NTNU
Norwegian University of
Science and Technology

Improving User Experience of Indoor Maps Through Merging of Rooms

Helmer Råby Schaug Njærheim
Morten Alver Normann

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Odd Erik Gundersen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Problem Description

Controlling the detail of the information visualized when zooming in maps is important to avoid cluttered maps. The task is to analyze possibilities and develop a method in close cooperation with MazeMap, which is a provider of indoor maps.

Abstract

The field of indoor map representation is an emerging field of research, contrary to outdoor maps, which has been under research for decades. A lot of focus in indoor maps has been on indoor positioning. However, map representation of indoor maps has not gained much attention. This thesis investigates the concept of level of detail in the context of indoor maps and how the right information can be displayed, potentially leading to an improved overall user experience.

There exist many generalization techniques for outdoor maps, but this has not been much discussed for indoor maps. These techniques and how they can be adapted to indoor environments are discussed in this thesis. An alternative indoor map solution has been implemented based on an existing one, where the new solution is a more dynamic representation of the map, based on zoom level. This solution has been tested through an experiment, where the new solution was compared with a more classical one. The results indicate that a more dynamic level of detail based on zoom level, can contribute to a more efficient navigation and improved user experience for indoors maps.

Sammendrag

Innendørs kart er et ungt felt som det er gjort relativt lite forskning på, sammenlignet med utendørs kart, som det har vært fokus på i tiår. Mye av den forskningen som er gjort på innendørs kart har angått innendørs posisjonering. Mens hvordan et innendørs kart representeres, har ikke fått noe særlig oppmerksomhet. Denne avhandlingen undersøker konseptet detaljnivå for innendørs kart, og hvordan den mest relevante informasjonen kan vises, som har et potensial til å forbedre den totale brukeropplevelsen.

Det finnes mange generaliseringsteknikker for utendørs kart, men dette har ikke blitt mye diskutert for innendørs kart. I denne avhandlingen har slike teknikker blitt diskutert med tanke på hvordan de kan tilpasses til innendørs kart. En alternativ innendørs kartløsning har blitt implementert, basert på en eksisterende løsning, hvor den nye løsningen har mer dynamiske detaljnivå, basert på zoom-nivå. Denne løsningen har blitt testet gjennom et eksperiment, hvor den ble sammenlignet med en mer klassisk innendørs løsning. Resultatene indikerer at mer dynamiske detaljnivå, kan bidra til mer effektiv navigasjon og forbedret brukeropplevelse for innendørs kart.

Acknowledgements

We would like to thank our supervisor Odd Erik Gundersen for all the frequent help and advice he has offered throughout our project work and master thesis, and for always being available for questions.

We also want to thank Iván Sánchez Ortega for his technical expertise.

Contents

List of Figures

List of Tables

Part I

Research background

Chapter 1

Introduction

In this chapter, the motivation for this project is presented. In addition, the research context and research approach are explained. This chapter was mostly written in a previous project where the main tasks were to find out which generalization that were used in indoor maps, and if it would be feasible to merge rooms together in an indoor map. This project is a continuation of the last project where, in addition, a solution will be implemented and tested. Therefore, most of the research background will contain the same information as in the previous project.

1.1 Background and Motivation

Justin O'Beirne wrote an article about the level of detail in Google Maps[1], where he compares maps from 2010 with maps from 2016 (O'Beirne, 2016). One comparison can be seen in Figure 1.1.

In the left image, there is a lot of cities and not many roads. He argues that this results in some cities seeming like they are unreachable by driving. In the right image, it is the exact opposite. There are more roads but far fewer cities. In this case, it looks like many roads are going nowhere.

His conclusion is that there is an undesirable imbalance between roads and cities. Even though it is not discussed in the same way, Pinedo et al. (2013) are discussing the problem of too much complexity in an outdoor map. Their goal is to achieve a map that removes unnecessary information, and most importantly, perform user

[1]https://maps.google.com

tests for evaluation. Unfortunately, they haven't run any tests yet to get empirical results for their approach.

