P1 Project

Bag Packer Program

Statusseminar

Group B130

Software

Department of Computer Science
Aalborg University

The 28th of March 2012

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

Synopsis:

Title:

Bag Packing Program

Project period:

P2, spring 2011

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: 2012

This report contains documentation of which problems some people might have packing their suitcases before a flight 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#. There will be a chapter describing how the structure of the program will be, and the reason behind the construction. There will also be a section about the reflections made, when the program was being made.
One of the last chapters will concern the testing and conclusion of the finished program. What could have been done better to the program and why does it not work, if that is the case. This chapter will also concern the reflection 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 that 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.

Date: Aleksander Sørensen Nilsson

Date: Christian Jødal O'Keefe

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. We are group B130. The project began on 8.2.2012 and was finished on the 20.5.2012, at the University of Aalborg, where we are studying Software.

Most of us come from Tech College Aalborg, but some come from other colleges. We all chose University of Aalborg because we like the PBL model. PBL stands for Problem Based Learning and it is a unique learning method that is only used at University of Aalborg and Roskilde University Centre.

The goal with this project is to give us skills in using the programming language C# (C-Sharp), and make us better at team-working. This project is about optimizing and analysis of algorithms, and our main question is: How can a program be developed which helps the customer through the progress of packing one or more suitcases the most effective way by size and weight?

Contents

1 Introduction

FiXme Fatal: ma

Problem Analysis 2

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 in account that the suitcase must not exceed the maximum weight limit of some types of trips, and some people might get fined for packing a suitcase to heavily. 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 the "Why there is a maximum weight on suitcases when one is traveling"-question.

2.1 Problem of packing

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

When going on a vacation or a business trip packing a suitcase is normally needed. You need all you personal items and clothes with you. The length of the stay, the purpose of the trip, and the number of people traveling together effects 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 time spend away affects how much clothes the traveler will need to bring and/or if the traveler will need to wash the cloth while away. A longer stay means of cause that the traveler will need more clothes. This affects the size and weight of the luggage because you will need more or fewer items with you on different trips.

The amount of items to bring also depends on what kind of trip it is and the purpose of the trip. For example if the purpose of the trip is a sport competition, you might need to bring some equipment, you would not normally bring on a holiday trip or other kind of trips. If it is a short business trip all you might need could be a briefcase with your papers and a suit to wear.

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 that has to be spread in different suitcases and need to be found afterwards and it might also effect the total weight of the luggage.

It can be frustrating, when traveling with many and/or heavy suitcases, if you need to carry the luggage far. It can also be hard to maneuvre if where you are suppose to walk is a small space or a narrow passage or if simply filled with a lot of people constantly blocking your way.

There can also be a problem with too large or 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]. One of the reasons that this fee has been introduced is so the airlines bag-handlers do not risk getting injuries by carrying too many, heavy suitcases.
To avoid this fee it can be an advantage to pack the suitcases properly and limit the choice of items to bring so the total weight does not exceed the allowed weight.

2.1.1 Packing

Because of the weight and size restrictions at the airports it can sometimes be difficult to get all the items you want to bring into a suitcase that is small enough and not exceed the allowed weight. It can sometimes be necessary to acquire more suitcases for the trip or leaving some of the items you wanted to bring. Bringing more luggage means you have to pay more to get the extra luggage with you on the plane. A way you might be able to avoid buying and paying for extra suitcases is to pack the suitcases more than the suitcases are meant to contain. This mean that the suitcases packed more compact. This on the other hand increases the bags weight [SAS, 2012]. 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 bags might exceed the limit for weight and therefore trigger a fee for overweight luggage. It seems people often pack their luggage more compact instead of taking extra bags with them on vacation. Generally people take a lot with them on vacation and they might not have packed their luggage the most effective way [of Transportation Statistics , RITA].

Airline	First quarter	Second quarter	Third quarter	Summery for the three quarters in 2011
Delta	197,971	226,291	232,508	656,770
American	137,210	156,114	152,750	446,074
US Airways	120,925	134,752	128,761	384,439
Continental	76,304	91,332	94,301	261,937
United	66,245	71,111	74,758	212,114

Table 2.1: This table contains the top 5 of the companies that have collected the most fees regarding luggage in 2011. The numbers are from [of Transportation Statistics , RITA] and the fourth quarter is not released yet, therefore missing

An American website for statistics, [of Transportation Statistics , RITA], shows the amount of fees given regarding luggage that have been registered at the U.S. airlines. Table 2.1 is a segment of the statistic table found at [of Transportation Statistics , RITA]. Through this it is possible to see that there are people that exceed the set of limits given by the airline. A note regarding this source is that the size of the fee is a combination of

the different rules and related fees. Therefore the statistics do not give an accurate image of the problem with 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. But flights is the one transport where it plays the biggest role for the traveler because it has economical consequences. With train and car it is more or less up to the traveler how heavy the luggage is allowed to be. There are restrictions to how big the bags are allowed to be in the train. For train 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 you cannot just pay to get more luggage with you than fits into your 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 that 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 and thereby makes 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 the 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 item in a more effective way. This is done by using all the bags in the most effective way and thereby not take the ownership of the bags into consideration. This method means that the bags will not be sectioned but the packed item will be packed into different bags despite the ownership of the item to active the most effective way of packing. Original bags tends to be packed so the bags 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 known were to find the desired item. When this method of packing is used, it will be more effectively packed, but it have to be kept in mind that the weight should be distributed evenly, so none of the luggages are too heavily packed compared to an other.

Round 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 to the trip home due to the bought souvenirs or other items. The reason to 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.

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 that exceed the size limits might not be allowed to have their luggage with them if the dimensions of the suitcases are to big. There is not a specific program on the marked that specifically helps packing luggage for a trip, but there exist at least one program that can pack a container[SolvingMaze]. This method of packing a container can be compared to packing luggage. A other solution is a smart-phone application that help with packing a suitcases by making a check-list over what to have with you on the trip [top iphone application, 2011], see section 2.4.

2.2 Luggage rules

This section will focus on the general rules regarding luggage when going abroad, whether by plane, train or cruise ship.

2.2.1 Luggage table

This section will contain a table which displays the various limits for luggage in different public transportations. The table will be used to give a general overview of the different limits for luggage for different types of transportations.

Type of luggage	Dimension limit	Weight limit
Check-in luggage(Airplane)	158 cm *	20-23 kg
Extra luggage(Airplane)	158/277 cm *	20/45 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.
* The sum of the dimensions; height, weight and depth must not exceed the given value. Depending on how much is paid on extra luggage different dimension and weight limits are given.
** There are no set limits, it just have to be carry able and not be a bother for other passengers

2.2.2 Charter trips on air planes

Given below is the various restrictions when travelling by plane.

Checked-in luggage

Check-in luggage is the luggage that 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, 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 airport.

Carry on

Carry on luggage is what the passenger is allowed to bring aboard in the cabin.
Approved items:

- 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 have to be closed (1 litre bag per passenger).
- The bag have 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 have 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, have very few rules regarding the luggage you are allowed to bring with you.

The only rule is that your luggage need to be able to lie on the luggage rack or under the seat and not be bothering 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 you are on. They have no other rules regarding luggage [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 passengers luggage should be kept in their cabin during the trip[MSCcruise].

2.3 Luggage allowance

Due to the hijacking and crashing of the airplanes into the World Trade Center on the 9th of September 2001, the security of airports have 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 have 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 chapter are used to research the marked and thereby get a image of what solutions there already exist on the marked. Through the research it is also possible to determine, how the existing solution helps the user with the stated problem. By looking at existing solutions it can used to determine what features that 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 offers help and provide tips for packing for traveling. Some of these lists and guides have been developed into applications that are available for the customer to use. An application or in short app, is a program that fulfill different kind of services for the user. Apps are used in the web browser, computer, smart phone, and tablets. The term app is general mentioned in context to smart phones and tablets.

