 4.9 opens when the "Manage Suitcases"-button at the main form was clicked. The 3D-viewer has been replaced compared with Figure 4.4 with a list of all the added suitcases. When it comes to functionality the buttons do the same as those in the "Manage Items" window just with suitcases. Clicking the "Save Suitcase List"-button will let the user save the list and will return them to the main window.

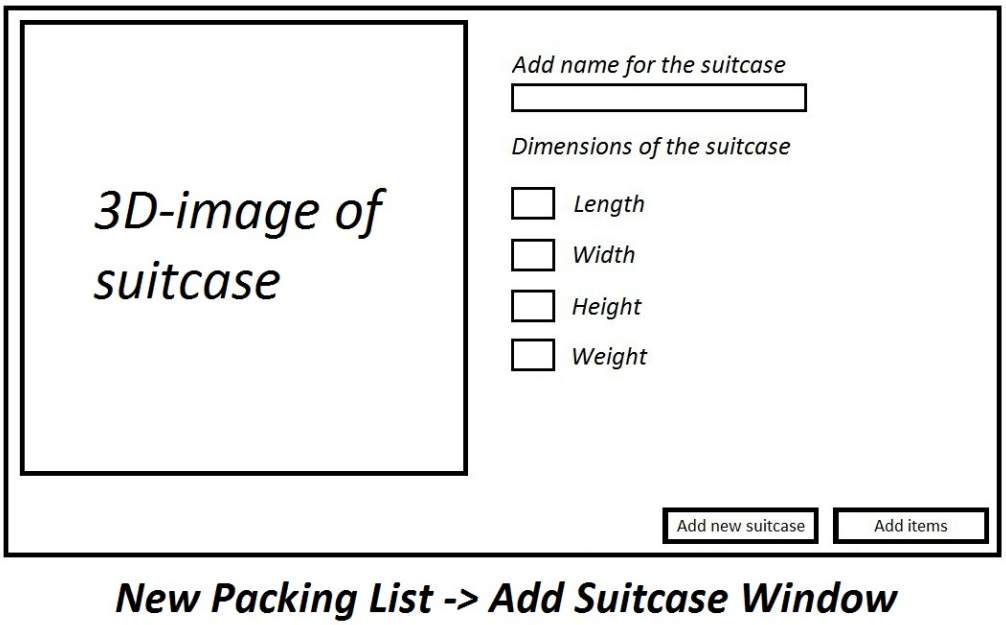

Figure 4.10: Sketch of the window which appears if the user clicks the "New packing list"-button

When clicking the "New packing list"-button the window seen on Figure 4.10 will be shown. Here you are able to add the suitcases, you want the program to pack. On the left side in the window the program will show a 3D picture of the suitcase. The "Add items"-button will send you to the next window.

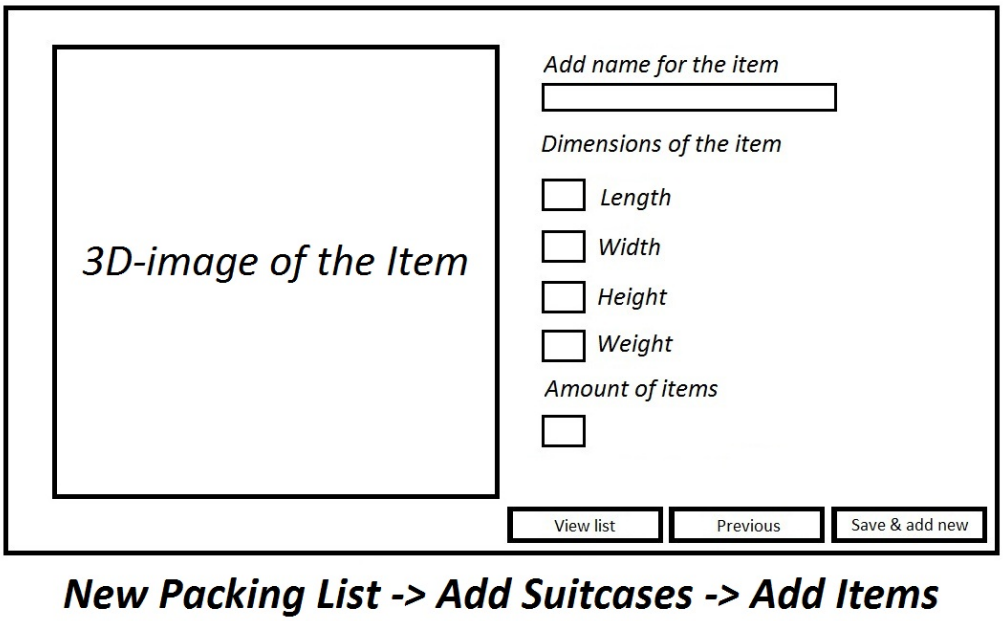

Figure 4.11: Window opens when the user adds new items

Figure 4.11 shows how the user can add the items he/she wants packed to a list. After the user has added all the items the "View list"-button will show a list with all the items and afterwards return the user to the main window.

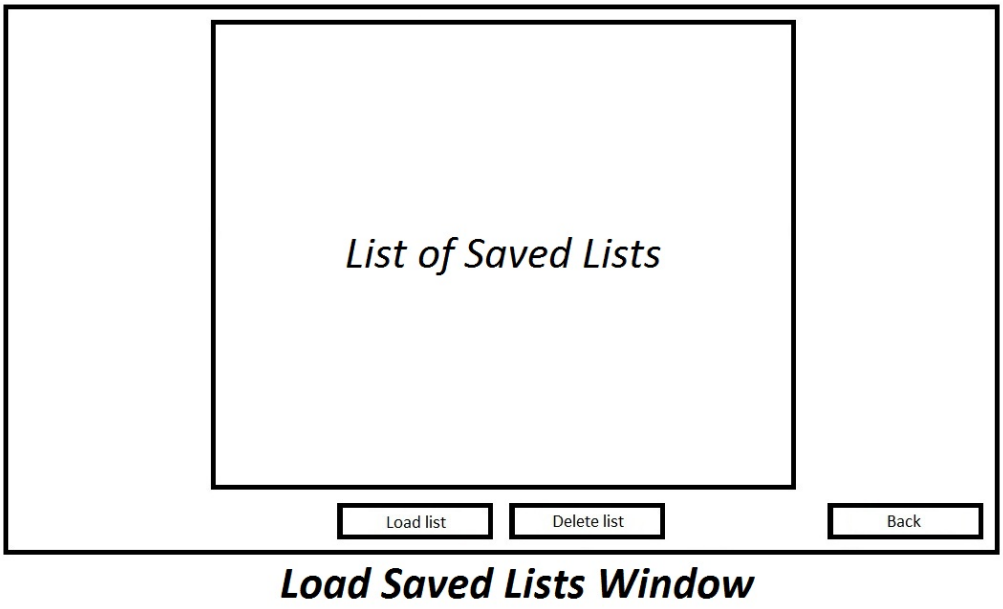

Figure 4.12: Sketch of a window which shows the saved lists

Figure 4.12 is the window which will be shown when the "Load saved list"-button is clicked. Here the user would be able to load a saved list and delete lists he/she would not want to save anymore. The "Back"-button will return the user to the main window.

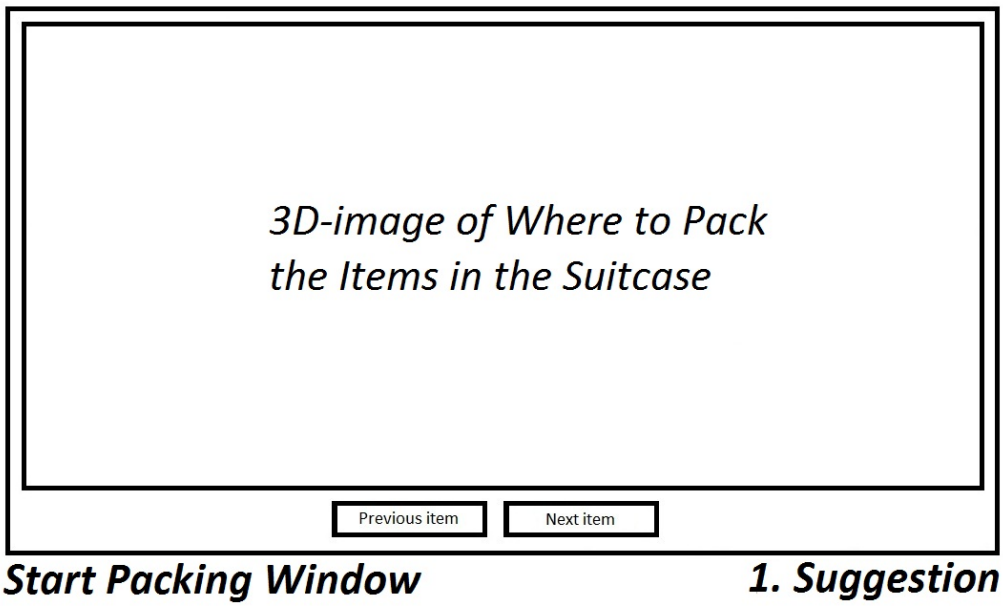

Figure 4.13: Sketch of the 3D-viewer of the program

Figure 4.13 is one of the suggestions on how the window showing where the items should be packed, could look like. Here the 3D-viewer shows the position of every item, one item at a time. The two buttons are used for going back and forth through the items.

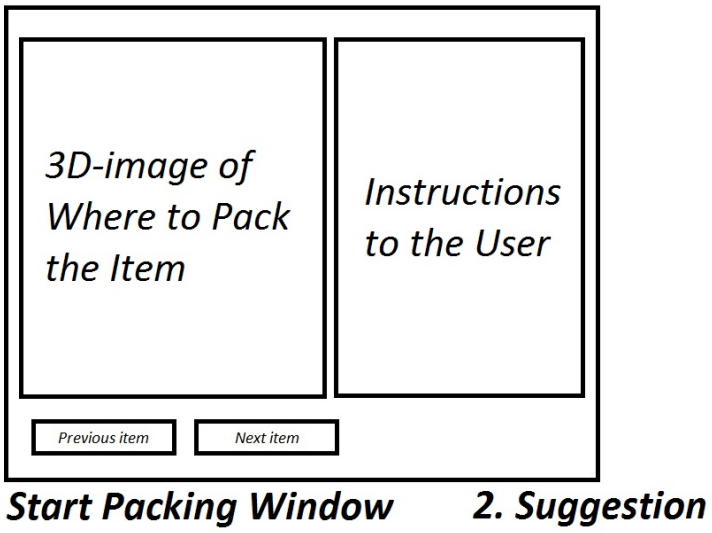

Figure 4.14: Sketch of the 3D-viewer with instructions added

Figure 4.14 is the second suggestion on how the window should look like. Here the

3D-viewer is smaller but on the other hand has a list of instructions on the side to better guide the user. The two buttons are used for going back and forth through the items.

Figure 4.15: Sketch of the message box which will be shown when the items have been packed

The message box seen on Figure 4.15 will be shown when all the items have been packed. The "Pack again"-button will show where to put all the things again. The "Make new list" will return the user to the main window, where he/she can start making a new list, load a saved list or manage the already saved list. Most of the sketches do not look like the program what so ever, this is because the sketches where made before the program, and when the program was developed, the GUI was adapted to be more logical because of some good feed back from the tests.

5 Development

In the following section the program will be described. Each of the most important functions is described in this chapter. The final GUI is also described.

5.1 GUI Description

The GUI(Graphical User Interface), as the name implies, is the interface the user interacts with when operating the program. The main interface window of the program looks like this, see Figure 5.1.