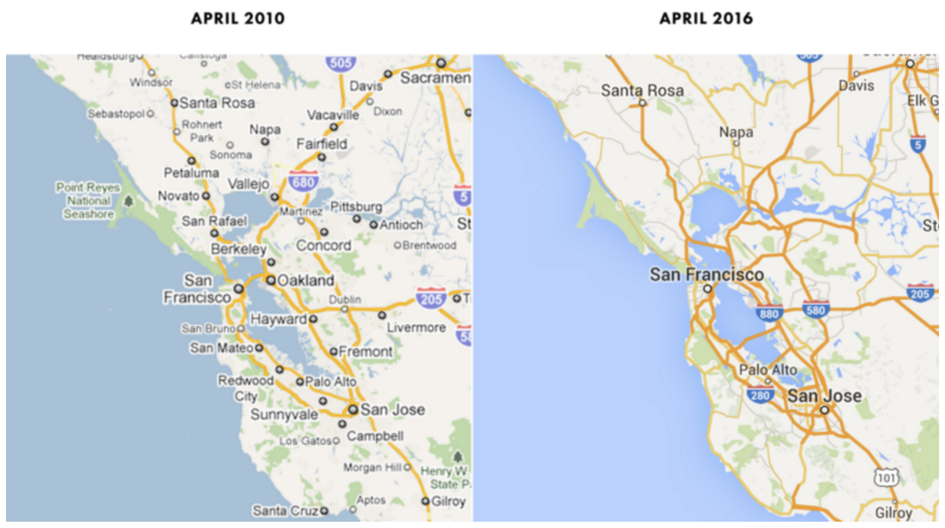


Figure 1.1: Comparison between Google maps 2010 and 2016 of the San Francisco Bay Area (O'Beirne, 2016)

The issues with the display of information are based on opinions, and actual evidence is not presented. Even though the issues mentioned are presented in the context of outdoor maps, they give the same motivations for indoor maps, since indoor and outdoor maps share a lot of the same concepts. Roads can be thought of as rooms, while landmarks like cities, street names, etc. can be thought of as room names, restroom icons, stairs icons, etc. Landmarks will be more generally referred to as points of interest (POIs).

In a case study conducted by Puikkonen et al. (2009), where the test subjects had to navigate in a shopping mall, some of their recommendations based on their results was to show fewer details on the map and make landmarks more apparent. This solution proposal sounds consistent with the process of evening out the imbalance from the 2016 version of Google Maps, discussed by O'Beirne.

1.2 Hypothesis and Research Questions

Based on the motivations presented so far, the following hypothesis is defined:

HYP: A more dynamic level of detail (LOD) based on zoom level, will contribute to a more efficient navigation and improved user experience for indoor maps.

This means that an indoor map solution that contains a dynamic level of detail has to be implemented. The concept of level of detail is explained in detail in Section 2.1. The most important points are creating some form of generalization of the map depending on the zoom level, where the most relevant details are emphasized. From our previous project we answered the following research questions:

- RQ0.1: What indoor map solutions exist already?
- RQ0.2: What methods exist for indoor map generalization?
- RQ0.2.1: What kinds of more general methods can be applicable for indoor map generalization?

After performing a structured literature search, we could with great confidence state that there didn't exist any solutions or research that considered an approach like ours, answering RQ0.1. This literature search is described in Section 3.1. An extensive literature search gave us a very strong hint that indoor map generalization was not an active research field, but we found a lot of more general algorithms, that could be applied to indoor map generalization, and verified that it was a feasible task, answering RQ0.2 and RQ0.2.1. We believed that merging rooms together into larger rooms on lower zoom levels, and merging hallways together so that they look like indoor roads, would be a good way to create a more dynamic level of detail, according to the motivation presented.

Based on the answers to the previous research questions, a new set of research questions could be defined to reach the goal of verifying or rejecting the underlying hypothesis:

RQ1: Will merging of rooms be helpful for the user, when navigating in an indoor map?

We had to determine if merging rooms actually had an impact on the user experience. If merging rooms would turn out to be improving the user experience,

then the merging itself could be worth the effort.
Since merging of rooms would change the geometry of the rooms, we believe showing room names in a corresponding way so that it fits the underlying geometry would be more appropriate. Hence, the next question is defined:

RQ2: Will merging of room names (POIs) be helpful for the user, when navigating an indoor map?

Rooms have their own name displayed over them. When merging a room it would be natural to merge the room names as well. However, that doesn't mean if one of them is helpful, the other will be too. Hence, we believe it was necessary to ask this question as well.

Based on the analogy about cities and roads in outdoor maps, compared to rooms and hallways in indoor maps, we believe it would be important to also test the effect of merged hallways, and see if it is beneficial for the user experience. This creates the next and final research question:

RQ3: Will merging of hallways be helpful for the user, when navigating in an indoor map?

1.3 Research Context

Even though the main motivation has arisen from Google Maps, this thesis will present an indoor map solution modified from the existing application called MazeMap[2], an established indoor map solution. Recall the mapping between outdoor and indoor maps mentioned earlier. Figure 1.2 displays a building in MazeMap. One can see that the imbalance in this image is the same kind as in the right image in Figure 1.1.
A lot of rooms are displayed, but a significant amount of them has no room name or another POI and is therefore not very helpful for a user trying to navigate. Creating an improved balance between rooms and POIs could be beneficial for the user experience.