There also exists programs, that have integrated algorithms to handle optimization of the packing, on the marked that can be used. First a look into these lists and guides and the more advance solution thereafter.

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 around.

Figure 2.1: Picture of 2 of Packing Pros menus from [top iphone application, 2011]

Packing pro is designed with a panel in the bottom of the screen that allows the user navigate through the menus. Packing Pro provides the user with a help menu that contain information on how to use the app. On figure: 2.1 can an example of how a check list could look like.

These templates are designed to different purposes regarding the customer, gender, type of trip, and purpose of the trip. The customer can then load the wanted template for the purpose. The user also have the possibility to create their own lists by adding things that should be remembered for the trip by them self. And to select an existing list and delete the objects that were found irrelevant by the user. The user can then check object on the lists off as it 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 have to be bought before it can be used [top iphone application, 2011].

Packing Pro works as check list and help the user remember what to pack, but it does not preform any organization of luggages content itself. So Packing Pro itself does not solve the described problem but helps the user remembering 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 compatible with Iphone/Ipad operating system. It would be nice if the program made work cross platforms but it is still not required to solve the problem.

2.4.2 App - Checkmark Packlist

Checkmark Packlist is an free app for the smart phones running the Android system. Checkmark Packlist offers different templates for check lists that the customer can use. One of these templates is the list for packing for a trip. That way the customer can select and use this template for remembering what they will need to pack for the trip. Checkmark Packlist uses touch to navigated in the program. This means that the customer with their fingers can navigated through the check list and check off things that have been packed.

Figure 2.2: Picture of the Checkmark Packlist in action from [GreenbeanSoft, 2010]

On figure: 2.2 there can be seen an example of the product and how Pheckmark Packlist looks like for the user. Checkmark Packlist does not provide customization tool that let the customer add more categories to the check list. This is only a featured provided in the paid version of Checkmark Packlist [GreenbeanSoft, 2010].

This app does not provide a solid solution for luggage packing but helps the user remembering what there should be packed for trip. This check list feature give a guiding effect and this is a useful feature and can be used in the product design to solve the problem. A feature to consider is the mobility by designing the program for hand held devices.

2.4.3 Online check/tip list

The online check list works as a reminder when packing luggage. It also give tips and tricks that 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 as computers, tablets, and smart phones that 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 customer 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 packing, instead it helps with the planing of materials that the user might want to have on the trip. The website is designed with a menu left that let the user navigate through the different content of 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 planing 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. Instead it would more be focus on helping the user with the packing.

2.4.4 The e-Commerce shipping calculator

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

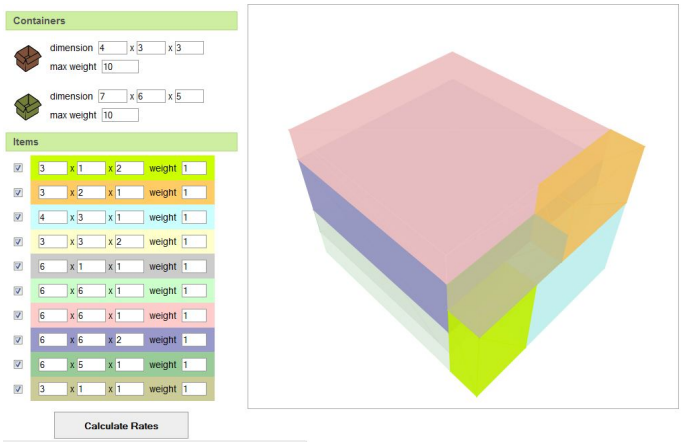

Figure 2.3: Screen shot of the program running taken from [SolvingMaze]

The customer have to type all known data in and press "Calculate Rates" and the program will then form a 3d model, this can be seen on figure: 2.3. This product have a number of useful features that can be used in the final product. This solution can take items dimensions and weight and calculate the most optimal placement in the container. This can be related to packing your luggage for a trip.

2.4.5 Recapitulation

This sections main object is to look at the wanted features and recap them. Packing Pro and Checkmark Packlist is similar in the way that they help the user. They both provide the user with a sort of check list that can be check off and thereby help the customer remember what have not been packed yet. One of the differences is that Packing Pro

have a price while Checkmark Packlist is free. But this difference means that Packing Pro allows the user to edit the check list while that are not possible in Checkmark Packlist. These two apps do not help the user arrange the luggage content or take in consideration of size and weight of it. Thereby is two apps do not help people with the packing itself but more what should be remembered for the trip. The online check/tip list provide the user with advices for the trip and what to pack. Advices are great to get a general idea of what to take pack but it still does not give a more effective way to pack. The e-Commerce shipping calculator is the one solution with the most wanted features. One of the strong feature that can be used is the ability to calculate the most effective that a container should be packed. A important side note is that the intentions is not to pack bags but the feature can be related to packing content of a bag.

Included in product **Solutions**	App - Packing / Packing Pro	App - Checkmark Packlist	Online check/tip list	The e-Commerce shipping calculator
Guide the user*	x	x	x	x
Distribute weight				x
Distribute space				x
On the road	x	x		x
Where in the suitcase				x
Packing list	x	x		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 vertical and existing solution on the market listed horizontal. The crosses indicates when the particular product have the particular feature. The purpose of the table is to give a overview of the products and their features that was found essential to the problem. Table 2.3 shows tha 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 Thesis Statement

In this section a thesis statement will be formulated which will be used to develop a method to the problem of this project and to get a more precise problem to work with. The method is used to try and solve the problem stated in the thesis statement.
The problem analyses shows that there are two general types of programs on the market. One type is a form of a packing list that tells the user what they might want to pack,

the other type of program is a packing program which packs containers for the user and calculates the shipping cost to a designated location. These two types of programs only fulfill parts of the criteria this project have put forward. With this in mind a thesis statement have been formulated to help shape the solution for this projects problem.

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

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

Sub Statements

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

- **How can it be checked that the weight in the suitcases are evenly distributed and it 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 must most effective solution for the given data. It will also have to check that the solutions weight does not exceed the given limits or the volume of the luggage.

- **Which functions 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 optimising and thereby the program will fulfil its purpose.

- **How should the program communicate with the user and inform where the items are placed, and tell how much space is left?**

  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.

- **How can a program be developed so it takes the length of the journey into account, and enable the user to update the program on the go?**

  The program should also be developed so it is possible for the user to update the programs database while on vacation thereby get a packing order for the new content of the bags.

From this thesis 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

This project structure will be based on Aalborg PBL (problem based learning). The Aalborg PBL is a method whereby the learning process lies in the work with a problem and try 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 give them tools to handle the processes that goes 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 thesis statement is formed and is used to produce a list of product requirements. The requirements are then used to design and develop a product that should solve the problem stated in the thesis statement. The design will be describe in chapter 4. The development will also have it own chapter were the program will be describe and how the different functions are made. This can be seen in chapter . The program are then tested on the target group of the problem. 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 thesis 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 document 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, article, 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 are general used when using others materials as documentation. Therefore it is also a relevant method to use when using sources in the project work.

Theory 3

In this chapter there will be taken a closer look on the theoretical aspect of writing a sorting program. First off is a quick description the NP-problem followed by a look on different ways to pack different objects. 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.