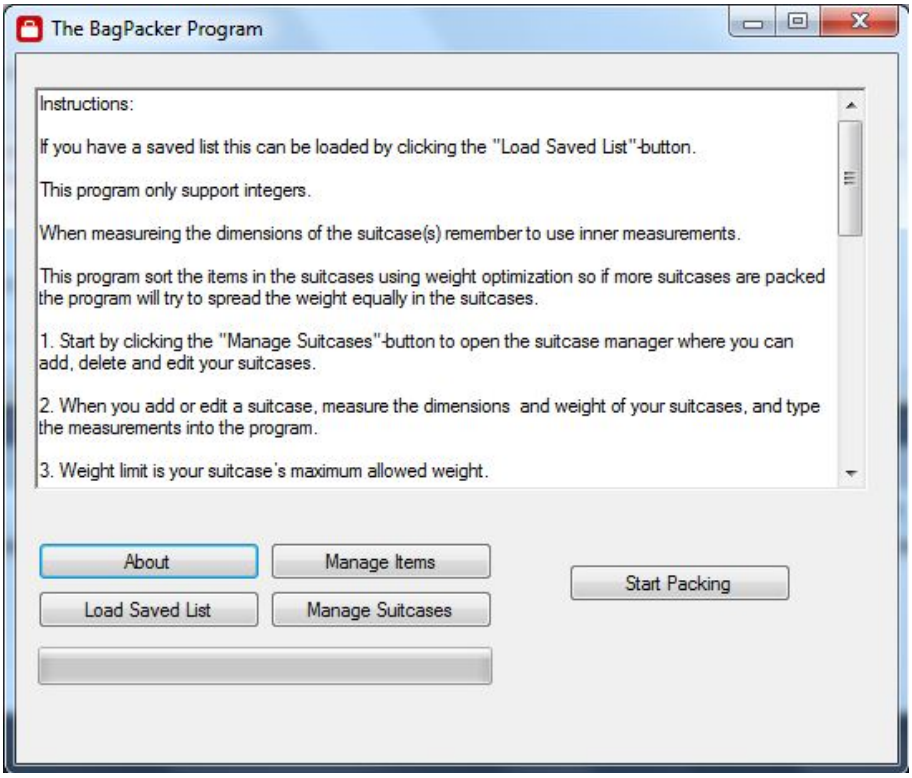

Figure 5.1: This is a screenshot from the main window in the program.

The main window contains the instructions on how to use the program, an "About"-button which tells who the creators are and what they do. A "Manage Items"-button and "Manage Suitcases"-button have been added where the user can add, edit, or delete

items/suitcases. The button "Load Saved List" can be used if the user has previously made a list and wants to use it and/or add/edit some items or suitcases from the list. The "Start Packing"-button initiates the algorithm of the program and packs the suitcase(s). The progress bar is associated with the "Start Packing"- button and starts when the user presses the button. The main window design varies from the sketch, see Figure 4.7. The one difference is that the buttons no longer are placed aligned in the bottom of the window but instead the buttons are placed as a group. This is done to achieve a more logic design and reduce the width of the window. Another difference is that the space reserved for the "name of the creators" have been made into an "About"-button. This is done because it is not necessary for the program to work, nor for the user to understand how the program works. The "introduction of how the programs works" has been kept because it helps the user use the program and it does not take up much space. The "New Packing List"-button has been removed because it was found irrelevant since the program is design so the user can edit the items / suitcases lists in other windows.

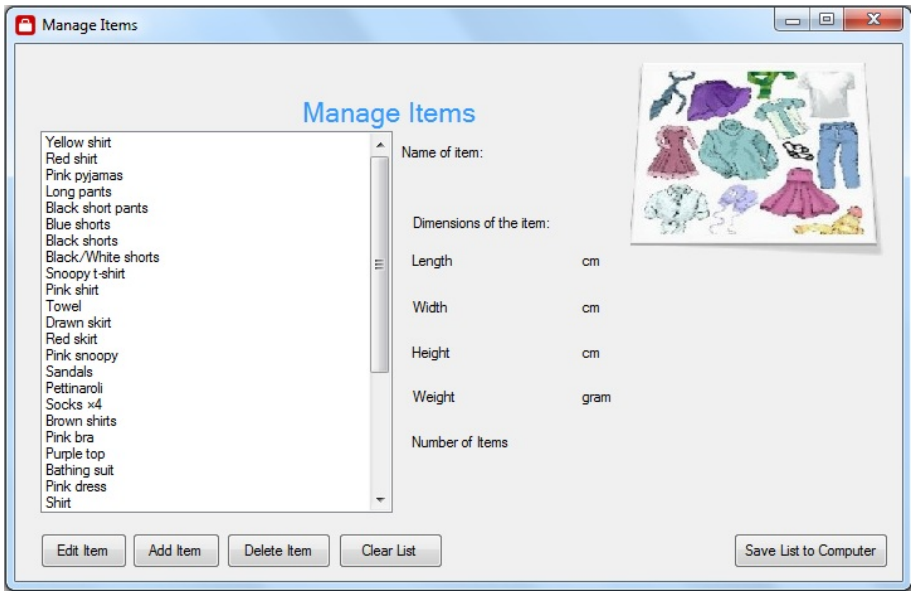

Figure 5.2: This is a screenshot from the manage items window in the program.

The Manage Items window is where the user can add, delete and/or edit items, see Figure 5.2. The difference between the actually design and sketch, see Figure 4.8, is that the "Item List" has been narrowed down and moved to the left of the window. This is done because the list itself does not need to be that wide. Another difference is that six labels which updated and shows the selected items data have been added. The labels are placed to the right of the list and have some name labels to go with them. The "Manage Item"-button has been renamed to "Edit Item" because it is a more describing name for the button. A "Clear List"-button has also been added to make it easier for the user to remove all the items. All the buttons also have different alignment and moved to the left.

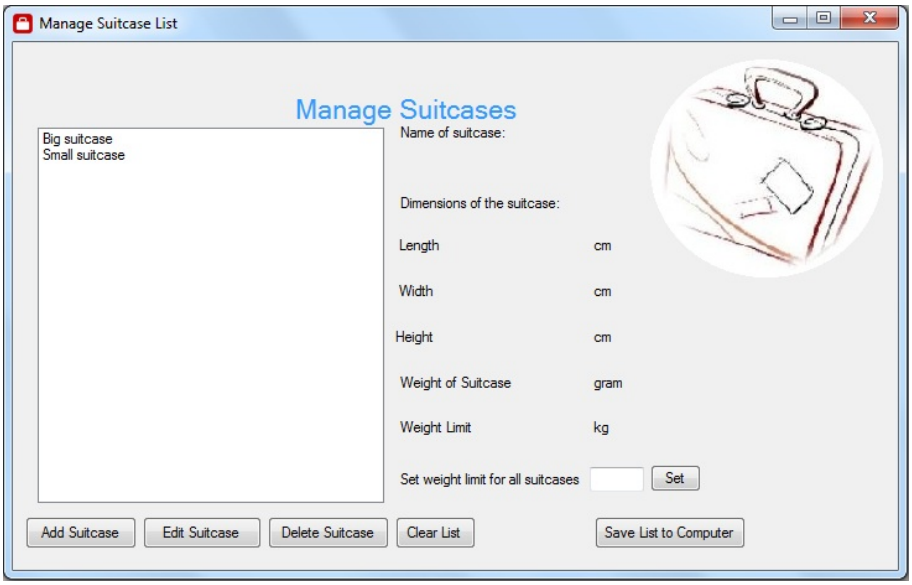

Figure 5.3: This is a screenshot from the manage suitcase window in the program.

Add, delete, or edit suitcases in the Manage Suitcase window, see Figure 5.3. The differences between the actual design and the sketch, see Figure 4.9, are almost the same as "ManageItems" just with other properties. There have also been added a text space where the user can set the weight limit for all the suitcases.

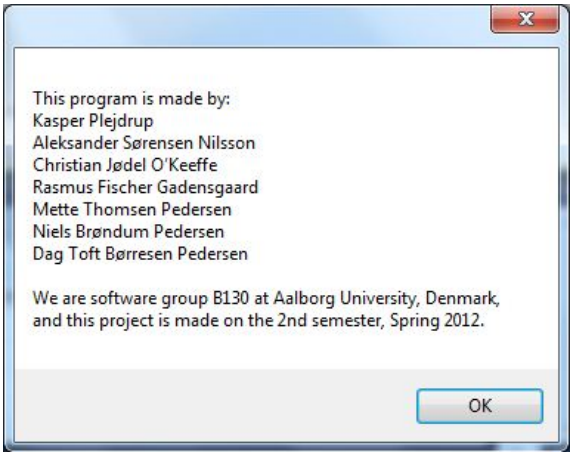

Figure 5.4: This is a screenshot from the about window in the program.

The "About"-button which tells the user who the programmers are, and when the program was made. This can be seen on Figure 5.4. The "About" form has been made so it is no longer in the main window but opens through the main window with a button. The reason has been mentioned earlier.

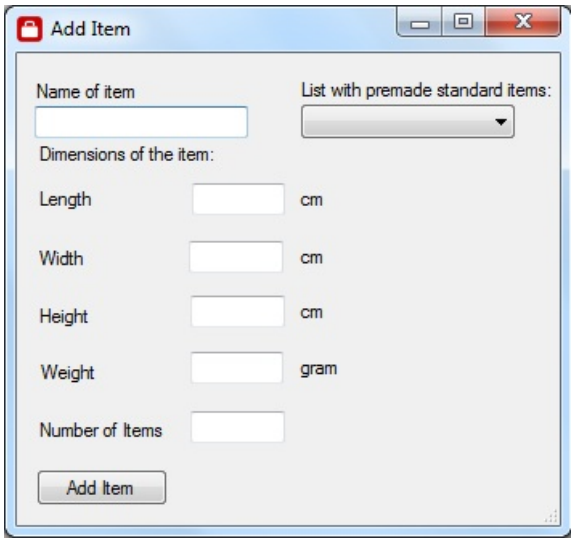

Figure 5.5: This is a screenshot from the add item window in the program.

When the user presses the "Add Item"-button in Manage Items, this form can be seen on Figure 5.5. In this form there is a text box, where the user types the name of the current item. There are 5 other text boxes which are for the length, width, height, weight, and number of items. The biggest difference between the design and the sketches, see Figure 4.11, is that the 3D image has been completely remove from this section of the program. Another change is that the form itself has become smaller because it was unnecessary to use so much space when the 3D image no longer is there. The "View List"-button is unnecessary because the user goes from the "Item List" to the "Add Item" form. The "Previous" and "Save & Add new" have been made into an "Add Item"-button. There has also been a drop down list where the user can select premade items to add. This is done to save time for the user when adding items to the list.

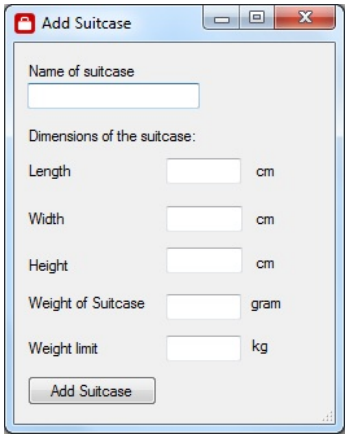

Figure 5.6: This is a screenshot from the add suitcase window in the program.

When the user presses the "Add Suitcase" button in Manage Suitcases, the user can add a new suitcase in a new form. The new form can be seen on Figure 5.6. The data needed is the length, width, height, weight, and the maximum weight of the suitcase. The changes between the design, see Figure 5.6, and the sketch, see Figure 4.10, are almost the same as with "Add Item" form, except the drop down list and some properties.

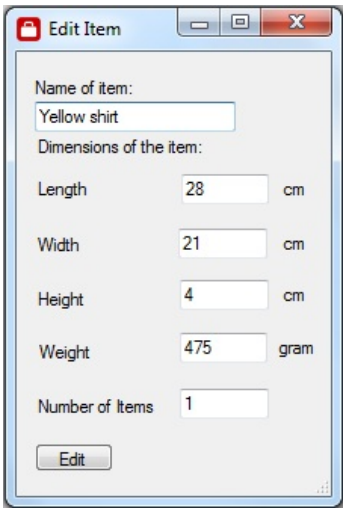

Figure 5.7: This is a screenshot from the edit item window in the program.