[2]`https://use.mazemap.com`

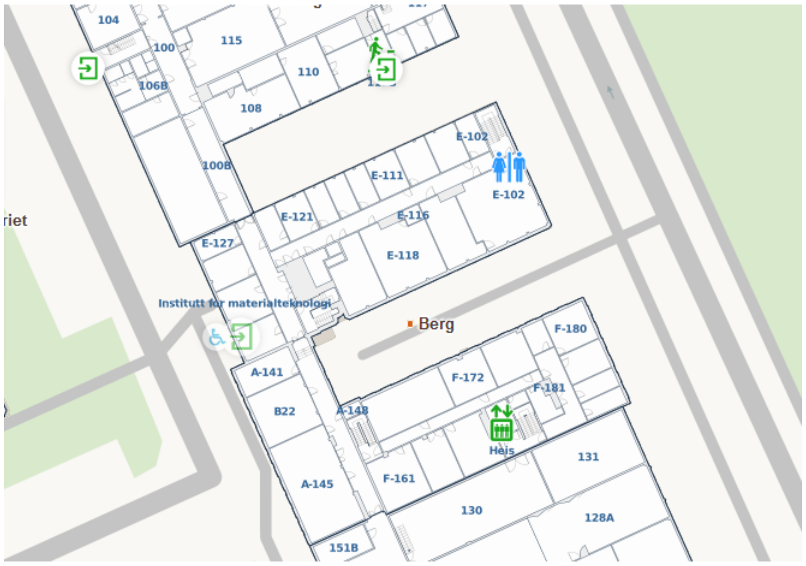


Figure 1.2: Screenshot of a building (”Berg”) in MazeMap, NTNU Gløshaugen, Trondheim

1.4 Research Approach

The main goal of this thesis is to implement a new solution to even out this imbalance. Then this solution can be compared to the original one, and empirical data can be generated.

First of all, a structured literature search was performed to get an overview of existing research. This will be further explained in Chapter 3. Shea (1988) describes objectives for generalization of cartographic maps. One type of objectives presented is philosophical objective which is the most fitting for the approach used in this thesis. From all the sub-objectives presented under philosophical objectives, Shea (1988), page 17, it has been decided that the following three are most relevant to the approach used in this thesis, where the focus is on user experience:

- Reducing Complexity: Remove information that does not have any significant effect on the user experience.
- Maintaining Spatial Accuracy: Keep the generalization as similar as possible to the original, to prevent user confusion.

- Maintaining a Logical Hierarchy: Give a clearer ordering of objects. Different objects are of different importance to the user.

In the next subsection, different possible improvements will be considered with these three points in mind.

1.4.1 Level of Detail

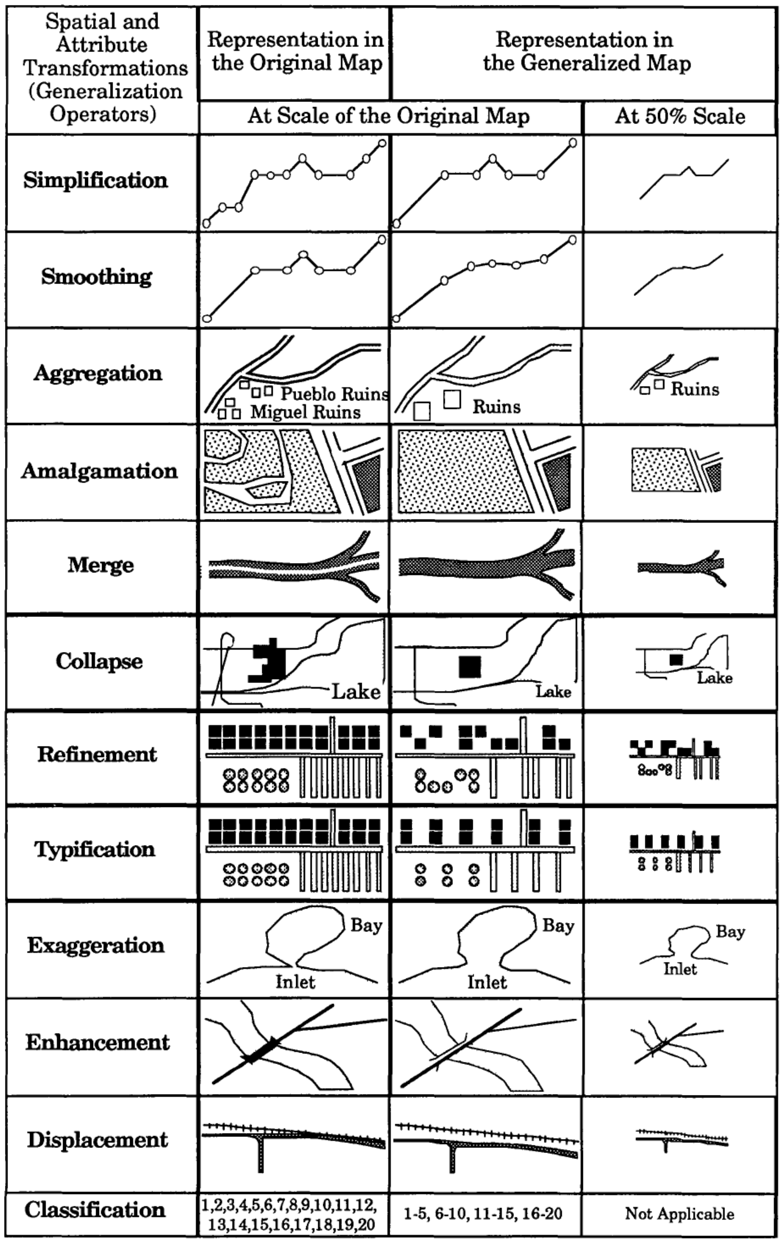
Ideally, a user only wants to see the information he/she needs, to achieve the best navigation. The current MazeMap implementation doesn't have a very dynamic level of detail when it comes to rooms. The POIs are dynamic, but this could also be improved. Both of these will be discussed in the following two paragraphs.