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 can be formulated as the solution to the following linear integer programming formulation:

$$maximize \sum_{j=1}^{n} p_j x_j$$

Meaning: Maximizes the total value (p) of items (j) in knapsack

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

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

Meaning: Total weight (w) of items(x) may not exceed the knapsacks capacity (c)

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

3.2 Bin packing problem

The bin packing problem is a combinatorial NP-hard problem [Korte and Vygen, 2002]. This means that the problem is at least as hard as an existing NP-complete problem. NP-complete means, in simple terms, that every combination must be examined, and then the best combination must be chosen. An example is the Traveling Salesman Problem (TSP), where a list of cities and distances between the cities is given. The problem is then to find the shortest route to visit all cities, and to end at the starting point. 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 solve the bin packing problem. Bin packing problem is focusing on bins instead of suitcases but they are basically the same, the only difference is the size. Some of the popular methods will be described in the following section. To describe these packing algorithms, illustrations will be used. The illustrations shows how the packing algorithms works in one dimension - but it gives a nice basic understanding of the algorithms. Figure 3.1 is an illustration of the unpacked elements:

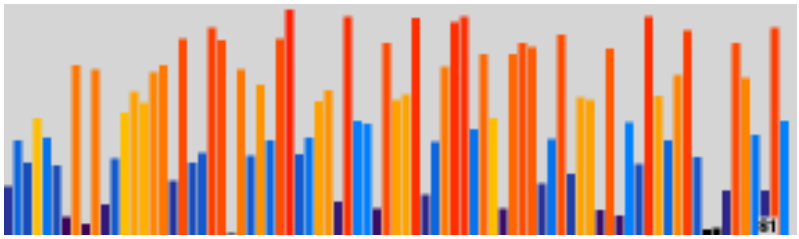

Figure 3.1: Initial elements (source: [bin])

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.

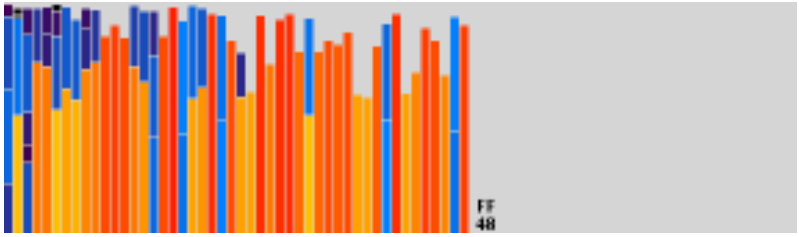

Figure 3.2: Elements after FF has been applied (source: [bin])

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 fit. 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.

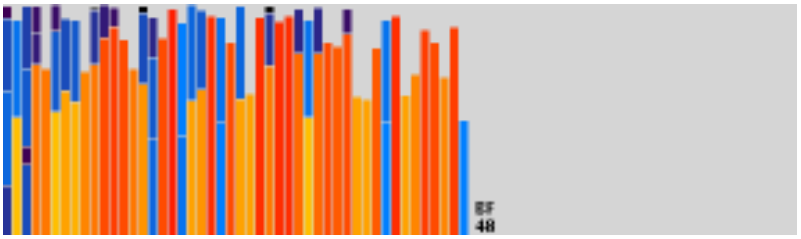

Figure 3.3: Elements after BF has been applied (source: [bin])

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.

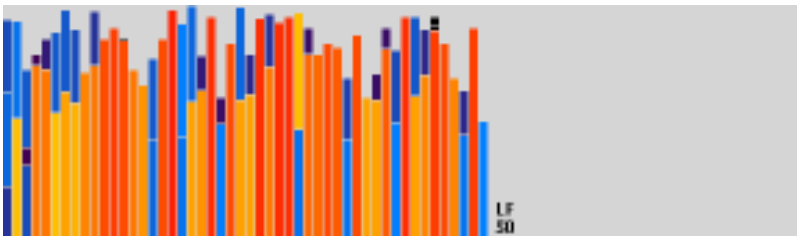

Figure 3.4: Elements after LF has been applied (source: [bin])

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 suggest, 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.

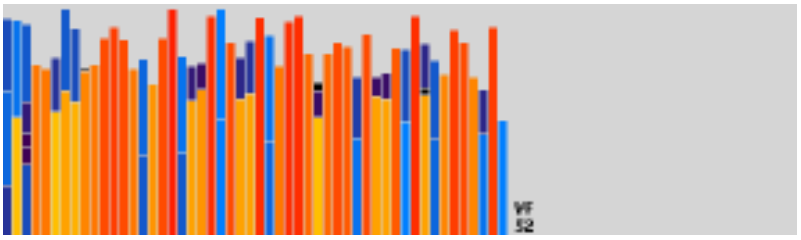

Figure 3.5: Elements after WF has been applied (source: [bin])

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.

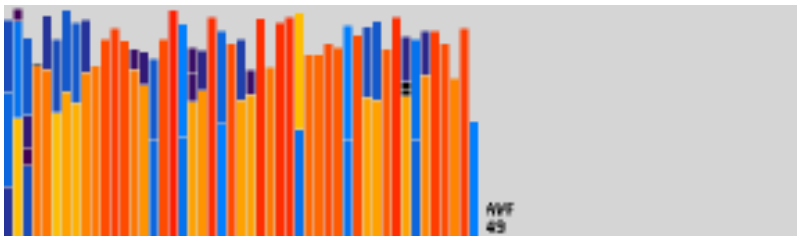

Figure 3.6: Elements after AWF has been applied (source: [bin])

3.2.6 First fit decreasing(FFD)

The above algorithms 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 algorithms 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.

3.2.8 Round up

It seems that it is more effective to sort the lists 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, and use the others mathematical experiences as a benefit.

e Fatal: Et eller
det sted skal der
enlignes mellem
ing og knapsack

Design 4

In the design chapter a look will be taken on the specification requirements and why those features are needed or wanted in the program. There are also a section detailing some features that will be nice to have if there is enough time to implement them in the program.

4.1 Specification Requirements

Through the problem analysis it has been documented that there are some strict rules regarding some forms of public transportation when going on vacation. Based on this research a list of features have been composed, that the program must fulfil to meet the base requirements. Furthermore another list have also been made composed of some additional features that would make the program better and more user friendly. They are not needed for the base requirements, but rather as improvements to further make the program ideal for the user.

For the user to better get started on the program there will be a guide that come with the program. The guide gives an explanation on how to use the program. The guide will be short and well formulated so the user with ease can read and understand the guide. The project description states that the program language must be written in C#. The program itself needs to have a few features for it to solve the problem that is the focus of this project. The program needs to make sure that the weight is evenly distributed in the bags and that it does not exceed the bags weight limits. The program also needs to distribute the space of the bags to make sure that the program does not fill a bag more than there is physical room for. When the user is on the trip the program needs to have a function that allows the user to edit the list over items that are in the bag so if the user buys some souvenirs or throws something away, the list of items will be updated and thereby a new way to pack the luggage. The program will need a function to help the user see where the items are placed in the luggage. The program will also have to check that the suitcases are below the limits set for weight and size.

There are some features that not are essential for the program to work but will improve the program. One of these features is to handle changeable shape of items e.g. a T-shirt or other forms of clothing. This makes the program able to pack more efficient. This means that to program can handle like solid, liquid and bendable shapes. But this may not be in the program at the start since this will be hard to develop and implement. To better help packing and planing ahead the program needs a list of different trip types that can help the user with packing the luggage for a given type of trip. Another nice feature to have is

to save space for possible souvenirs the user might buys on the trip. These features means that the user does not need to check if there is room for the souvenirs before buying it.

4.1.1 Targeted Features

These are the essential features that the program will have.

Program language is C#: The program need to written in C# since the requirements for this project is that the program need to written in C#.

Guide the user: The program will have a little "readme" file, or other form of guide, that will tell the customer how to use the program.