If the user wants to edit an item, he/she can press the "Edit Item"-button in Manage Items, see Figure 5.7. In the form there are 6 text boxes for each input parameter, and a button saying "Edit", which saves the changes the user has made and closes the form. This form has been added directly to the program without any sketches because of different program structure.

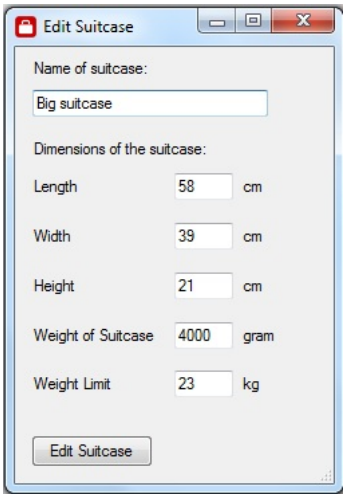

Figure 5.8: This is a screenshot from the edit suitcase window in the program.

In "Manage Suitcase", there is a button called "Edit Suitcase" which opens a new form. The new form can be seen on Figure 5.8. It allows the user to change the data of a suitcase, if e.g. the measurements are wrong, or the user wants to use another suitcase, which does not have the same measurements.

If the user already has used the program before and has saved an item list and a suitcase list, both can be loaded. This is done through the load button which activates the Windows standard load explore. This explore is then used to find the files. The load button can be seen on Figure 5.1 which is the main window.

The 3D-viewer shows how the program has packed the different items in the suitcases.

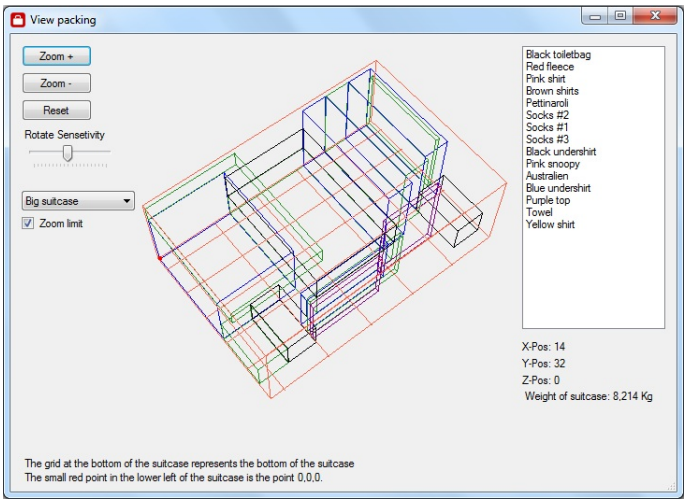

Figure 5.9: This is a screenshot from the 3D-viewer window in the program. The bottom of the suitcase is marked with a grid

This, see Figure 5.9, is the first thing the user will see when the packing is finished. It shows how the items are placed in the suitcase. The image can be dragged, moved, and zoomed with the mouse, as seen below. When the user clicks on an item in the list on the right side, the marked item will be highlighted in the image. Below the list are the x-, y-, z-points to see where the item is supposed to be placed, and the current weight of the suitcase.

To highlight an item in the suitcase the user selects the item from the item list. This is not a part of the sketch but the plan was that the user could use two buttons which can go through the list, back and forth.

On the left side of the window are three buttons: zoom in, zoom out and reset. On the lower left side of the window is a check box called "Zoom Limit". It sets the limit for how close and how far the user can zoom the image. The buttons have been made in the case the user does not have a mouse with a scrolling wheel or is using a laptop. The track bar on the left is a tool to adjust the speed, when the user rotates the image. The reset button resets the track bar. Below the reset button is a drop-down list containing the suitcases the user has packed items into.

The reason why it looks as it does is because it gives a good overview with the item list on the right, the 3D-image in the middle, and the image options (zoom in, zoom out, reset etc.) on the left. In the 3D-image a small box has been made in point(0;0;0) to indicate where it is, so the user easier can navigate through the items. The sketch of the 3D-viewer can be seen on Figure 4.13.

5.2 How the Program Handles Different Forms

Because the program has different forms with different functions, it is often necessary to parse variables from one form to another. This is done by creating a new instance of the form, and parsing some inputs to the form. Just like one would create a new object and parse inputs to a constructor. An example is the form that shows the 3D-drawing of the suitcase. This should be automatically opened when the packing algorithm has packed all the items in the suitcases. The exact trigger which opens the new form, is the

frm3D.ShowDialog(this); This can be seen on Listing 5.1.

```
1 //If the packing is done, show the 3D-Viewer
2 if (finish_packing)
3 {
4     //Parse the lists to the 3D-Viewer
5     frm3DViewer frm3D = new frm3DViewer(luggage_items_to_pack,
          luggages_to_pack, this);
6     //Timer stops when the packing is done
7     tmrCheckForFinishPacking.Enabled = false;
8     //Prepares for a new packing
9     finish_packing = false;
10    //Show the 3D-Viewer
11    frm3D.ShowDialog(this);
12
13 }
```

Listing 5.1: Open the 3D-viewer when all items are packed. Source: frmMain.cs

It is clear that three variables are parsed; the list of the items which has been packed, the list of the suitcases, and finally the "frmMain" form is also parsed to the "frm3DViewer" form. Naturally the "frm3DViewer" form needs to know the different items and suitcases. The reason why the "frmMain" form is parsed, is that the "frm3DViewer" form resets the progress bar when the "frm3DViewer" is closed. The tmrCheckForFinishPacking is a timer, which checks if the packing is finished in every call, and is disabled after packing.

Let's take a look at how the "frm3DViewer" receives these inputs. The inputs lie as input parameters in the "frm3DViewer" method within "frm3DViewer" as seen on listing 5.2

```
1 //The 3D-Viewer, recieves the list of suitcases, the list of items,
      and the frmMain form as inputs
2 public frm3DViewer(List<frmMain.luggage_item> input_lug_items,
      List<frmMain.luggage> input_lugs, frmMain input_form_main)
3 {
```

Listing 5.2: The input parameters of "frm3DViewer". Source: frm3DViewer.cs

5.3 The Development of The Packing Algorithm

The algorithm was developed from the theory described in section 4.4. The theory has been extended from the experiences when packing the program. These extensions have made the packing more effective, which results in more items being packed. To understand the different methods, the algorithm as described in section 4.4 must be extended a bit. The extension will be described in this section, with a summing up at the end. To describe the code, this section will be divided into different subsections to understand the different parts of the code.

5.3.1 Classes

To handle the information and functions needed to handle suitcases and items to pack, the classes luggage and luggage_item have been made. They do both inherit from the class Cube_Shape, with the variables width, depth, height and name. The Cube_Shape class provides the information of a cube, which both suitcases and items can be seen as.

The Class "luggage"

In the class "luggage" there are a lot of private fields (starting with "_") which can be accessed through properties. It contains all the values needed: the maximum weight, the weight of the suitcase etc. It also contains a set of methods which is used through the packing process. The properties which should only be accessed in the methods in the class has a private setter, so functions outside the class can only get the value, and not change the properties by mistake. An example of this is the property "weight", which indicates the total weight of the items in the suitcase and the weight of suitcase. This value should only be set by the class itself in the methods placing an item. In the class "luggage" there is a set of methods. These methods provide the necessary functions to handle the suitcase while packing.

The Class "luggage_item"

The class luggage_item is for the items to pack in the luggage. As in the luggage class, this class has some variables, which can only be accessed through properties. Those properties, which should only be set inside the class have a private setter. There is also a private method "Rotate_Lug_Item", which should only be called inside the class.

5.3.2 Description of the Code

The algorithm to pack has been developed from the flowchart seen on figure 4.1. The following will describe the code and the extensions of the flowchart.

Reset and Sorting

As seen on the Figure 4.1, the first thing which is done is to sort the suitcases and the items. Before sorting the lists of suitcases and luggage, it resets the values which were changed when packing. This is to ensure the algorithm has the right values if the users packs it twice.

The reset methods for the items and the suitcases are in their respective classes. A loop will call the reset methods in all suitcases and items in the lists. This is for example to reset the values of where there are saved items in the luggage. The items and luggage is also rotated so the luggage has the longest side as width. The rotate function can be seen on Listing 5.3. Through the three "if" statements, the program checks which of the values that are the biggest. In the first "if" statement the program checks if depth is bigger than width. If depth is bigger than width they are swapped around. In the second "if" statement the program checks if height is bigger than depth. If height is bigger than depth they are swapped around. The first "if" statement is then used again because depth could have gotten bigger than width. Through this process the width should have the biggest value by now. This is done, so the algorithm packs as optimally as possible. Likewise the items are rotated, so the height is the smallest side. This will ensure that the algorithm will try to pack items lying, and not upright.

```
1 //Rotate so width is the longest side
2 int temp;
3
4 //If the depth is higher than the width, switch the values
5 if (depth > width)
6 {
7     temp = depth;
8     depth = width;
```

```
 9     width = temp;
10 }
11
12 //If the height is higher than the depth, switch the values
13 if (height > depth)
14 {
15     temp = height;
16     height = depth;
17     depth = temp;
18 }
19
20 //If the depth is higher than the width, switch the values
21 if (depth > width)
22 {
23     temp = depth;
24     depth = width;
25     width = temp;
26 }
```

Listing 5.3: The code to rotate the luggage until the longest side is the width. The rotation is made by switching the values of 2 sides. (From the luggage class in the Main form.)

The method to sort the luggages and items will sort them by size, so the largest items will be attempted to be packed first in the biggest suitcases. This will give a better packing, because the smallest items are easier to fit than the big items.

Weight Distribution

The average distribution of weight in the suitcases is calculated before going in the loop of the packing algorithm. In the flowchart on Figure 4.1 it will "Find next suitcase" when packing an item. This is not as simple as shown in the algorithm. The next suitcase is found by not only size, but also weight. It will start with the largest suitcase and see if the weight of the suitcase, its content and the item to pack will exceed the average weight per suitcase. If it does, it will try to use the next suitcase. If the item cannot be fitted in any luggage using the average weight, it will try again with only the maximum weight of the luggage in mind, so an item can be packed even though it exceeds the average weight per suitcase. The code of this can be seen on Listing 5.4, where the stop_check_for_avg_weight variable checks if there should still be check for average weight. The lug_id variable is the index in the luggage list of the suitcase which is checked now. The lug_items_counter is the index of the item to pack. This loop will continue until the item is packed, or every possibility has been tried.

```
1 if (stop_check_for_avg_weight == true ||
2     (luggages_to_pack[lug_id].weight +
3      luggage_items_to_pack[lug_items_counter].weight <=
4      weight_per_luggage))
5 {
6     if (luggages_to_pack[lug_id].weight +
7         luggage_items_to_pack[lug_items_counter].weight <
8         luggages_to_pack[lug_id].max_weight)
9     {
```

Listing 5.4: The statements checking if the item can be fitted into this suitcase by weight. (From the method pack_items in the Main form)

Checking For Fit in Pivot Points

If the item could be packed in a suitcase by weight, the algorithm will try to fit the items by size. The next point in which to try to pack the item is calculated by a sorted list of packing points. The list is sorted, so the item is packed at the lowest points first. This is done in the "find_next_pivot_point" function in the luggage class. If the item placed in the given pivot point without exceeding the dimensions of the suitcase it will call the method in "check_item_for_fit" in the luggage class. This method will run through all the effected point and check if any items are placed in the points. This method can be seen on Listing 5.5, where the function will check every point in the array value. This value is an indicator of how many elements are placed in the point (the value should of course never be higher than one).

```
1 // Set the check_for_fit variable to true. If the item cannot by
      packed in one or more points, it will be set to false.
2 bool check_for_fit = true;
3 // Loop which will run through every implicated z-point
4 for (int count_z = pivot_test_point.z; count_z < pivot_test_point.z
      + test_item.height; count_z++)
5 {
6     // Loop which will run through every implicated y-point
7     for (int count_y = pivot_test_point.y; count_y <
          pivot_test_point.y + test_item.depth; count_y++)
8     {
9         // Loop which will run through every implicated x-point
10        for (int count_x = pivot_test_point.x; count_x <
              pivot_test_point.x + test_item.width; count_x++)
11        {
12            // If there are placed an item in this point, set
                  check_for_fit to false
13            if (value[count_x, count_y, count_z] != 0)
14            {
15                check_for_fit = false;
16            }
17        }
18    }
19 }
20
21 return check_for_fit;
```

Listing 5.5: The method to check if the item can be placed at the "pivot_test_point" (from the luggage class in the Main form)

If the item could not be fitted in the point in the suitcase, the item will be rotated and tried fitted again. If every possible rotation has been tried, it will then use the next packing point - or if there are no more packing points, the next suitcase.