Rooms
Currently, in MazeMap, all rooms are always shown below the zoom level where the user sees the inside of buildings.

Decreasing the number of rooms displayed might be a good idea. Rooms have different levels of importance. Rooms like lecture halls and meeting rooms are much more popular destinations than closets. Simply removing rooms of lower importance, when zoomed out, could be a potential solution. However, an undesirable effect of just removing rooms gives the impression of empty spaces in the building, even though there are rooms there, which may confuse the user. This effect goes against the principle of maintaining the spatial accuracy, mentioned in the previous section.

Shea and McMaster (1989) give an overview of different ways of cartographic generalization and also provide a figure which summarizes them all. This is shown in Figure 1.3. It would be desirable to simplify rooms while still keeping their orientation and shape. From Figure 1.3, refinement and typification show the undesirable effect of the empty space impressions mentioned earlier. Merging, however, keeps the overall shape and orientation intact which is the objectives of this solution. Small rooms can be merged into bigger ones. The user would see one large room instead of several small rooms. The complexity would be reduced and the spatial accuracy would be maintained.

Additionally, a dynamic merging of rooms could give a better logic hierarchy. Rooms of lower importance, like storage rooms, could be merged together, while more important rooms, like lecture halls, would not be affected.



Spatial and Attribute Transformations (Generalization Operators)	Representation in the Original Map	Representation in the Generalized Map	
	At Scale of the Original Map		At 50% Scale
Simplification			
Smoothing			
Aggregation	Pueblo Ruins Miguel Ruins	Ruins	Ruins
Amalgamation			
Merge			
Collapse	Lake	Lake	Lake
Refinement			
Typification			
Exaggeration	Bay Inlet	Bay Inlet	Bay Inlet
Enhancement			
Displacement			
Classification	1,2,3,4,5,6,7,8,9,10,11,12, 13,14,15,16,17,18,19,20	1-5, 6-10, 11-15, 16-20	Not Applicable

Figure 1.3: Table of cartographic generalization techniques (Shea and McMaster, 1989, p. 9)

POIs
In MazeMap, at the time of writing of this thesis, POIs that don't include room names seem to follow a reasonable pattern in terms of level of detail. But room names have a significant potential for improvement.
A little above the center in Figure 1.2 there is a very small room, not even accessible from the hallway, which is marked by its name ”E-116”, while there are other bigger rooms connected to a hallway, not marked by their name. This seems counter-intuitive since those big rooms are most likely more popular than this small room. Hence, showing room names of more popular rooms, before showing room names of less popular rooms as the user is zooming in, is one possible improvement that can contribute to better hierarchy logic.

1.5 Thesis structure

This thesis is divided into three parts. Part I presents the research background, which contains an introduction to the thesis, background information, and state of the art of the central research topics of this thesis. Part II presents the methodology, which includes what the implementation should do, how it is made and a description of how an experiment was performed to test it. Part III contains the result and discussion as well as evaluation, conclusion and future work.

Chapter 2

Background

This chapter will briefly explain different background topics that are relevant to this project. The goal is to give the reader a better overview of the concepts before reading the rest of this thesis.

2.1 Level of Detail

Level of detail (LOD) is the concept of decreasing detail of a graphical model the further you get away from it. For example, if there is a model of a car far away from the viewer, smaller details like the antenna and the door handles, won't be of any concern to the viewer. If the viewer wants to inspect the parts mentioned, he/she has to get closer to get a good look. In computer graphics, a different number of polygons are used to render an object based on its distance from the viewer, since the viewer would not be able to make out many details anyway when it is so far away. The improvement in rendering performance is vast, by using this concept. Figure 2.1 shows an example of this concept. The most detailed level contains over 16 000 triangles while the simplest model consists only of 66 triangles.

LODs are typically divided into two groups: Discrete LODs (DLOD) and continuous LODs (CLOD). DLODs are most widely used in applications today (Ribelles et al., 2010).

DLODs are usually using models from a relatively small pool of models. The models in Figure 2.1 are a typical example of a DLOD, where there are 5 models and each model is significantly different in details. It will be easily noticeable when one model changes to another in the pool. One clear advantage is that there are not many models to keep track of. However, a disadvantage is that the transformation from one model to a less or more detailed one results in noticeable changes of the model in an instant, called ”the popping effect” (Ribelles et al., 2010). This can be undesirable for applications where the goal is to display a realistic environment.