Distribute weight: The program must be able to distribute weight of 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 of the program, is that it should be able to tell the user how to pack the suitcase, and be able to tell if there is enough space for eventual souvenirs. Lastly it should inform the user how much space, if any, is left.

On the road: The program will be able to tell you, while you are on the trip, if there is enough space for a souvenirs, if you input the dimensions and weight of that item. And if you what to remove a item from your luggage the it can this as well.

Baggage rules: The program will need to know basic baggage rules. For example the luggage must not weigh too much, and it must be below certain dimensions.

Structure of packing: When the user asks the program if an item will fit in the suitcase, the program will show exactly where in the suitcase the item will fit.

Packing list: To make it easier for the user to know what will be packed an editable lists will be included depending on the type of trip.

4.1.2 Optional Features

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

Solid/liquid/bendable shapes: The program will 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 nec-

essary because each trip might require different items.

4.2 Solution suggestions

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

4.2.1 Application for smartphones

This idea is to make an application that helps the user pack one or more suitcases. The user will need to put in the height, length, depth, weight and a name of the items to pack. The application will then calculate if all the items can be packed. All of these calculations means 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 requires learning how to write applications for smartphones and therefore this solution requires modification to the time schedule. This solution is easy to bring everywhere because you rarely leave your phone. On the other hand people without a smartphone would not be able to use the program.

4.2.2 Extension for an existing program

A second solution could be to make an extension for another program that already exists. This extension should add the missing functions of the original program. It could either 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. Another solution could be an extension to the e-Commerce Shipping calculator. This extension should be able to also 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 not able to work with a program written in C#. Another problem is testing the program if the other program is not open source and therefore the company's permission is needed before the testing can begin. If the existing program is not open source the company's permission and cooperation will be needed in the making of an extension of their product. The testing is important to be sure that the programs are able to work together as planned. On the other hand it is possible to make a program that focus more specifically on what is missing in the other program and therefore cover more of the important features. So this solution would need the company's permission to make use of their code for testing 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.

4.2.3 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 a 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. This will be 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 display figure 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 user wants to use it on-the-road.

4.2.4 Choise of solution

When understanding the three solutions it is possible to determine, which of the solutions best solve the described problem. 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 that 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 own the program then needs to give permission to use their code. If the original program is not open source can it be a problem to get this permission without long negotiation with the company. 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 calculation 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 most pros and fewest cons. It can be a problem on the go, but every solution has a problem on the go - the app will need internet access, and both the program needs a computer. Many people have a computer on a vacation, so the problem is not that big. Also more time is not needed, which means there are more time to make the program and thereby more time to finish the program before the deadline.

4.3 Program planning

This section is to plan how the program should work and describe the flow of the program. The program will be described in a flowchart 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. Pseudo code is used as a schematic for the program and thereby gives some foresight into any problems that can be encountered when writing the actual code. 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.

The program then performs the algorithms to place the items in the most efficient way regarding volume and weight. The program will also check that the suitcases does 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 can not fit the item in any of the accessible suitcases the item will be marked as not packable. If the program reach 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 have to pack the suitcases and report if there were any items that 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.

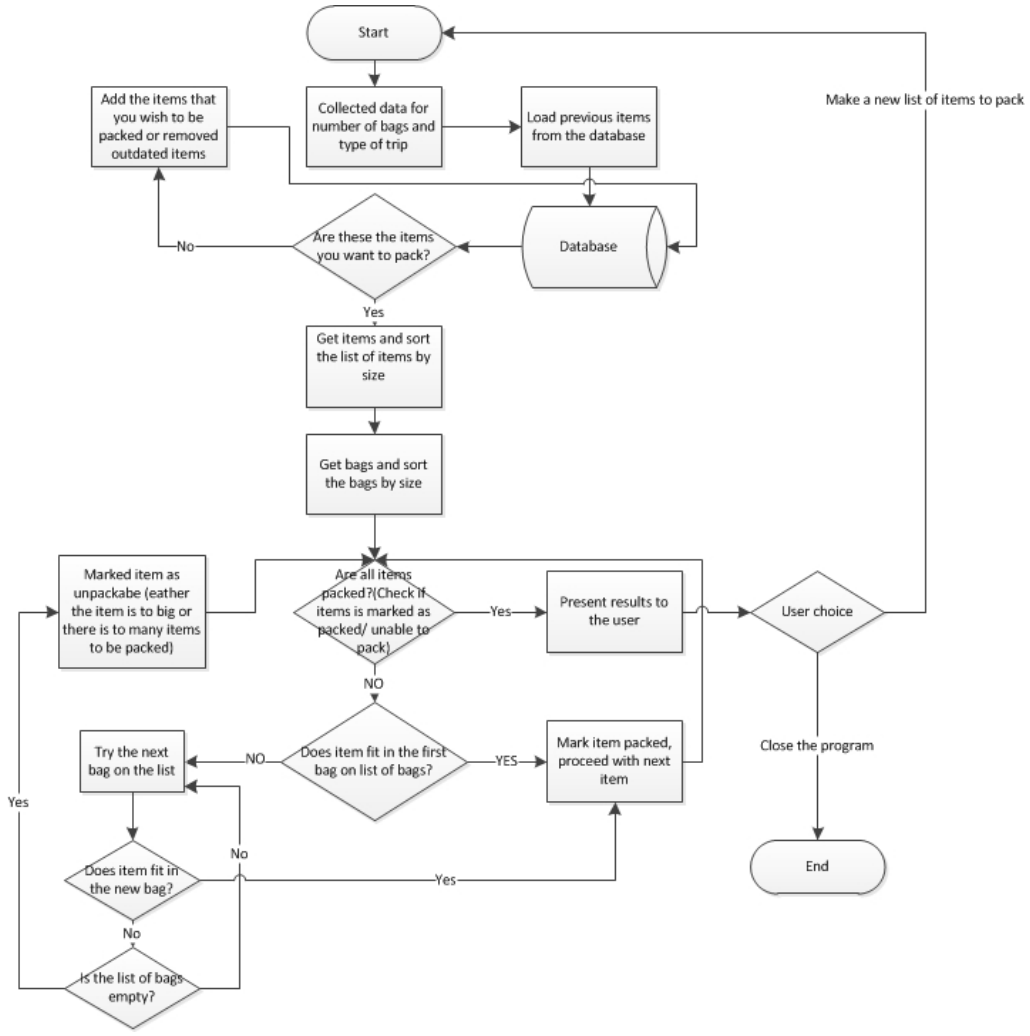

Figure 4.1: This is the flowchart of the program (not quiet finished yet)

Fatal: Finish the flowchart!

Hereby the general structure of the program has been formed and can be described by a flowchart, seen on figure: 4.1. The arrows shows the direction of the flow in the program. Some of the arrows also have small labels indicating which answers there were 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.

4.4 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.

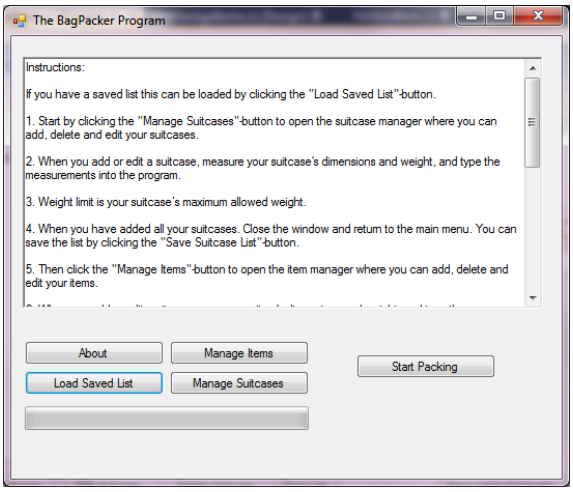

Figure 4.2: Main window

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" and "Manage Suitcases" button where the user can add/edit/delete items/suitcases. The button "Load Saved List" can be used if the user have 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 program's algorithm and packs the suitcase(s). The progress bar is associated with the "Start Packing" button and starts when the user presses the button.