Placing the Item

If the item could not be packed into any suitcase, it will be added to a list of not packed items, which will be printed to the later on. It will then be removed from the packing list, and the next item will be packed. If the item could be packed, it will try to move it lower in the suitcase until it meets another item in the "check_for_space_z" method in the luggage class. This will ensure an optimal packing. The item will then be packed in the suitcase by adding 1 to the value in every point the item will fill out in the suitcase, using the "place_item" method in the luggage class. The packing points is saved to the item,

so it has the information of, where it is saved. This is done by the method "save_item" from the "luggage_item" class. The corners of the item is added to the list of packing points, and the list is sorted. The item is now saved and the next item can be packed.

When Every Item is Packed

When all items have been packed (or placed in the list of not packed items), the next step in the algorithm is to generate the colors each item should be displayed with in the 3D-viewer. This code can be seen in section 5.5. In the end the algorithm will check if there has been a mistake during packing, so one of the values in the luggage has exceeded 1. If it has, it will throw an exception. It will then check, if some of the items could not be packed. This will show an error message to the users, with the information of which elements that could not be packed as seen on Listing 5.6.

```
1 if (Not_Packed_Items.Count > 0)
2 {
3     //Start the error_message string
4     string error_message = "All your items could not be packed. This
          could be because of lack of space or too heavy items. The
          rest of the items, has been packed. The items are:\r\n";
5
6     //Add each not packed item to the string
7     foreach (luggage_item not_packed_item in Not_Packed_Items)
8     {
9         error_message += not_packed_item.name + "\r\n";
10    }
11
12    //Print the error message in a MessageBox
13    MessageBox.Show(error_message, "Fatal error",
          MessageBoxButtons.OK, MessageBoxIcon.Error);
14 }
```

Listing 5.6: The code will return an error message to the users if some items could not be packed (from the method "check_error_after_packing" in the form Main)

5.3.3 Summing Up

The algorithm checks for a lot of different criteria, also a few more than described here. This will ensure an optimal packing of the items, so the highest possible number of items can be fitted into the suitcase. Therefore, the algorithm has been extended since the first design in section 4.4. The extended, but still a bit simplified, flowchart can be seen on Figure 5.10.

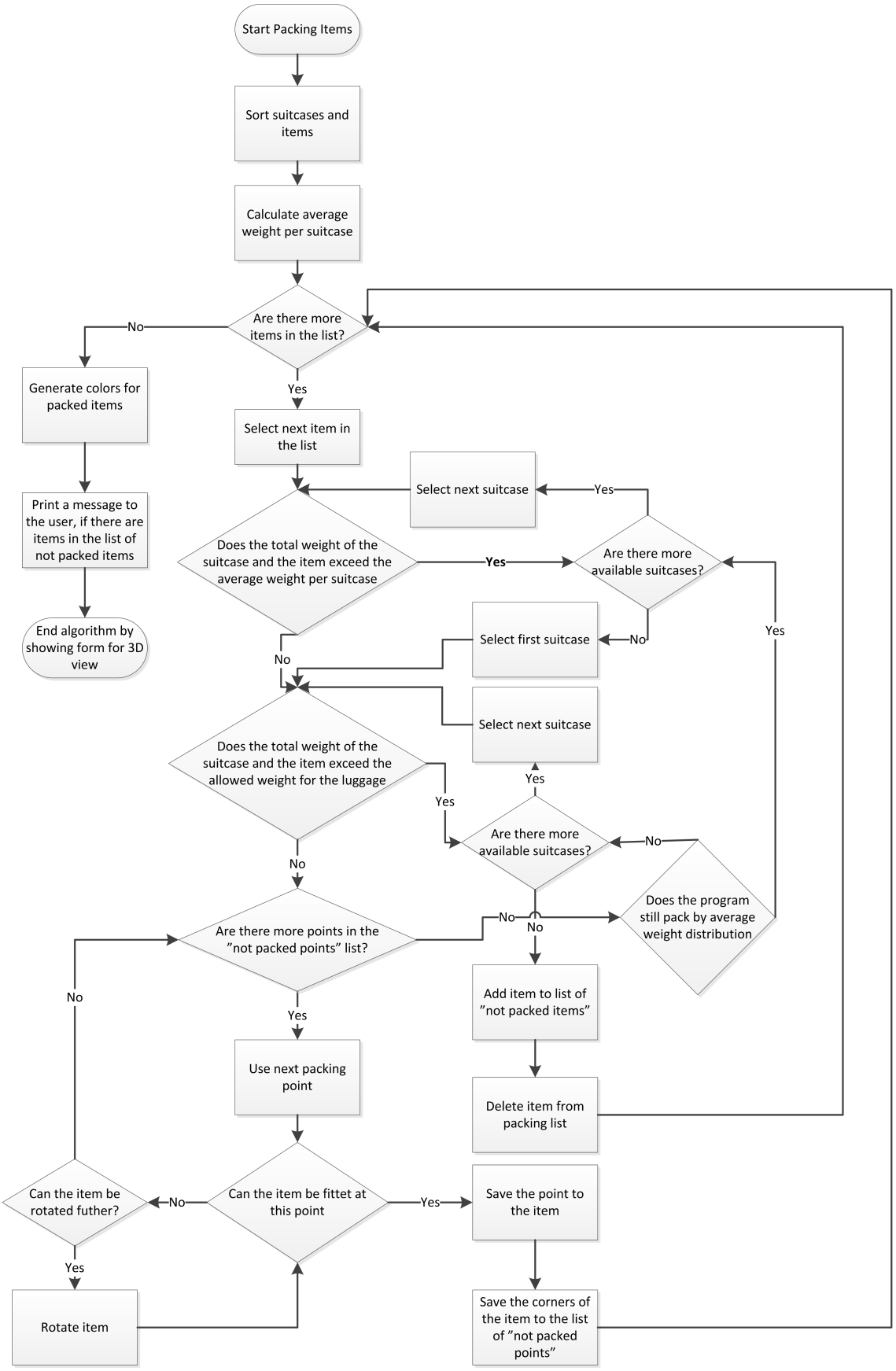

Figure 5.10: The flowchart for the packing algorithm

5.4 Color Assignment Function

This function gives each item a color and ensures that items that lie next to each other do not get the same color. The function has four steps to complete before its task is finished.

The first step is to add certain coordinates of each item into a list. This is done for all items which are to be packed. First, a "for" loop runs through the items that are to be packed. Inside the first "for" loop there are three "for" loops which run through the x, y and z coordinates starting at the saved coordinate for the current item. The loops then run until the limits are reached for the three "for" loops, which are the height, width and length. These loops can be seen on listing 5.7. At the end of the three "for" loops there is an "if" statement, which controls that it is only coordinates(x, y, and z) on the outside of an item that are added. The "if" statement works by checking if one of the coordinates (x, y or z) is either at the respective start point(saved point x, y, or z) or at the maximum limit (height, weight or length) and then only allow coordinates where this is fulfilled and thereby achieve coordinates on the outside of the item. When the "if" statement is fulfilled, the three integers x, y and z are added to their representative lists that are stored in a temporary variable of type cbox. Cbox is a class which has 3 lists in it. These three lists outline the sides of their respective object. The temporary variable are then stored in the "BoxColor" list which is the final list. Due to the size of the "if" sentences, it is not displayed in the report. See the program on the CD, see Appendix D.

```
1 for (i = 0; i <= Number; i++)
2 {
3     tmp_list = new cbox();
4     // Loops that goes through the 3 coordinates
5     //First the x-coordinate
6     for (x = (luggage_items_to_pack[i].saved_point_x - marking); x
          <= (luggage_items_to_pack[i].saved_point_x +
          luggage_items_to_pack[i].width + marking); x++)
7     {
8         //Then the y-coordinate
9         for (y = (luggage_items_to_pack[i].saved_point_y - marking);
              y <= (luggage_items_to_pack[i].saved_point_y +
              luggage_items_to_pack[i].depth + marking); y++)
10        {
11            //And finally the z-coordinate
12            for (z = (luggage_items_to_pack[i].saved_point_z -
                  marking); z <=
                  (luggage_items_to_pack[i].saved_point_z +
                  luggage_items_to_pack[i].height + marking); z++)
13            {
```

Listing 5.7: The four "for" loops which control the item and the (x; y; z) value that makes the point for the items. The slice of code is taken from the function Colors in frmMain

The second step is to check which items are neighbors through two "for" loops. Thereby determining if they are located next to each other. If they are, the item number will be saved in a list that is called "neighbor". This step works by four "for" loops, where the first loop keeps track of the current item that is being checked for neighbors. The second loop keeps track of the item that is being checked, and if it is a neighbor to the first one. The last two loops work with the points that are in the item's lists. But it will only run these loops if the items are in the same suitcase (Which can be seen by the "if" statement that are between the first two and the last two "for" loops). If they are next to each other it will save the number of the item into a list in the other item. This is done for the other

item as well. The 2 loops and setting of an the neighbor of the item can be seen on listing 5.8.

```
1 for (k = 0; k <= BoxColor[i].box_x.Count - 1; k++)
2 {
3     for (q = 0; q <= BoxColor[j].box_x.Count - 1; q++)
4     {
5         /*If statement that controls that there are any points that
              are the same in the Lists.
6           This is done by comparing the 3 koordinates to each other*/
7         if (BoxColor[i].box_x[k] == BoxColor[j].box_x[q] &&
8             BoxColor[i].box_y[k] == BoxColor[j].box_y[q] &&
9             BoxColor[i].box_z[k] == BoxColor[j].box_z[q])
10        {
11            /*Only when the if statement is fulfiled the item is
                  added to the neighbor List*/
12            luggage_items_to_pack[i].neighbor.Add(j);
13            luggage_items_to_pack[j].neighbor.Add(i);
```

Listing 5.8: The 2 "for" loops and the "if" statement that check if the two items shares any point. The slice of code is taken from the function Colors in frmMain

Thereafter the list "neighbor" is gone through to remove all the duplicates of themselves from the list. This works with one "for" loop which keeps track of which items are being worked with and when the loop is done the neighbor list is sorted. Then there is a "while" loop which goes through all the neighbors (items that are next to it) and checks if the item itself is in it and removes it. It will also check if the number after the current is the same, if it is the same it will remove one of them. This can be seen in listing 5.9.

```
1 if (!(count - k == 0))
2 {
3     /*If statement that compaired the curret neighbor with the
          neighbor after,
4       if they are identical the curret neighbor is removed*/
5     if (luggage_items_to_pack[i].neighbor[k] ==
          luggage_items_to_pack[i].neighbor[k + 1])
6     {
7         luggage_items_to_pack[i].neighbor.RemoveAt(k);
8         k = 0;
9         count--;
10    }
11    else
12        k++;
13 }
```

Listing 5.9: "If" statements which check the item after the current item to see if they are identical and if they are remove one of them. This slice of code is taken from the function Colors in frmMain

The last step taken is to go through the item list and give a color to each item. In this part the colors are numbers, where number 0 is the standard color. First there is a "for" loop which sets the current item. Before the current item is given a number based on its neighbors, its value is set to 0. Then there is a "while" loop which goes through all the neighbor of the item and check if the color is the same. If they are the same, then the color number of the item is increased by 1. This can be seen in listing 5.10. When the neighbor list is finished, the color is saved in the item, the color number is set to 0 again, and the next item in the list is checked.

```
1 if (Bcolor > ColorCode.Count() - 1)
2 {
3     j = luggage_items_to_pack[i].neighbor.Count() + 1;
4     Bcolor = 0;
5 }
6 else
7 {
8     /*Here the neighborcolor is compaired to the color on the Bcolor
          index in Colorcode.
9       If they are the same the Bcolor is counted up 1 and j set to 0
            to reset the while loop, else j is counted up by 1*/
10    if (ColorCode[Bcolor] == neighborcolor)
11    {
12        Bcolor++;
13        j = 0;
14    }
15    else
16    {
17        j++;
18    }
19 }
```

Listing 5.10: "If" statement that checks if an item and neighbor have the same color. The slice of code is taken from the function Colors in frmMain

5.5 How the Program Handles 3D

The 3D in this program is handled by objects of the class Shape3D. A Shape3D object contains an array of objects of the class Polygon. A Polygon object contains an array of objects of the class Vector and an object of the class Pen. The Pen class contains a Color and a width. A Vector object contains three integers: x, y and z, which are the coordinates of the Vector.