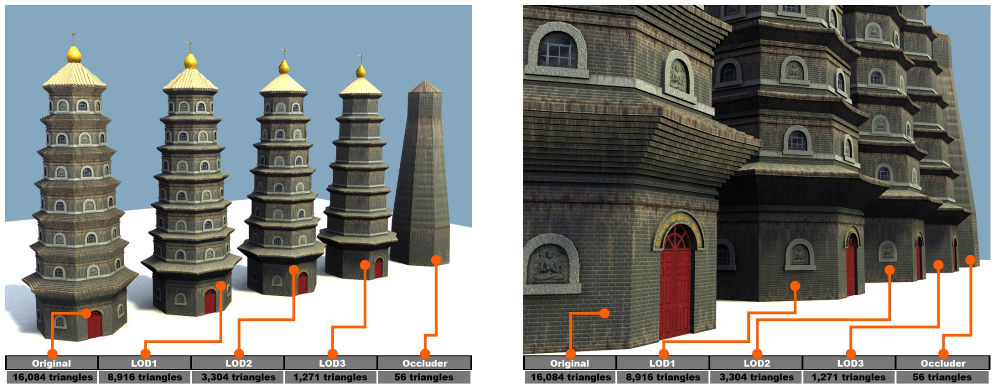


Figure 2.1: Different levels of detail of a 3D model of a tower (Biswas, 2009)

CLODs are using a much larger pool of models. The transformations from one model to another will be less noticeable and lower the popping effect. An obvious disadvantage is the large amount of models to create.

2.1.1 Viewing and Zooming

There are different techniques for viewing graphical interfaces and zooming in them. Cockburn et al. (2009) give an overview of different techniques that can make it easier for users to navigate and focus on a digital map.

Overview and detail can be combined. In the case of a map, the detailed, smaller part of the map can be displayed on the majority of the screen, while a small view in the corner can contain an overview of the entire map, making orientation over the whole map easier for the user.

Another approach is using a fish-eye view. One small region of the map is magnified, while the area around this region is skewed out of this area, allowing for full focus on this region.

The transition between zoom states can be discrete or continuous. Discrete transition means immediately changing the focal region and the appropriate level of detail. In continuous zooming, the graphical features gradually change toward the corresponding focal region. The level of detail can be changed after this smooth transition, helping the user to keep track of what corresponds to which.

2.2 Rendering

Rendering is the process of generating a final digital product from a specific type of input (Christensson, 2016), which in our case it how the map is generated. This section briefly covers different rendering techniques.

Tiled Rendering

The most intuitive rendering approach, especially for standard websites, is to update the entire screen as soon as all the DOM elements are ready to be displayed. This is called immediate mode rendering. There is another approach, called tiled rendering, where the map is split into smaller tiles, for instance, 256x256 pixel tiles. When each tile is ready to be rendered, it is displayed on the screen, independently of other tiles. Hence, the entire screen is gradually rendered, tile by tile, instead of the entire screen suddenly updating. The advantage is that the user gets feedback that something is happening instead of the screen freezing and suddenly updating.

SVG and Canvas

When SVG (Scalable Vector Graphics) is rendered on the screen it is kept in memory. That is, if a small change to the vector data is changed, it is not necessary to repaint the whole screen, but a smaller part can be updated instead.
Canvas works differently since when canvas elements are displayed on the screen, they are immediately forgotten. So every change requires a full repaint of the screen. However, nothing is stored in memory, so it is more memory efficient.
If the application requires a lot of change in graphics from user movements like dragging and zooming, canvas has an advantage since it requires less to update the entire screen than SVG does. SVG is preferable when smaller changes happen now and then, say like a graphics manipulator program.

2.3 Indoor vs Outdoor Maps

There are many differences between indoor maps and outdoor maps. For instance, outside objects are often scattered, while inside, the rooms are normally side by side. Inside objects are usually simpler and more predictable. In outdoor maps, there are a lot of natural features like lakes, rivers, height levels, woods etc. Indoor maps only have strictly designed rooms. Buildings and rooms often have sharp 90 degree corners and simple shapes like rectangles. However, indoor maps have a different complexity in the form of different floors.

An additional important difference is that outdoors is public space. Everyone has access to it. However, not everyone wants to share their floor plans to the public, even though it may be a public building. This is probably an important factor to why there are so many solutions for outdoor maps compared to indoor maps.

A map of a building with rooms can be compared to a map of a country with regions. But the rooms differ from the regions because the rooms do not share edges since the walls have a thickness. Therefore, it can be additional challenges when merging rooms, compared with merging of regions.

2.4 Polygon Simplification

A natural question regarding LOD is how to create the pool of simplified models in the first place. According to Heckbert and Garland (1997), there is no general algorithm that works well for all contexts. There are many polygon reduction algorithms, dependent on polygon model structures and shapes. Heckbert and Garland (1997) covers various polygon reduction techniques and gives a good overview of how to solve different problems, based on contexts. A commonly used algorithm that can be relevant to map cartography will now be explained in more detail.