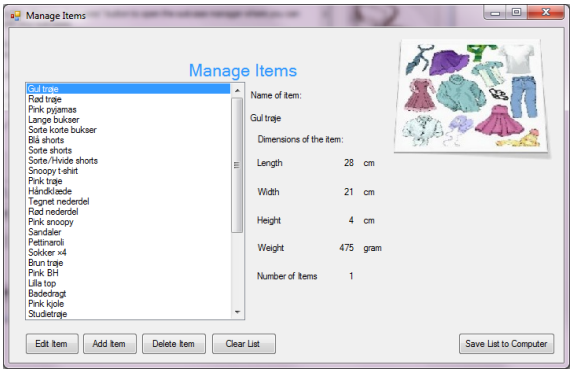

Figure 4.3: Manage items window

The Manage Items menu is where the user can add, delete and/or edit items.

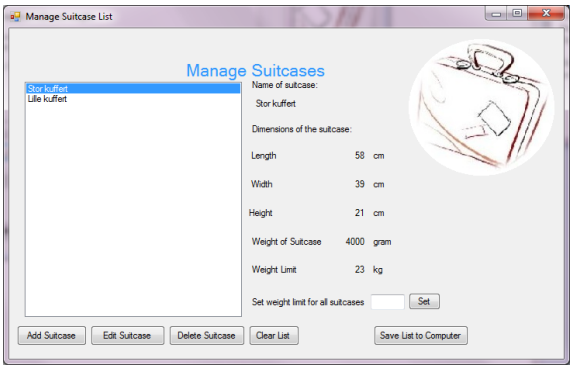

Figure 4.4: Manage suitcase window

Add/delete/edit suitcases in the Manage Suitcase menu.

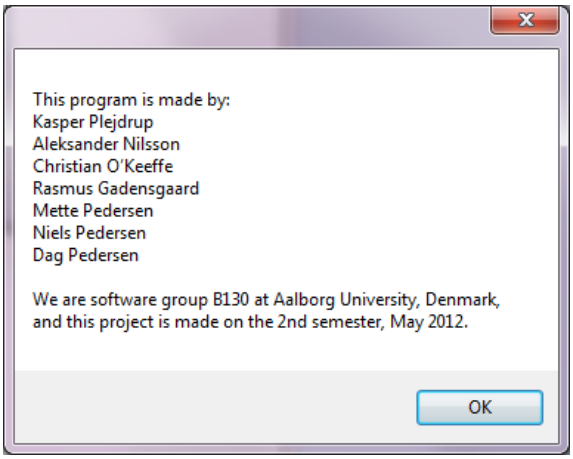

Figure 4.5: About window

The About button which tells who the programmers are, and when the program was made.

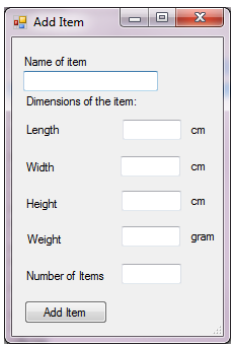

Figure 4.6: Add item window

When the user presses the "Add Item" button in Manage Items, this form shows. In the 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.

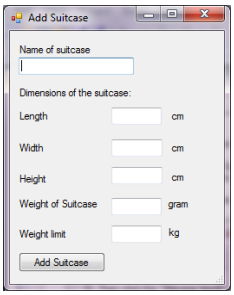

Figure 4.7: Add suitcase window

When the user presses the "Add Suitcase" button in Manage Suitcases, the user can add a new suitcase. The data needed is the length, width, height, weight, and the maximum weight of the suitcase.

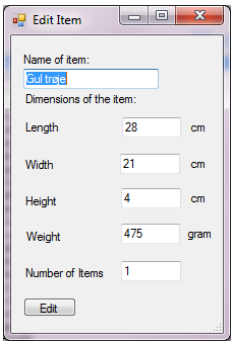

Figure 4.8: Edit item window

If the user wants to edit an item, he/she can press the "Edit Item" button in Manage Items, and this form shows. 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.

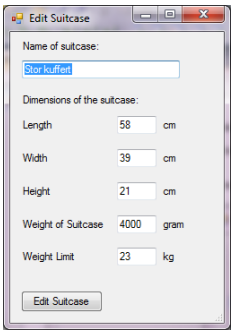

Figure 4.9: Edit suitcase window

In "Manage Suitcase", there is a button called "Edit Suitcase". 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 do not have the same measurements.

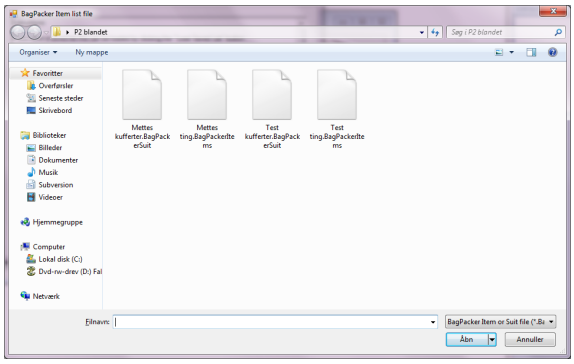

Figure 4.10: Load List

If the user already have used the program before and have saved an item list and a suitcase list, both can be loaded here.

4.4.1 3D-viewer

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

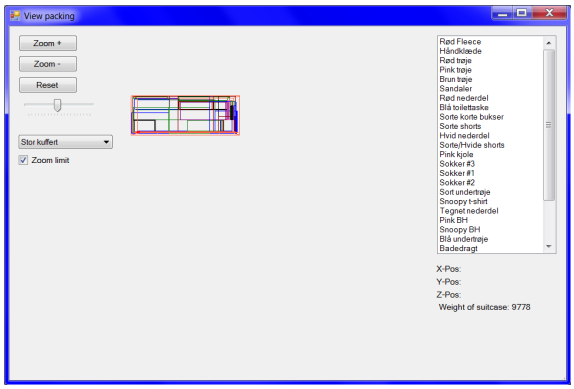

Figure 4.11: 3D Viewer window

This is the first thing the user will see when the packing starts. 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 xyz-points to see where the item is supposed to be placed, and the current suitcase's weight.

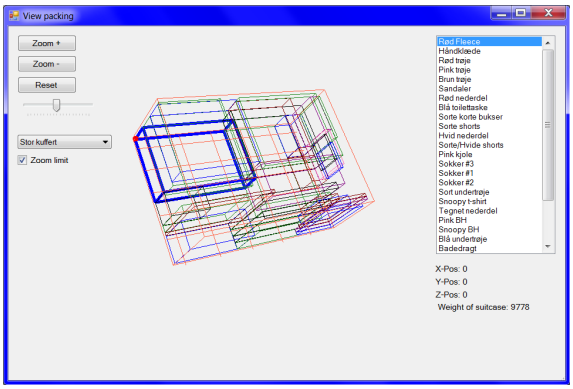

Figure 4.12: 3D Viewer2 window

On the left side of the window are two buttons, zoom in and zoom out. 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 on 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 like 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.

5 Development

In the following section the program will be described. Each function is described in its own part to make it more clear. There are some parts of the program that will not have a description, simply because code description will not be in focus here, but this will merely tell what it does and how it works.

5.1 How the program handles different forms

The program is aimed at families who pack their suitcases together, and are not bothered by having other than their own items in their suitcase.