Instead of programming the whole 3D part ourselves, an existing solution was found and developed further. The original solution [Keselman, 2002] was only a demonstration on how to draw a cube, an axis and a more advanced polygon using C#. This solution was further developed to support drawing cubes of different sizes. Along the development, unnecessary features such as drawing an axis was removed. The interface of the 3D was also improved, so that no GUI elements were created directly in the code which handled the 3D.

5.5.1 Shape3D Class

A Shape3D object contains four methods: Draw, Rotate, ScaleSize and RePaint.

The Draw method is really just a "foreach" loop which calls the Draw method in the Polygon object on all Polygon objects within the Shape3D. The Draw method in the Polygon object will be explained in Section 5.5.2.

The Rotate method in the Shape3D object is used to rotate the object. The method takes one input parameter, which is a RotateMatrix called "mat". Next the method creates a new Polygon array newP, and runs through all Polygon objects in the Shape3D. The method then calls the Rotate method within the Polygon object on the current polygon, and inserts the results into the Polygon array newP. The method also inserts the current Polygon's Pen into the newP Polygon's Pen. The function can be seen on Listing 5.11.

```
1 public Shape3D Rotate(RotateMatrix mat)
2 {
3   int count=0;
4       //Initialize new Polygon array with enough space allocated to
            hold all the polygons
5       Polygon[] newP = new Polygon[poly.Length];
6
7       //For each of the polygons in the current polygon array,
            rotate then.
8       foreach (Polygon p in poly)
9       {
10          newP[count++] = p.Rotate(mat);
11          newP[count-1].MyPen = p.MyPen;
12      }
13      //Return the new Shape3D
14  return new Shape3D(newP);
15 }
```

Listing 5.11: Method to rotate an object (from Shape3D)

.

The ScaleSize method in the Shape3D object is almost the same as the Rotate method in Shape3D. The only difference is that instead of calling the Rotate method in the Polygon object, the ScaleSize method calls the ScaleSize method in the Polygon object.

The RePaint method in the Shape3D object is used to mark one item in the suitcase(one polygon), with a thicker line than the others. In the program this is used when the user clicks on an item in the list of items to the right of the drawing of the suitcase. If the user clicks on an item in the list, the program will mark that item with thicker lines. This function is also used when the 3D viewer is opened - a small cube with dimensions of 0.5 x 0.5 x 0.5 is drawn in the [0;0;0] point of the suitcase, so the user knows where the [0;0;0] point is. This cube is also marked with a thicker width than the items.

The method takes two integers as input parameters: "num" and "col_w". The "num" input is the item that has been selected, and the "col_w" is how wide the line must be. The method initiates a new Polygon array, newP, where all the polygons will be stored. The method also contains three counters. One that counts the elements in the polygon, this counter is called "element_counter". Here an element consists of four sides. Another counter called "shape_counter" is counted up when a polygon is made thicker. The last counter is counted up whether or not a polygon is made thicker. This counter is called "polygon_counter" which keeps track of the Polygon array newP.

First the method checks if the color of the current polygon is red. This only happens if the current polygon is the cube representing the 0,0,0 point. If the color is red, the width of the polygon is set to 2, and the counter "counter" is counted up. The method now checks whether or not "counter" is 4. This occurs if all four sides of a polygon is made thick. If this is the case, the counter "counter" is set to 0, and the element counter is counted up, implying that the method should now check the next element in the Shape3D object. This can be seen on Listing 5.12.

```
1 //If the color of the current polygon is red, the polygon will be
      marked thicker.
2 if (p.MyPen.Color == Color.Red)
3 {
```

```
4     newP[polygon_counter] = p;
5     //Make the width of the line thicker
6     newP[polygon_counter].MyPen.Width = 2;
7     shape_counter++;
8
9     //All four sides in the polygon must be marked thicker.
10    if (shape_counter == 4)
11    {
12        shape_counter = 0;
13        element_counter++;
14    }
```

Listing 5.12: Method that makes an item thicker (from Shape3D)

At the end the method returns a new Shape3D object, newP, which replaces the old Shape3D, and is drawn.

5.5.2 Polygon Class

The Polygon class contains five methods: Draw, X2D, Y2D, Rotate and ScaleSize. The essential method in this class is the Draw method. This is the method which actually draws the suitcase and items on the form. The Draw method takes two input parameters; a Graphics object "g", and a Pen "MyPen". The Pen is where the color and width of the lines are stored. Before explaining how the Draw method actually works, a closer look will be taken on where it is first called.

Every time the method 'Invalidate' is called, the Draw method in the current Shape3D object is also called. The 'Invalidate' method is called every time the program needs to refresh the drawing of the suitcase and items. For example when the user changes suitcase in the drop down element, the program needs to redraw the image, so that the items in the new suitcase are drawn instead.

In the frm3DViewer, the Draw method is called inside the method MyPaint, which takes two input parameters: an 'object' "sender", and an 'PaintEventArgs' "e". The method then calls the Draw method in the Shape3D, with the parameter e.Graphics. The Draw method in Shape3D receives the 'Graphics' "g". The method then runs trough each Polygon in the Shape3D and calls a Draw method in the Polygon class on each of the Polygons. It is now clear that the Draw method is called on each Polygon in a Shape3D.

The Draw method in the Polygon class begins with a "for" loop which runs through the length of the Polygon. The length of a Polygon is defined by the combined length of the Vectors within the Polygon. Inside the "for" loop, two lines are drawn with the built in method DrawLine in the Graphics class. The DrawLine method takes three input parameters, a Pen, and two points. But the Polygons do not contain any points. Therefore the Vector must be converted to points before it is drawn. The Draw method in the Polygon class can be seen on listing 5.13.

```
1 public virtual void Draw(Graphics g, Pen MyPen)
2 {
3   for (int count=0;count<Len-1;count++)
4               //Draw all the vertical lines
5           g.DrawLine(MyPen,
6               X2D(vectors[count])  ,Y2D(vectors[count])  ,
7               X2D(vectors[count+1]),Y2D(vectors[count+1]));
8               //Draw all the horizontal lines
```

```
 9             g.DrawLine(MyPen,
10                 X2D(vectors[Len-1])   ,Y2D(vectors[Len-1])   ,
11                 X2D(vectors[0])       ,Y2D(vectors[0]));
12
13 }
```

Listing 5.13: Method to draw on form (from the Polygon Class)

The Rotate method in the Polygon class basically receives a matrix, which contains how the Polygon should be rotated. The method then run through each vector in the Polygon calling the Rotate method in the Vector class on each Vector with the rotate matrix as input. When this is done a new Polygon is returned. This is the rotated Polygon.

The ScaleSize method in the Polygon class is used to scale the polygon with a factor. This is used to zoom the drawing in and out. The method receives a float as input, runs through every Vector in the Polygon and calls the ScaleSize within the Vector class on every Vector, parsing the float as parameter. When this is done a new Polygon is returned. This is the scaled Polygon. The ScaleSize method in the Polygon class can be seen on Listing 5.14.

```
1    //B130 method to scale vectors
2 public Polygon ScaleSize(float d)
3 {
4   int count=0;
5   Vector[] retVec=new Vector[Len];
6        //For each Vector in the polygon, call the ScaleSize method
             from the vector class on the vector
7   foreach (Vector vec in vectors)
8     retVec[count++]=vec.ScaleSize(d);
9        //Return the new polygon.
10  return new Polygon(ZVal,Zero,retVec);
11 }
```

Listing 5.14: Method to scale the polygon (from the Polygon Class)

5.6 Function to Draw Cubes

The function called "DrawSomething" is the function which basically converts dimensions and coordinates into the vectors which are to be drawn. "DrawSomething" takes 11 input parameters. These inputs are the dimensions and coordinates of the cube to be drawn represented as floats. Next the function takes 3 integers which are: The number of the suitcase, the item to be drawn lies in, the factor of zoom that should be applied to the item, and the index of the array of polygons which has currently been reached. Lastly, the function takes a Vector and a Pen as input.

The "Pen" represents the color and width of the cube to be drawn. For example the function could draw a red cube with 5 pixel lines. In the beginning of the function some variables are defined. These are the start location of the drawing. Four Vector arrays are created each holding 4 Vectors, which together form a square. Three floats are defined, they will contain half of each of the dimensions of the item. The current items y and z coordinates are inverted, because of the way the Vectors will be calculated later. The function contains a conditional expression which checks which type of cube the function

should generate. This is because the function can be used to generate a suitcase, the items in the suitcase, and a small cube(which represents the $\begin{pmatrix}0\\0\\0\end{pmatrix}$ point of the suitcase).

The expressions checks on the lug_num variable which is really the number of the suitcase the item lies in. But if the lug_num is set to -1, it means that the function should draw a suitcase instead of an item. And because the drawing should rotate around itself, it has to take the coordinates of each item and subtract half of the suitcase dimension. This, in turn, makes sure that the [0;0;0] point of the coordinates systems will be placed in the middle of the suitcase, allowing the user to rotate the suitcase around itself, but the small cube is still placed in the $\begin{pmatrix}0\\0\\0\end{pmatrix}$ point of the suitcase. So if the function has to draw a suitcase, it sets the suitcase dimensions to the half of the dimensions received as input. If the function is to draw an item in the suitcase, the function will find the dimensions of the items and half of the dimensions of the suitcases, and use these to manipulate the start point of the current item. This is to place the item correctly inside the suitcase in the drawing, while allowing the user to rotate the suitcase around itself. If the function receives -2 as the lug_num, it means that it should draw a small cube with dimensions $0.5 \times 0.5 \times 0.5$ at the $\begin{pmatrix}0\\0\\0\end{pmatrix}$ point of the suitcase.

After the conditional expression, the dimensions of the input item is subtracted by 0.01. This is so no item touch another item, allowing the user to identify each item easily. The next part of the function is where the Vectors are calculated and put together into a Polygon and inserted into the array of Polygons. Only the part where the first Polygon is calculated will be explained, because the three next are essentially the same, but with the vectors pointing in other directions, forming the other side of the cube which will be drawn. A Vector contains three points, x, y and z. It is these points that now will be calculated. Earlier four Vector arrays were created: points1, points2, points3, and points4. Each of these Vector arrays contains four vectors to form 1 side of the cube to be drawn. Keep in mind that the y and z coordinates of the item to be drawn were inverted earlier. To calculate the x-point of the first vector of the first side, the function takes the x-coordinate and subtracts half of the width of the suitcase. This is again because the $\begin{pmatrix}0\\0\\0\end{pmatrix}$ point of the drawing must be in the middle of the suitcase. The y and z point of the first vector are essentially the same, but adding half of the width and depth of the suitcase.