2.4.1 Douglas-Peucker Algorithm

According to Wu et al. (2004), the Douglas-Peucker line-simplification algorithm is recognized as the one that delivers the best perceptual representations of the original lines. The Douglas-Peucker algorithm simplifies a polyline by first, making a line between the first point, called an anchor, and the last point, called a floater. Then, the intervening points along the original line are examined to find the one with the greatest perpendicular distance to the straight line defined by the anchor and the floater. If this distance is less than a tolerance distance, the straight line is accepted as a simplification of the polyline. If the distance is greater than the tolerance, the problem is divided into two sub-problems. The first sub-problem will still have the first point as the anchor, but the floater will be the point that had the greatest perpendicular distance to the previous straight line. The second sub-problems will use this point as the anchor, and the last point as the floater. This process is repeated until all points are within the tolerance distance to one of the simplified lines (Douglas and Peucker, 1973).

Douglas-Peucker gives a unique simplification of a polyline since it keeps the endpoints of the line. However, it is not unique for polygons, since the starting point can be any point. Even a point that should have been removed, if another starting point had been used.

2.5 About MazeMap

Before going into the details of this thesis, it would be beneficial to give a brief introduction of MazeMap and their current map implementation.

MazeMap provides a collection of interactive indoor maps for the public to use. This collection typically consists of large public buildings, like universities, hospitals, airports, etc. MazeMap's solution is built on top of the Leaflet JavaScript library, which is targeting interactive outdoor maps. The maps provide details about the floor plans and POIs, which are icons that mark room names, stairs, elevators, restrooms, and much more. The main motivation is to make it much easier for a user to navigate in a building. Currently, most maps are in Norway, but some buildings in foreign countries have also been added.

MazeMap have made available part of their Data Application Programming Interfaces (APIs) for developers to use[1]. In this project, these Data APIs are used to get JavaScript Object Notation (JSON) objects for floors and POIs. The JSON objects contain information like coordinates and room types.

[1]`https://github.com/MazeMap/Data-API` (Last checked 14/11/16)

2.6 Related Solutions

MazeMap's solution is not the only solution for indoor maps. Even though the focus of this project is on MazeMap's solution, a few other solutions will be addressed, to compare and find pros and cons.

2.6.1 Google Maps

Google Maps[2] also contains indoor maps for specific buildings in the world. A sample screenshot is shown in Figure 2.2.

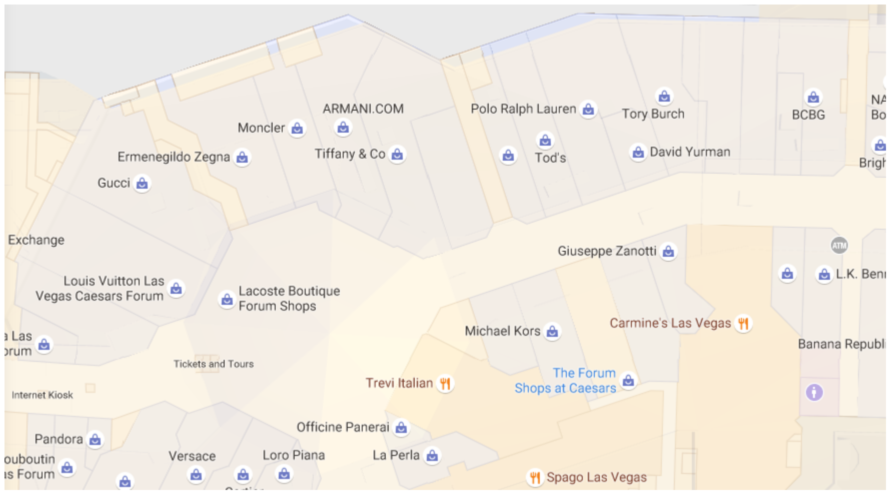


Figure 2.2: Small section in Caesars Palace, Las Vegas shown in Google Maps

Although the design looks different from MazeMap's solution, its level of detail is not that different. All rooms and their details are shown at any level, until the user zooms out far enough, where nothing is shown. This leads to a lot of tiny details scattered around, on a low zoom level. The balance between POIs and rooms looks good at this level, but it does suffer from the same level of detail issue as MazeMap's solution does.

Google Maps include more use of colors in their design. This gives a clearer

[2] `https://www.google.com/maps/about/partners/indoormaps/` (Last checked: 05/12/16)

contrast between specific types of rooms. This makes it easier to, for instance, see the hallways.

2.6.2 Mapwize

Mapwize[3] is a solution for indoor maps, where users can upload their own maps with defined POIs. Figure 2.3 shows a sample from Mapwize.