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 that opens the new form, is the frm3D.ShowDialog(this); This can be seen on Listing 5.1.

```
1 if (finish_packing)
2 {
3     frm3DViewer frm3D = new frm3DViewer(luggage_items_to_pack,
          luggages_to_pack, this);
4     tmrCheckForFinishPacking.Enabled = false;
5     finish_packing = false;
6     frm3D.ShowDialog(this);
7
8 }
```

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 that 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.

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

```
1 public frm3DViewer(List<frmMain.luggage_item> input_lug_items,
      List<frmMain.luggage> input_lugs, frmMain input_form_main)
2 {
```

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

5.2 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 has 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 code will try to place all the items in the luggage so it is packed most optimal in both size and weight.

5.2.1 Optimization of weight

The optimization of the weight is done, so no luggage exceeds 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 5.1, where N = Number of items.

$$\text{Avarage Weigt} = \frac{\sum_{i=1}^{N} I_{weight}}{N} \tag{5.1}$$

It is possible to distribute the weight evenly in the suitcases, when the optimal weight for each suitcase (the average weight calculated by equation 5.1) 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.

5.2.2 Description of the algorithm

The algorithm use the First Fit Decreasing (FFD) method to pack the items in the luggage. This means that the algorithm at first sorts the items by size. It will start by packing the largest items first, which will give a better result for the packing. When the list is sorted it will then also sort the list of luggages by size. The algorithm can now go to the actual packing.

The general algorithm is:

- Sort the items by size.
- Check if the item can be in the first luggage, while the luggages total weight does not exceed the average weight per luggage.
- Check if the item can fit in the luggage, else check the next.

- If the item cannot be fitted in any luggage, 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 good enough detail. To explain the algorithm, a flowchart can be seen on figure 5.1.

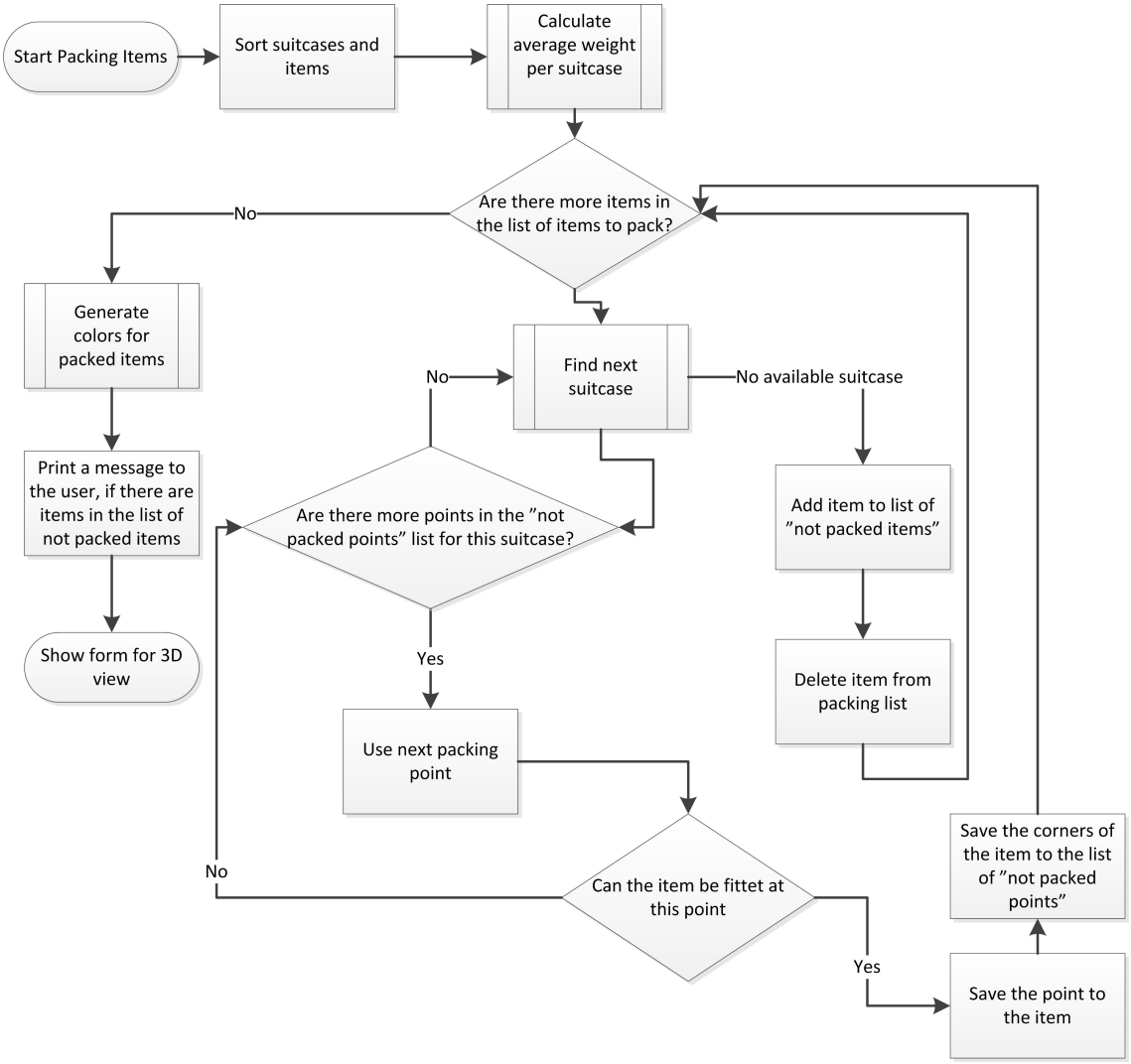

Figure 5.1: The flowchart for the packing algorithm

As seen on the flowchart, 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 make 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 have 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 effective 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 is packed and those which is 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 points are packed, it will generate the colors for the items. The items must have a color, but no color must be the same 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.

5.2.3 Round up

This is the theory of the algorithm to pack 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 FFD and Best Fit will ensure an equal distribution of weight and space at the same time.

5.3 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 all the item's coordinates into a list, this is done for all items that are to be packed. First off a "for" loop runs through the items that are to be packed. Inside the first "for" loop there are three "for" loops that run through the x, y and z values starting at the saved coordinate for the current item. The loops then runs until the limits are reached for the three "for" loops which are the height, width and length. These loops can be seen listing 5.3. At the end of the three "for" loop there are 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 that has 3 lists in it. These three lists outlines the sides of their respective object. The temporary variable are then stored in the "BoxColor" list which is the final list.

```
1 for (i = 0; i <= Number; i++)
2 {
3     tmp_list = new cbox();
4     /*Loops that goes through the 3 koordinates*/
5     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++)
6     {
7         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++)
```

```
8           {
9               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++)
10              {
```

Listing 5.3: The four "for" loops which controls the item and the (x; y; z) value that make 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 works with the points that are in the items lists that are being worked with. But it will only run these loop if the items are in the same suitcase (Which can be seen by the "if" sentence that are between the two first and the two last "for" loops). If they are next to each other it will save the items number into a list in the other item. This is done for the other item which is being checked as well. The 2 loops and setting of a items neighbor can be seen on listing 5.4.

```
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 5.4: 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 work with one "for" loop that 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 that goes through all the neighbors (items that are next to it) and check if the item itself is in it and remove it. It will also check if the number after the current is the same, if it is it will remove one of them. This can be seen in listing 5.5.