For example if the program has a suitcase with the dimensions $50\times100\times80$ (height $\times$ width $\times$ depth), an item with the dimensions $40\times70\times30$ that reside in point $\begin{pmatrix}0\\0\\0\end{pmatrix}$ of the suitcase, the first x-point of the item would lie in point -50 (0-(100/2)). The y-point of the first vector is calculated by adding half of the depth of the suitcase in this case 25 (0+(50/2)). And the z-coordinate of the first vector in the example item would then lie in point 40 (0+(80/2)). When the points have been loaded into the new Vector, the vector is scaled by a factor of "zoom_factor". This also happens when zooming in and out, but of course with different values. See section 5.5 to see how the function "ScaleSize" works.

These calculations are shown on Listing 5.15

```
1 //Calculate the first point in the cube
2 points1[0] = new Vector(item_point_x - lug_half_width, item_point_z
      + lug_half_height, item_point_y +
      lug_half_depth).ScaleSize(zoom_factor);
```

Listing 5.15: Calculates the points of the first vector

From the calculations above the first Vectors coordinates are $\begin{pmatrix} -50 \\ 25 \\ 40 \end{pmatrix}$, and can be seen on Figure 5.11.

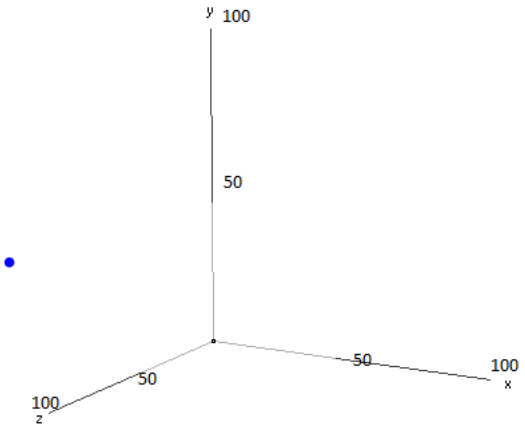

Figure 5.11: First point of the first side of the example item

Naturally the second vector of the side needs to have at least one point in common with the first vector. Else the side would not be a complete square. The common point of the first and the second Vector is the x point. The y coordinate of the second Vector is calculated by subtracting the item depth from the y-coordinate of the item and then adding half of the suitcase height. With the example item, this gives an y-coordinate of the second vector of -15 (-0 - 40 + (50/2)). The z-coordinate of the second Vector is calculated by adding the y-coordinate of the item to half of the suitcase depth. This gives a z-point of the second vector of 40 (-0 + (80/2)). The calculations can be seen on Listing 5.16.

```
1 //Calculate the second point in the cube
2 points1[1] = new Vector(item_point_x - lug_half_width, item_point_z
      - item_height_float + lug_half_height, item_point_y +
      lug_half_depth).ScaleSize(zoom_factor);
```

Listing 5.16: Calculates the points of the second vector

The second Vector thereby becomes $\begin{pmatrix} -50 \\ -15 \\ 40 \end{pmatrix}$, and can be seen on Figure 5.12.

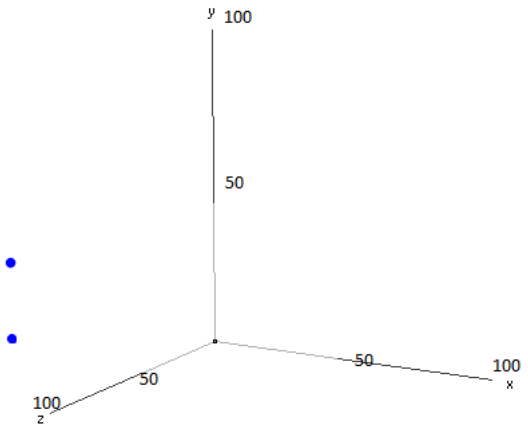

Figure 5.12: First point of the first side of the example item

Now for calculating the third Vector. The third Vector has both the y- and z- coordinates in common with the second Vector. These are thereby accordingly -15 and 40. To calculate the x-coordinate, the x-point of the item is added to the width of the item, while half of the width of the suitcase is subtracted from this. To continue the example, the x-coordinate of the third Vector would thereby become: 20 (0 + 70 - (100/2)). The calculations can be seen on Listing 5.17.

```
1 //Calculate the third point in the cube
2 points1[2] = new Vector(item_point_x + item_width_float -
      lug_half_width, item_point_z - item_height_float +
      lug_half_height, item_point_y +
      lug_half_depth).ScaleSize(zoom_factor);
```

Listing 5.17: Calculates the points of the third vector

The third Vector thereby becomes $\begin{pmatrix} 20 \\ -15 \\ 40 \end{pmatrix}$, and can be seen on Figure 5.13.

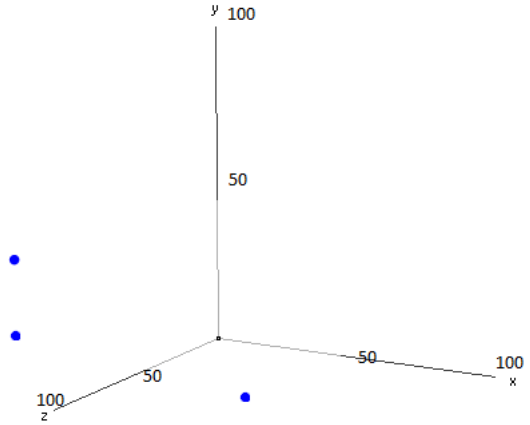

Figure 5.13: First point of the first side of the example item

The fourth Vector has both the x- and z-coordinate in common with the third Vector, while having the y-coordinate in common with the first Vector. The calculations can be seen on Listing 5.18.

```
1 //Calculate the fourth point in the cube
2 points1[3] = new Vector(item_point_x + item_width_float -
      lug_half_width, item_point_z + lug_half_height, item_point_y +
      lug_half_depth).ScaleSize(zoom_factor);
```

Listing 5.18: Calculates the points of the fourth vector

The fourth Vector thereby becomes $\begin{pmatrix} 20 \\ 25 \\ 50 \end{pmatrix}$, and can be seen on Figure 5.14.

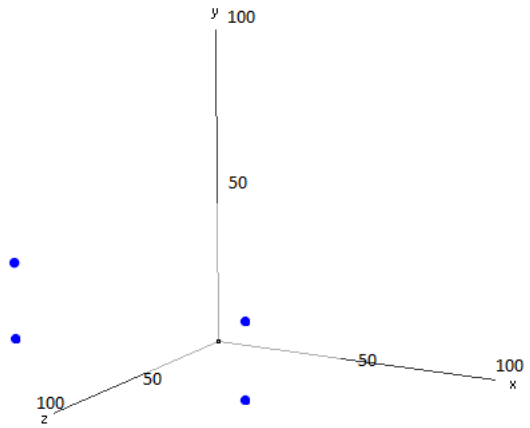

Figure 5.14: First point of the first side of the example item

By drawing a line between these four points, it is clear that these form a square, which is the first side of the item in the example. This can be seen on Figure 5.15

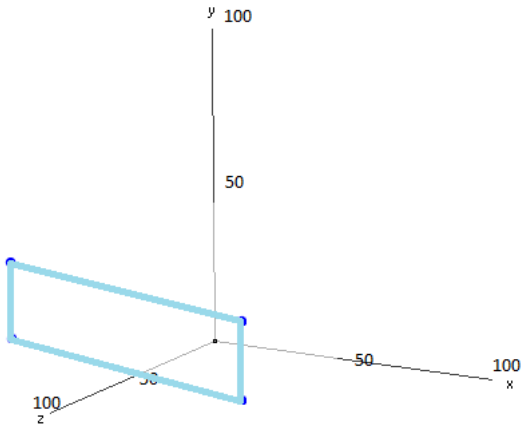

Figure 5.15: How the first side of the example item looks

When the four Vectors have been calculated, the Vectors are inserted into a new Polygon, which is inserted into an array of polygons. Next the current Polygon (the one just created), is assigned a Pen. The Pen contains the color and the width of the Polygon. This can be seen on Listing 5.19

```
1 //Create a polygon from the four calculated points, and add it to
      the polygon array
2 poly_array[poly_counter] = new Polygon(1000, z, points1);
3 poly_array[poly_counter].MyPen = new Pen(itempen.Color,
      itempen.Width);
4 poly_counter++;
```

Listing 5.19: Calculates the points of the fourth vector

This was the first side of the item, it is basically the same for the rest of the sides.

5.7 Rotate 3D-view with the Mouse

To make it easier for the user to see exactly where and how the various items are packed within the suitcase, it is necessary with some kind of rotate function. As described in section 5.6, the point which the suitcase rotates around is residing within the suitcase, so that the suitcase is able to rotate around itself. When the user moves the mouse over the drawing, an event is triggered. The event triggers the function called "MyMouseMove", which receives two input parameters, an 'object' "sender" and a 'MouseEventArgs' "e". The "MouseEventArgs" contains all the information regarding the mouse which is needed to perform certain checks. The "MyMouseMove" function is called whenever the user moves the mouse on the form. But the function should only do something, when one of the mouse button is clicked while the mouse is moved. Therefore the "MyMouseMove" contains two conditional statements. The first condition checks if the left mouse button

is currently pressed, and the other condition checks if the right mouse button is currently pressed.

If the left mouse button is pressed, the function first defines some variables that are used later. The icon of the cursor is also changes to be the "hand" icon, representing that the drawing is rotating. Next the function gets the current x- and y-coordinates of the mouse. The coordinates of the mouse when the click occurred is now subtracted from the current mouse coordinates. The result is how much the drawing should be rotated. But if the result is just passed as how much the drawing should rotate, it would be uncontrollable, therefore it is divided by 15, and then by a variable called "rotatesensitive". The "rotatesensitive" contains the current value of the scrollbar on the form, where the user can set the sensitive of the rotate. Next the result is passed as a new "XRotate" for the x-coordinates and a new "YRotate" for the y-coordinate. This can be seen on Listing 5.20

```
1 //Divide by rotatesensitive and 15 so that the rotating is
      controllable
2 rotx = rotx / (rotatesensitive * 15);
3 roty = roty / (rotatesensitive * 15);
4
5 Xmat = new XRotate(rotx);
6 Ymat = new YRotate(roty);
```

Listing 5.20: The function that handles the rotating with the mouse

This is what actually creates the rotation the next time the drawing is invalidated (refreshed) which is what happens next. Just before the "invalidate" is called, the coordinates from when the mouse was first clicked, is set to be the current mouse coordinates. This creates an even rotation, else the drawing would rotate faster and faster the farther the user moved the mouse from where it was first clicked.

5.8 File Serialize Function

When the user shuts down the program, it should save the lists of the items and suitcases and open them next time the program is opened. There are many different ways for saving data. It can e.g. be saved in a raw binary file, a text file or a database like SQL. The SQL databases either needs a program which can handle SQL or an Internet connection to a server, and is therefore not used in this project. A text file is more complex way of saving data, because the program will need methods both for writing and reading the text file. The binary raw file is therefore used in the project because it is smarter than SQL and the text file.