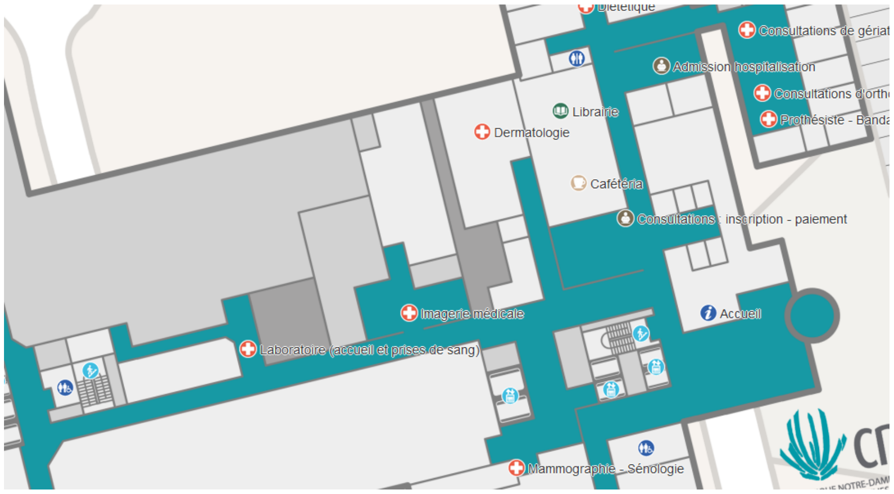


Figure 2.3: Section of a hospital in Paris in Mapwize

It is clear that they have thought of how important hallways are. All hallways are filled blue which we would deem as helpful for navigation. One con is that doors are not drawn. In Figure 2.3 there are some large rooms, and figuring out where to enter them seem like pure guessing.

When it comes to the most significant part, level of detail, Mapwize have the same approach as MazeMap and Google Maps. All rooms are always shown, which results in a lot of seemingly empty rooms. Also, some of the POIs seem a little off. For example, the POI saying mammography (Mammographie) in the central bottom is on the intersection of three rooms, giving no clear indication to which room it is.

[3]`https://www.mapwize.io/en/` (Last checked: 05/12/16)

2.6.3 Micello

Micello[4] has a little different design than the three others discussed so far. The user finds the building he/she wants to navigate, then an isolated map for that building is loaded and displayed in another window. A sample can be seen in Figure 2.4. Again, all rooms are always shown, leaving a lot of blank rooms for a lower zoom level. Some POIs don't adapt to the current zoom level, so, for example, a POI for a bathroom is very small when zoomed far out, which can make it more difficult for the user.

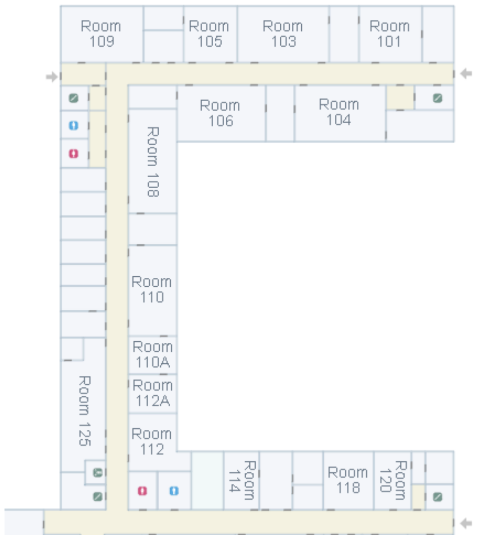


Figure 2.4: Art Building SJSU, San Jose, California in Micello

However, like Mapwize and Google Maps they are using a different color for hallways and doors are also included.

2.6.4 Summary of related solutions

It is safe to state that these solutions don't have any dynamic level of detail when it comes to rooms. To our knowledge, there is no such approach in any indoor map solution.

[4]`https://www.micello.com/` (Last checked: 05/12/16)

Having a more dynamic level of detail will hopefully make the user focus on the right things appropriate for each zoom level and avoid getting distracted by unnecessary details. This should ultimately lead to a better user experience for indoor map navigation.

There are different approaches to using colors and contrasts in current solutions, which is probably worth experimenting with in MazeMap to further ease the navigation for the user.

2.7 Summary

In this chapter, different relevant background topics have been presented, so that the reader have a basic understanding of concepts that are important for this thesis. Additionally, related solutions have been presented to show the reader the state of the art of indoor maps solutions.

Chapter 3

Relevant Research

This chapter discusses research that has been performed on relative fields to the problem discussed in this thesis. It first describes how a structured literature search was performed, then it discusses other relevant research the can be used in this solution.

3.1 Method for Structured Literature Search

In an earlier project, a research was performed to get an overview of what had already been done. To get this overview, Google Scholar[1] was used to search for relevant search terms. Some of the terms are displayed in Table 3.1.