```
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 5.5: "If" statements that checks the item after the current item to see if they are identical if they are remove one of them. The 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 that 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 that goes through all the item's neighbors and check if the color is the same. If they are the same, then the item's color number is increased by 1. This can be seen in listing 5.6. 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 5.6: "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

5.4 Function to draw cubes

The function called "DrawSomething" is the function which basically converts dimensions and coordinates into the vectors that are to be drawn. "DrawSomething" takes 11 input parameters. Simply put, 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 that 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. The Vector is essentially a dummy-vector, because the polygon class requires a vector in its constructor, but the Vector is set to zero in the class.

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 coordinate is 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 0,0,0 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 suitcase's 0,0,0 point.

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 item's dimensions and half of the suitcase's dimensions, 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 x 0.5 x 0.5 at the suitcase's 0,0,0 point.

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 is 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 was 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 suitcase's width. This is again because the 0,0,0 point of the drawing must be in the middle of the suitcase. The y and z point of the first vector is 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 50x100x80 (height x width x depth), an item with the dimensions 40x70x30 that resides in point 0,0,0 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 has 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 to see how the function "ScaleSize" works.

FiXme Fatal: eSizefunctionref

These calculations are shown on Listing 5.7

```
1 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 5.7: Calculate the points of the first vector

From the calculations above the first Vectors coordinates are as follows (-50, 25, 40), and can be seen on Figure 5.2.

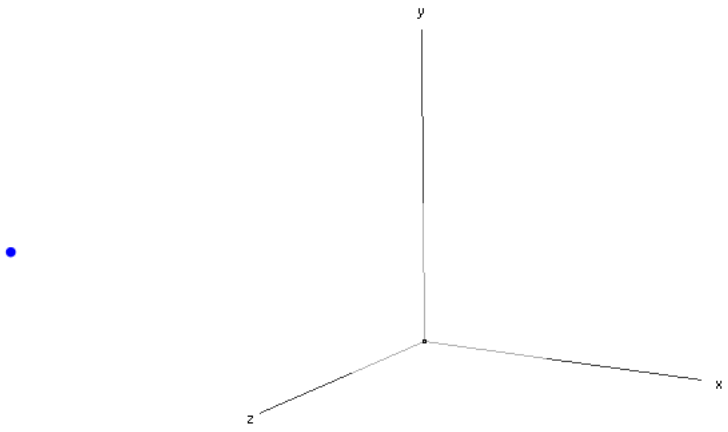

Figure 5.2: First point of the first side of the example item

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 item's y-coordinate 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 5.8.

```
1 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 5.8: Calculate the points of the second vector

The second Vector thereby becomes (-50, -15, 40), and can be seen on Figure 5.3.

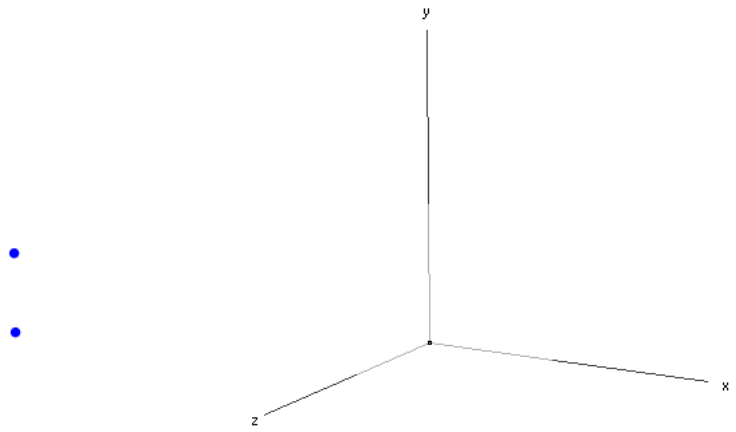

Figure 5.3: First point of the first side of the example item

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 the 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 5.9.

```
1 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 5.9: Calculate the points of the third vector

The third Vector thereby becomes (20, -15, 40), and can be seen on Figure 5.4.

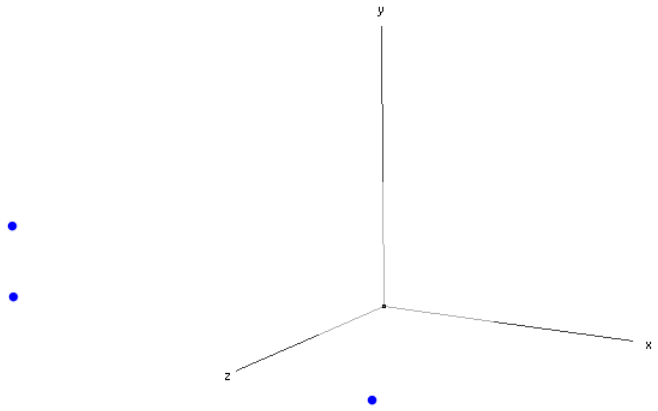

Figure 5.4: First point of the first side of the example item

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 5.10.

```
1 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 5.10: Calculate the points of the fourth vector

The fourth Vector thereby becomes (20, 25, 50), and can be seen on Figure 5.5.

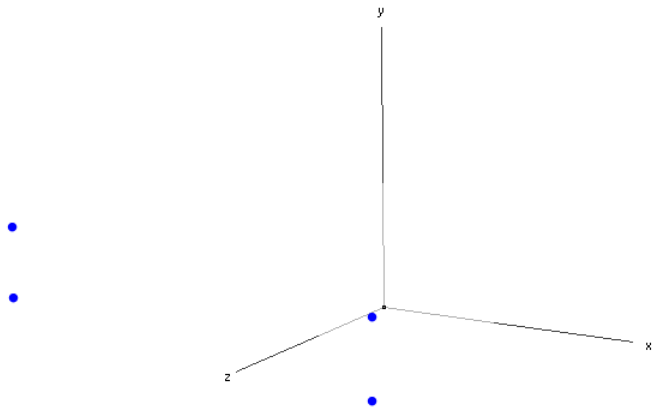

Figure 5.5: First point of the first side of the example item

By drawing a line between these four points, it is clear that these forms a square, that is the first side of the item in the example. This can be seen on Figure 5.6

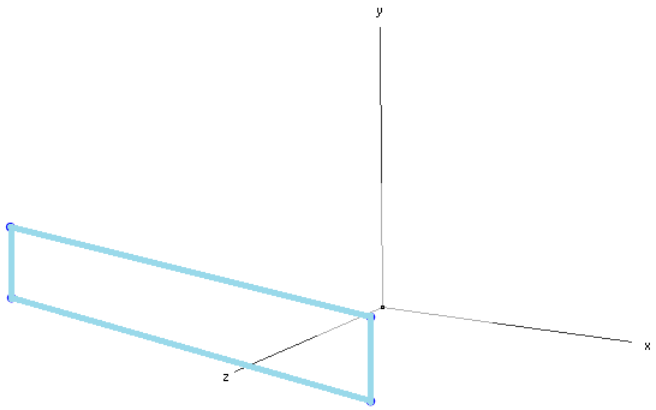

Figure 5.6: How the first side of the example item looks

When the four Vectors has 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 5.11

```
1 poly_array[poly_counter] = new Polygon(1000, z, points1);
2 poly_array[poly_counter].MyPen = new Pen(itempen.Color, itempen.Width);
3 poly_counter++;
```

Listing 5.11: Calculate the points of the fourth vector

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

5.5 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 , 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.

FiXme Fatal: ref draw_cube

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 5.12

```
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 5.12: 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.

5.6 File Serialize function

To save the data in the program a function in C# 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 "www.codeproject.com" from the "how to" "Custom Serialization Example" section. The next part of the function, is where the list of items and the luggages are loaded in the main part.