To save the data in the program a function called File Serialize is used. The function is called when the data from the program should be saved or loaded. The function lies in the file "FileSerializer.cs", which is taken from the website [sdktsg] and modified for this project. The code parts which should be implemented in the classes to use the FileSerializer is described.

```
1 /*here the info for the class is made for the FileSerializer so it
      can make the file*/
2 public void GetObjectData(SerializationInfo info, StreamingContext
      context)
3 {
4     info.AddValue("weight",this.weight);
```

```
5     info.AddValue("width", this.width);
6     info.AddValue("height", this.height);
7     info.AddValue("depth", this.depth);
8     info.AddValue("name", this.name);
9     info.AddValue("number", this.number_of_items);
10 }
```

Listing 5.21: Saves the values of the item to the variable "info" used by the FileSerializer. Code can be found in the luggage_item class

On listing 5.21 it can be seen that the "File Serializer" needs some data descriptions to turn the information from the item list into a data file. This is done by stating sentence, where the first part is the name of the certain information, which needs to be saved into the file.

```
1 /*here the info for the class is made for the FileSerializer to open
      and read the file*/
2 private luggage_item(SerializationInfo info, StreamingContext ctxt)
3 {
4     this.weight    = info.GetInt32  ("weight");
5     this.width     = info.GetInt32  ("width");
6     this.height    = info.GetInt32  ("height");
7     this.depth     = info.GetInt32  ("depth");
8     this.name      = info.GetString ("name");
9     this.number_of_items = info.GetInt32("number");
10 }
```

Listing 5.22: Get the values to the item from the variable info from the FileSerializer. Code can be found in the luggage_item class

To open the data file the program also needs to have data description on how the data is saved in the file, see listing 5.22. The program gets the information from the file, where the name of the certain information is, and puts it into the class list for each of the information which is saved.

5.9 Instruction Manual

The instruction manual is placed in the main window (see Figure 5.1) and is made as a help for the user. It is made as short as possible since experiences from earlier projects show that users tend to skip parts of instructions if they are too long. The instruction manual tells the user how to use the program and how it works. The instruction manual is made on the basis of how it is intended to be used.

The manual is placed in the main menu because then it is the first thing the user sees, when he or she opens the program. It is also made with bullet points to keep an overview of it, and to make it easier to read.

6 Testing

This section will describe the tests, the results from the tests and the improvement on the program which came from the results of the tests. To help ensure that the program does not have any obvious bugs, test were performed on the program which helps find those bugs. Therefore it is better that a person who is not a part of the project group is going to be the one testing it. If it is a person from the group who is testing, he or she might not find a bug or problem with the program, because he/she is already used to the program.

6.1 Choice of Survey Method

The user tests were made where the test persons tried the program and got interviewed afterwards. For the analysis and further work regarding the test results, a qualitative method has been chosen.

For the interview, a questionnaire has been made (which can be seen on Appendix C), which will be used as a guideline in the interview to get useful information from the test person. The qualitative method is the best suited method for the project, because it gives a more specific picture of how the test person finds the program, and what problems the test person might come across during the test. It is also a good method to get suggestions for improvements from the user, since it is normally easier to explain an idea while talking rather than writing. The problem with the interview is that the time it takes to analyze is longer than a questionnaire, where you can analyze a lot of data quickly. The questions at the interview need to be well thought and precise to get useful information from the test persons, as they would also need to be on a questionnaire. Based on the test, improvements should be made to the program. It is important when making a questionnaire that it is structured correctly and that the questions are precisely formulated so the test person cannot misunderstand the meaning of the question. It is important because poorly formulated questions might lead to misunderstandings the meanings of the questions which can lead to wrong or useless answers. In such case a questionnaire is just a waste of time.

After the test person has tested the program, an interview based on a questionnaire, with focus on what can be improved on the program will be made.

6.2 The First Round of User Tests

Two tests of the program have been made on two test persons in the first round of tests. The main idea was to see if the program had any bugs and to get some suggestions to

what could be improved and if anything was missing. The test persons were first given some instructions on what they were supposed to do in the test and were afterwards asked to read the instructions in the program. The test persons were to load two lists in the program and then add some missing items after having measured each weight of the items and the dimensions. Afterwards they were asked to pack two suitcases. When the test persons were done packing an interview was held. Two group members were present during the testing. One was documenting everything and afterwards interviewing the test person, while the other was placed behind the test person to see what mistakes the test person made.

After the first test some labels in the program were changed to help the understanding of how to use the program. The instructions in the program were also changed to describe the program better. The first test person did not have any problems at all in packing the suitcases but stated that the person did not find the program useful. The person suggested adding a function making the program able to take into account if an item is bendable.

The second test got interrupted because of some problems with loading one of the lists which caused a breakdown of the program. The test was retaken the day after the bugs was corrected. Before the 2nd test we made some changes in the instructions, saving functions, added the bottom of suitcase to the 3D-viewer, added some new buttons on the forms: manage suitcase, manage items, add item, and add suitcase. After we made those changes we began test 2.

This time there were no technical problems with the program, and the test was made without more serious problems. After the test the instructions were changed again in order to describe some features in the program better and also describe how to rotate and drag the 3D-model in the 3D-viewer. Another element that was changed was that the user now has to write the weight limit in kilogram instead of gram. The mistake with the program asking if the user wants to save the lists after the user has saved them by using the button was also fixed after the test. The list in the 3D-viewer now sorts the items so the ones on the bottom of the suitcase is in the top of the list. The last aspect which was changed was that an image of a suitcase was added in the "Manage Suitcases"-window and a picture of several items was added to the "Manage Items"-window, but this person thinks that the program is useful and might use it in the future.

6.3 The Second Round of User Tests

In the second round of tests we tested two test persons. Some small improvements to the test method were made in this round of tests. The instructions about what would happen in the test were written down instead of just told so the test person would not forget the instructions, and the instructions themselves were improved so they were easier to understand for the test persons.

The second round of tests were made a week after the first test. The test went without any technical problems and was a success. After the two tests the following was changed: Since both test persons complained about the time it would take to put in all the info needed in the program, a drop down menu of some standard items has been added in the "Add item"-window in the "Manage items"-window. A better description of what is at the bottom of the suitcase has been added in the 3D-viewer. The little box showing the [0;0;0]-coordinate has been explained in the 3D-viewer too. The problem about a lot of items being packed upright was solved and the algorithm has been optimized. A better

description on how to add more of the same items has been added to the instructions. If the user is managing an already saved list and clicks the "Save list to Computer"-button the program will ask the user if the user wants to overwrite the saved list. In the 3D-viewer the unit for the weight of the suitcase has been added. This time both test persons think that the program is useful and might use it in the future.

6.4 Conclusion on the User Tests

The tests went well and the group got some good feedback which lead to some great improvements of the program and the bugs in the program were fixed. Most of the test persons found the program useful if they had to pack a lot for a vacation. Testing more than once proved a very good idea, since all the test persons had good suggestions to improvements and found minor bugs that quickly were fixed. Despite some smaller problems regarding the testing, the results of the tests were useful for the project and therefore a good time investment.

6.5 Unit Tests

Unit tests are used to test specific parts of the code in a program. It is very important because overlooked bugs can be spotted more easily. In this project, unit tests were only used in a limited extend, because it was taught very late in the object oriented programming course. Unit tests were applied to the frmMain form in the program, and they only test methods regarding the managing of suitcases. The unit tests helped removed minor bugs in the handling of the suitcases.

```
1 /// <summary>
2 ///A test for luggage Constructor
3 ///</summary>
4 [TestMethod()]
5 public void frmMain_luggageConstructorTest1()
6 {
7     //Initiate new suitcase
8     frmMain.luggage target = new frmMain.luggage();
9
10    //Check if the initiated suitcase is not null
11    Assert.IsNotNull(target);
12 }
```

Listing 6.1: Test of the suitcase constructor

Listing 6.1 shows a unit test, which tests the constructor of the luggage class. If the "target" is not null, the test will pass, and the constructor works.

7 Discussion

This chapter will look at the final product and see if the Targeted Features, (see section 4.1), have been fulfilled and how well they are covered. After the discussion there will be a perspectivation of the project to look if some aspects of the program might be useful in the future, like how the interface is made to be user friendly and how there has been made tests on early program builds to help finding errors.

In this chapter the final product will be reflected upon whether the Targeted Features (TF) have been fulfilled and how well the specification requirements have been met (see section 4.1).

7.1 Discussion

All the requirements (see section 4.1) have been met, which can be seen as a success. The program can deal with many of the problems found in the problem analysis. The program can help the user in the process of packing a suitcase, and will by a simple interface pack the suitcase, where the user did have a problem packing it on his/her own.

The program lets the user add both items and suitcases to the program which then packs the suitcases with the items in a weight/space optimizing way. The result is then presented to the user, who can then see what has been packed and where. The items which have not been packed are shown to the user through a message box. So the program provides the user with a solution to the packing problem and thereby the program fulfills the project requirement, which can seen in appendix A.

To help the user get a better visual understanding on how to pack the suitcase a 3D-image is made for the user which can be can rotated, zoomed and dragged. In the right side of the 3D-form a list is displayed. This list shows all the packed items in the selected suitcase. By selecting an item it will be marked in the 3D-image so the user will easily be able to identify any given item. An explanation of this function can be seen in section 5.5. To help the user furthermore an algorithm has been made that ensures that any two items next to each other will have different colors, so they are easier to identify from each other. The function and a description of it can be seen in section 5.4. It is easy to get a good overview in the program, because the maximum number of buttons is 5 in all the different windows and no unnecessary features are in the windows. The 3D-viewer is very user friendly because the user can use the mouse to zoom and rotate the 3D-image of the suitcase containing the packed items. The GUI is described in section 5.1.

The 3D-viewer and GUI guides the user in the packing process and gives the user the

possibility to navigate through the program. This covers the requirement which states that the program will need to guide the user in order to help solving the given problem. The stated problem can be seen in section 2.5. The 3D viewer and GUI also provides the user with a packing list where the user can select an item and see the packed item on the provided 3D figure. This is the method used to cover the requirements "Structure of packing" and "Packing list".

The user is able to save a list which was made on the computer and load saved list every time the program is used. This means the user can load the list any time he/she wants. Thereby the user can load the list on the vacation and use the program. The only condition is that the user must bring a computer. The "save" function then covers the save/load requirement and the on the road.

So the program is based on the requirements and has been improved by using the results of the tests which have been made on the program. The tests have helped with improving the design of the program and a few function in the program. This has been a very good tool in improving and bug-testing the program. The improvement made after the tests can be seen in chapter 6.

So the product includes all the important requirements but the optional features have not been included because of the lack of time to work on them. It was decided that it was more important to make the targeted features as perfect as possible, before working on optional features. The program is capable of packing one or more bags based on data on the bags and a list of items which should be packed. But the program unfortunately does not provide a packing solution as the human brain can. The reason for this is that the program does not allow object to be flexible. On the other hand the program takes weight into account which can be hard to remember when packing. The program does also remember items from last time which can save time for the user because he/she does not need to start from scratch.

In the following section, Perspectivation, the experience will be reflected upon and try to related processes in the project to the real world outside the project. There will also be thoughts on how user will use the program.

7.2 Perspectives

When working with problem solving it is also important to look at how the user will use and interact with the solution. Through these assumptions a better solution can be formed that is more user friendly. Therefore it is a good thing to know who are going to use the solution, and from that design it more thoroughly.

The purpose in this project was to make a program which provided the user with a packing solution and help the users pack a suitcase more effectively. The program should also check that the weight of the suitcase does not exceed the limits given by the user. The program is made in Microsoft Visual Studio which provides the developer with a lot of good tools to make a user interface for the user of the program. This makes the program a lot more user friendly because the user can easier interact with the program. This means that it will be a lot easier for the user to operate the program and thereby the program has bigger chance at selling if it were the intention. The user interface also means that the user does not need to understand the program before he/she can use it. The program in this project also supports mouse control, this means that the user can use a mouse to navigate around the program. Because of the mouse support, and the interface the user

has less of a risk to do something wrong when using the program, but by implementing an interface there is a risk for new bugs and misunderstandings in the formulations of the interface. Therefore it is important to make a clean and user friendly interface which only contains the necessary buttons and well formulated text which helps the user by informing about what he/she should do, and what kind of unit the data is measured in.

To prepare the program and try to find potential problems, the program has undergone some tests, where people who have not worked with the project have tried the program. Their experiences from the test were then collected and used to evaluate the program. The evaluation then led to changes that would make the program better. After the changes were made, the program was tested again with the changes and the same procedure of the test result as the earlier test. This process of repeated testing is a good procedure to develop a working program because the potential user gets to use it and experiences are made in context to the use. These experiences are then used to make the program more suited for the target audience. The downside of this procedure is that it takes a lot of man hours to do the testing and make the changes. Therefore testing is a good but expensive tool, hour-wise, to use. It should always be taken into consideration to test the program, because it is important that the program is freed of the worst bugs when released. Program bugs on releases have a negative influence on the sale of programs and might scare of potential purchasers.

A side effect from making the program in Visual Studio is that the program can only run on Microsoft Windows with the .NET framework installed. The reason for this is that the program uses the .NET framework which only supports Microsoft's operating systems. This means that it is important to take a moment before developing and make a thought regarding what platform the program should be used on or better yet, if it should be able to run cross platforms. These different experiences made in this project can be used in upcoming projects.

The program, when used correctly, can help one person with a lot of suitcases or groups of people pack their luggage pretty effectively among all their suitcases. Furthermore, the program can help make sure that the user is not exceeding the weight limit, if any, for the given trip provided that the user knows the limit. This can save the user the burden of additional work and financing due to overweight and thus improve the overall experience of the trip as well as improving the environment because of the reduced weight to the transport vehicles. When the user saves money because he/she avoids the overweight fee, he/she might spend that money on the resort and thus help the given country he/she is visiting. This can have a positive impact on that country's resorts giving them more money they can spend to improve their facilities.

8 Conclusion

The final chapter of the report is a conclusion on the whole project and a conclusion on the problem statement.

To make a conclusion on this project it is necessary to take a closer look at the problem statement, see section 2.5. To meet and answer the problem statement, the program must be able to pack one or more suitcases by volume and weight in an effective way. It is also required that the program must present the result to the user in some way before it can actually help the user.

The program is capable of packing one or more suitcases with items while using the volume as effectively within the reach of the programs. When mentioning the reach of the programs, it is that the program handles the items as boxes instead as their real shape. The program does not pack as effectively as possible because that will include that the program treats items as different objects and not just boxes. The program also distributes the weight evenly among the suitcases if there are more than one and checks that the suitcases do not exceed a given limit. The program also has a user interface which fulfills the role as provider of the result.

The program gives a packing solution but not as effectively as possible and thereby the program itself does not give a full answer to the problem statement. The answer should be found through experience and knowledge gained in this project. The reason for this is that with new knowledge it will be possible to take this program and improve it so the program handles the items better. So for the program to be able to pack more effectively it must be better at handling all kinds of shapes and be able to place them according to each other.

So it can be concluded that the problem statement has been answered through the work with the final product. The problem statement is not answered through the product itself. The reason is that the program only gives the most effective packing solution within the criterion that items are handled as boxes.

The program also solves the problem stated in the project suggestion, about making a program able to help with planning how to pack a suitcase (see appendix A. It can be concluded that the program can pack one or more suitcases using weight distributing. It might be improved by a function which handles bendable shapes, but even though the program is missing that feature it is able to pack one or more suitcases in a good and efficient way. Therefore it is a sufficient program which helps the user with the process of packing one or more suitcases while using weight distributing but it could be improved if more time were available.

Bibliography

Airline. SAS Airline. *Checked-in luggage*. URL http://www.sas.dk/Charter/Bagage/Indchecket-bagage/?vst=true.

Airlines. Copenhagen Airlines. *Prohibited Items*. Internet. URL http://www.cph.dk/CPH/DK/MAIN/Foer+afrejse/Bagage/Indskrevet+bagage.htm.

Arizona.edu. Arizona.edu. *Bin Packing*. URL http://www.cs.arizona.edu/icon/oddsends/bpack/bpack.htm.

Baumgarten, **02 2012**. Henrik Baumgarten. *Det solgte de danske chartarbureauer sidste aar*, 2012. URL http://www.takeoff.dk/news/22548.

Bureau of Transportation Statistics (RITA), **23 jan 2012**. Bureau of Transportation Statistics (RITA). *Baggage fees by airline*, 2012. URL http://www.bts.gov/programs/airline_information/baggage_fees/.

CPH. CPH. *Farligt Gods*. URL http://www.cph.dk/CPH/DK/MAIN/Foer+afrejse/Bagage/Forbudt+i+h%C3%A5ndbagage.htm?WT.ac=Link-t-CPH-fra-farligt-gods.

DSB. DSB. *Bagage*. URL http://www.dsb.dk/kundeservice/service-i-toget/bagage/.

Dube and Kanavathy. Erick Dube and Leon R. Kanavathy. *Optimizing three-dimensional bin packing through simulation*.

EU-Kommisionen. EU-Kommisionen. *Bagage rules released*. URL http://europa.eu/rapid/pressReleasesAction.do?reference=MEMO/06/363&format=HTML&aged=0&language=EN&guiLanguage=en.

Foster. Susan Foster. *Pack smart for Travel*. Smart Travel press. URL http://www.roadandtravel.com/traveladvice/luggagepackingtips.htm.

Indian Railways. Indian Railways. *Luggage Rules*. URL http://www.trainenquiry.com/StaticContent/Railway_Amnities/E_R/LUGGAGE.aspx.

Kellerer, **2004**. Prof. Hans Kellerer. *Knapsack Problems*, 2004.

Keselman, **February 2002**. Ariel Keselman. *Rotating a 3D Cube and Sphere with GDI+*. website, 2002. URL http://www.c-sharpcorner.com/UploadFile/alfajores/Rotating3DCubeSphere10172005031038AM/Rotating3DCubeSphere.aspx.

MSCcruise. MSCcruise. *Cruise luggage*. URL http://www.msccruises.dk/dk_da/Krydstogtsguide/For-afrejsen.aspx.

Rosen, **1991**. Kenneth H. Rosen. *Applications of Discrete Mathematics*. McGraw-Hill, INC., 1991.

SAS, **2012**. SAS. *Bagageovervaegt*, 2012. URL `http://www.sas.dk/Alt-om-rejsen/Bagage/Bagageovervagt/?WT.ac=AOR-DK-bg-overvaegt`.

sdktsg. sdktsg. *Custom Serialization Example*. URL `http://www.codeproject.com/Articles/22787/Custom-Serialization-Example`.

SolvingMaze. SolvingMaze. *The e-Commerce shipping Calculator demo*. URL `http://solvingmaze.com/demo`.

Top iPhone Application, **february 2011**. Top iPhone Application. *Luggage packing app*, 2011. URL `http://www.topiphoneapplication.com/luggage-packing-app/`.

Zielbauer, **2001**. Paul Zielbauer. *After the attacks*. The New York Times, 2001. URL `http://www.nytimes.com/2001/09/13/us/after-attacks-airport-security-faa-announces-stricter-rules-knives-no-longer.html`.

Pak kufferten A

Problemstilling

Hvis man skal ud at rejse med overnatning, skal man huske at pakke sin kuffert. Det kræver en del overvejelser. Det går ikke at glemme sit pas eller sit undertøj. Men et kuffertpakningsprogram skal kunne klare meget mere end at være en avanceret huskeseddel. Et godt program skal også kunne hjælpe med at fordele bagagen i fordel til vægt- og volumenkrav. På mange flyselskaber er der således krav til den indtjekkede bagage og håndbagagen. Mange flyselskaber tillader 20 kg indtjekket bagage og har krav til dimensionerne på håndbagage. Bestemte slags genstande må ikke forefindes i håndbagagen, andre ikke engang i indtjekket bagage. Hvis man er en familie, der skal rejse sammen, er det vigtigt at fordele indholdet i kufferter, så de mindste medlemmer ikke slæber for meget, men heller ikke for lidt. Også volumen er en udfordring, hvis man skal rejse i bil. Hvilke kufferter og kasser er der plads til i bagagerummet? Og hvordan skal bagagen placeres i bagagerummet? Man kan selvfølgelig prøve sig frem, men det ville være godt at have en algoritme til hjælp. Ideelt set skulle et pakkeprogram være en lille app, som man kan have med sig i sin telefon eller tavlecomputer og let tilgå undervejs på rejsen, når man f.eks. skal finde plads til den nyindkøbte souvenir eller de fem flasker etellerandet.

Problem

Kan man lave et program, der kan hjælpe med at planlægge pakning af kuffert?

Formål

At udvikle et stykke software, der kan være en intelligent assistent, når man skal pakke sin bagage.

Mål

At udvikle et effektivt stykke software, der gør det muligt at pakke sin bagage bedst muligt.

Teknisk-naturvidenskabelige fagområder

Optimeringsproblemer. Analyse af algoritmer. Objektorienteret programmering.

Eksempler på kontekstuelle fagområder

Hvorfor er luftfartsselskabernes pladskrav opstået, og hvordan har de udviklet sig?

Forslagsstiller

Hans Hüttel.

Mindmap B

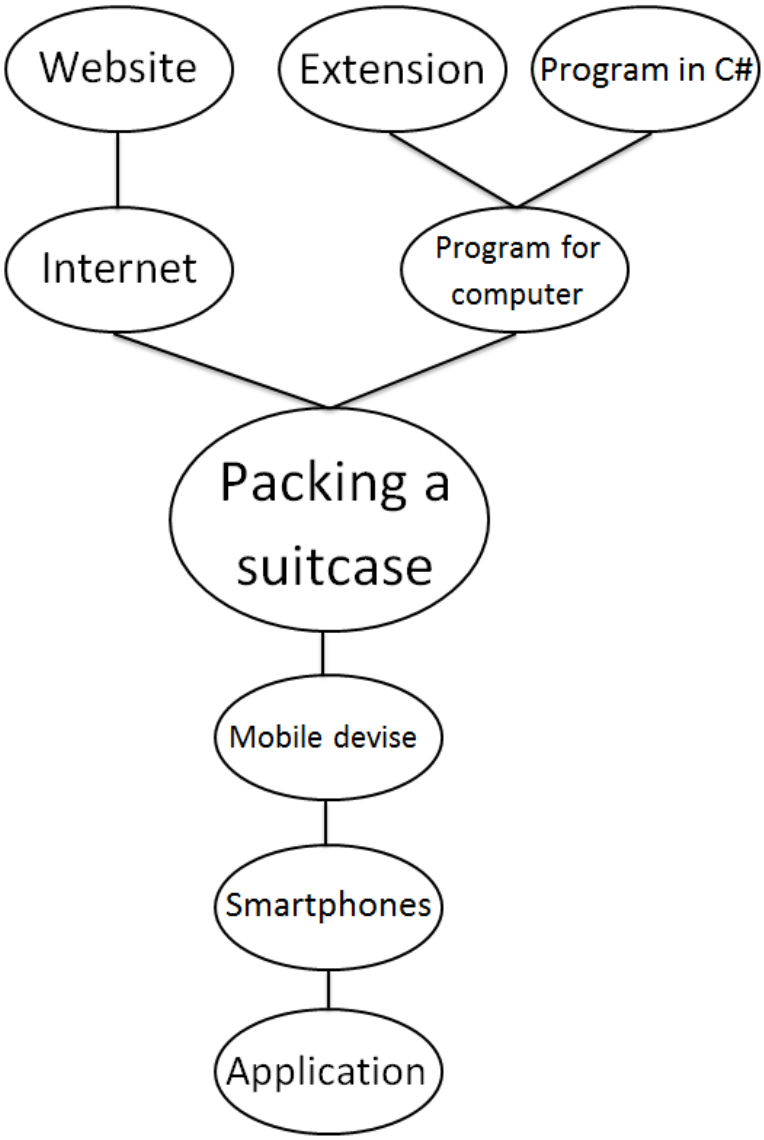

Figure B.1: Mindmap containing solution suggestions

Questionnaire C

1. Were there any problems finding out how the program worked?
2. Were there any technical problems using the program?
    - Other problems?
3. Was the program easy to use?
4. How was the design of the program?
5. Did you find the program helpful?
6. Would you use a similar program?
7. Suggestions for improvements?
8. Other comments?

CD D

This CD contains the program of this project.