```
1 info.AddValue("weight",this.weight);
2 info.AddValue("width", this.width);
3 info.AddValue("height", this.height);
4 info.AddValue("depth", this.depth);
5 info.AddValue("name", this.name);
6 info.AddValue("number", this.number_of_items);
```

Listing 5.13: Informs that there is saved to a file

Here 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 a kind of sentence, where the first part is the name of the certain information, which needs to be saved into the file.

```
1 this.weight    = info.GetInt32  ("weight");
2 this.width     = info.GetInt32  ("width");
3 this.height    = info.GetInt32  ("height");
4 this.depth     = info.GetInt32  ("depth");
5 this.name      = info.GetString ("name");
6 this.number_of_items = info.GetInt32("number");
```

Listing 5.14: Informs the user that data is loaded from a previously saved file.

For it to open the data file the program also needs to have data description on how the data is saved in the file. 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 that is saved.

6 Testing

This section will describe the tests, the finding from the tests and the improvement on the program that came from the finding of the tests. To help ensure the program do not have any obvious bugs, test were performed on the program which can be used to help find those bugs. Therefore it is better that a person that is not a part of the project group is going to be the one testing. 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.

7 Discussion

8 Conclusion

Bibliography

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

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

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

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.

GreenbeanSoft, **2010**. GreenbeanSoft. *Checkmark Packlist*. Andriodzoom, 2010. URL http://www.androidzoom.com/android_applications/travel_and_local/checkmark-packlist_sbut.html.

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

Korte and Vygen, **2002**. Benhard Korte and Jens Vygen. *Combinatorial Optimization - Theory and Algorithms*. Springer, 2002.

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

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/.

railways. Indian railways. *luggage rules*. URL http://www.trainenquiry.com/StaticContent/Railway_Amnities/E_R/LUGGAGE.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.

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

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.

List of Corrections

Group Contract A

B130 - 3/2/12

Rules

§ 1 If anyone breaks rule 2-9 the rulebreaking person has to buy cake/sandwich/pizza or the like to the other group members.

§ 2 Keep appointments - also meeting time.

a You need to notice the other group members if you are unable to come or are late.

§ 3 A normal workday is from 9:10-16:15, unless there is a class in the morning - Then the meeting time is when the class starts.

§ 4 Everybody need to be ready to work at 9:10.

§ 5 Lunch is from 12.00-12.30.

§ 6 Be professionelly dedicated about work, the classes and the solving of group work.

§ 7 All appointments about deadlines on parts of the project must be kept.

§ 8 For all work related documents or other things LaTeX and SVN is used.

§ 9 If a groupmember does not participate in the groupwork, or are doing something unserious, the other groupmembers can shout "Chicken" at the person. If a person gets 3 "Chickens", they must be punished according to §1.

§ 10 There are high ambitions about the project.

§ 11 When the group is making important decisions a minimum of 5 group members have to be present

a All decisions will be put in the folder "Beslutninger" - Group members not present are notified by sms.

§ 12 2 group members work on the worksheet. If the work is not satisfying 2 new group members will be assigned the work.

§ 13 There will be a morning-meeting from 9:10-9:20 every work day, where we do not have classes. In that case the meeting will be right after the class.

§ 14 If the workload is to much for a group member the rest of the group need to be notified as fast as possible.

§ 15 The group coordinator has the responsibility that this problem is solved.

§ 16 The group coordinator has the responsibility that the time schedule is followed.

§ 17 It is a common responsibility in the group that all the group members are serious about the work.

§ 18 Friday meeting is every Friday morning unless the group have to go to class - in this case the meeting is Thursday morning.

a On this meeting the work of the day and and the weeks work will be discussed

b It will be checked if the time schedule is being follow

c Common thread (Where do we want to go? What happens now?)

d The worksheet are to be send before 12.00 Friday. The worksheet is made just after the meeting. The structure of the worksheet is discussed on the meeting. The worksheets is send to the contact person.

e If any group member knowingly, repeatly breakes the rules, CET is responsable that it is dicussed on a group meeting.

Also see "Nøglesamarbejdsaftalen" - (The contract about keys the the group room)

Group roles

Coordinator: Christian (If not present Mette)

Moderator: Dag (If not present Aleksander)

Reporter: Mette (If not present Dag)

Contactperson: Christian

Postman: Rasmus

Writers of the worksheet: Aleksander and Niels

CET - Conflict Emergency Team: Mette or Kasper or Christian (Only one person needs to solve the conflict but is she/he is in the conflict another member of CET has to solve the conflict)

Mindmap B

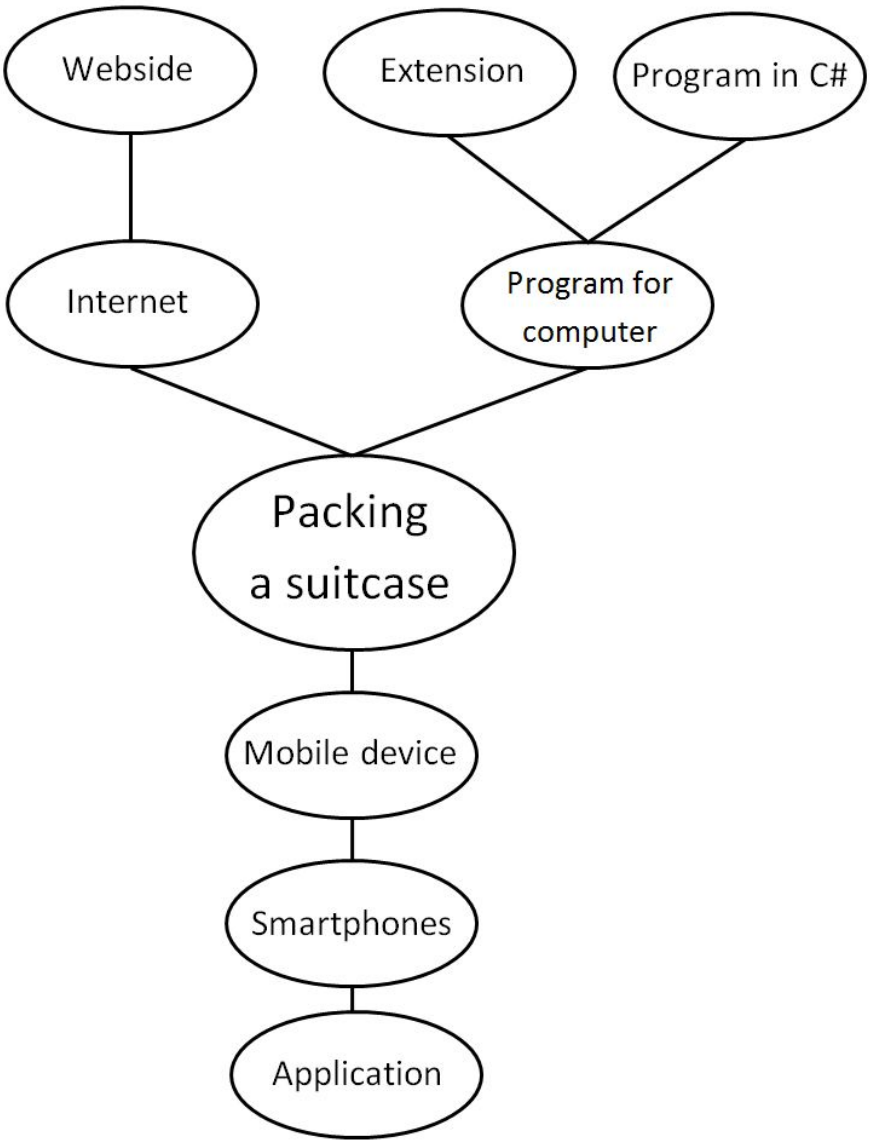

Figure B.1: Mindmap containing solution suggestions