Search term	**# of results**
indoor map	429000
indoor map ”points of interest” ”level of detail”	2620
indoor map zoom cartographic generalization ”points of interest” ”level of detail”	180
”indoor” map zoom cartographic generalization ”points of interest” ”level of detail”	78

Table 3.1: Search terms and the corresponding number of results

The difference between the to last search terms in Table 3.1, is that quotation marks

[1]`https://scholar.google.no/`

are added around the word *indoor* in the last search term. This makes sure Google Scholar only gives results that contain the word *indoor*, and not other similar words like *inside* and *interior*. Of those 78 result, only six discussed indoor maps. The relevance of these six papers for this project will now be discussed briefly.

How to design a pedestrian navigation system for indoor and outdoor environments (Radoczky, 2007)
In this paper, the focus is on navigation in indoor and outdoor environments, but it mentions a few relevant topics, like POIs and floor plans.

Adaptive Mobile Visualization – The Chameleon Framework (Pombinho et al., 2015)
The objective of this paper is to present the design of a framework for adaptive mobile visualization (AMV) applications. In the last chapter, an AMV application that uses the user's indoor location context is discussed. The focus is on the user's movement and positioning and is therefore not very relevant to this project.

Literature Survey BA8204 (Nossum)
This is a literature survey, where the task was to investigate the literature found on four different themes, namely: Indoor positioning, Indoor cartography, Novel evaluation methods in cartography and Novelties in cartography. In the literature survey, Nossum discusses papers on the different topics in the different chapters. The chapter containing indoor cartography is relevant for this project, but as Nossum mentions, several of the articles discussed focus on positioning technologies and other non-cartographic issues.

Exploring new visualization methods for multi-storey indoor environments and dynamic spatial phenomena (Nossum, 2013)
In this thesis, Nossum discusses visualization in general, and the problems of visualizing a multi-storey building. Some of the points made about visualization in maps can also be relevant when deciding how to display a simplified indoor map.

Indoor Maps (Sandström and Svensson, 2010)
The main goal of this master thesis was to create a useful format for storing indoor maps and to make an editor, which can quickly and easily create maps. Which means that the focus was not on simplifying indoor maps, but to make it easier for the users to create new maps.

Context dependent multimodal routing in indoor/outdoor environments based on IndoorGML and OpenStreetMap (Mouratidis, 2015) The goal of this master thesis is to reach a point of seamless navigation from outdoor to indoor space. Therefore, it is partially about indoor maps, but adjustments to indoor maps are not discussed.

Summary
Even though some of the papers reviewed contain relevant topics, none of them focus on generalizing indoor maps, which is the main topic of this project. Since there have not been done much research on indoor maps, the research in this thesis is mainly inspired by polygon simplification and regular map generalization.

3.2 User Satisfaction Based on Level of Information

This may sound like common sense, but people are different and have various preferences. There are little empirical results regarding the qualitative part of the level of information shown. Most research is concerned with quantitative results such as performance and efficiency (Brooks and Tobias, 1996) (Anders, 2005) (Lindstrom et al., 1996).

Cockburn et al. (2009) give results for empirical studies of different viewing techniques of graphical data. However, it is not concerned with the representation of the graphics themselves. The concept of visual analytics is discussed by Keim et al. (2008) and Thomas and Cook (2006), which admits the challenges of the growing number of data and how to visualize it in the most useful way, for better human decision-making. Keim et al. (2008) define ”The Information Overload Problem” in a general manner as the danger of getting lost in data, that leads to a waste of time and money, while Thomas and Cook (2006) talk more about the application of detecting unpredictable events like natural disasters and acts of terrorism, by better visual analysis of data. Visual analytics is relevant to this problem since the aim is to display data in the most user-friendly way.

As mentioned in Section 1.1, Pinedo et al. (2013) discuss generalizing in outdoor maps, specifically with focus on LOD. But they don’t have any empirical results, which doesn’t give any evidence regarding their reasoning.

Lorenz et al. (2013) present general empirical results for navigation in 2D and 3D indoor maps. They give useful information about the importance of map perspective and landmarks like location names, stairs, doors, etc. These results can be important for this thesis since they can give good design ideas for POIs. Users taking their survey seems generally unsatisfied with their 2D version, which is good evidence there is room for improvement.

3.3 Map Generalization

There has been a significant amount of research done on map generalization. Still, it is a challenging problem, because of the variety of maps. It should be noted that the majority of research is targeting outdoor maps. One challenging problem is the automation of generalization. This was the main subject discussed by Sester et al. (2009) where different approaches of automated spatial generalization were presented. Map partitioning is discussed by Van Oosterom (1995), where a tree data structure is used to divide a map into different groups such as forests, buildings, parks, etc.

Haunert and Wolff (2008) present an algorithm for generalization building outlines which is very close to the main goal. They use relationships between edges of a polygon to find ”shortcuts” to simplify polygons. Buildings were also the motivation for Regnauld (2001), but his focus was to generalize groups of buildings and not individual buildings.

3.4 Visual Attention

It is relevant to this subject to discuss research on visual attention. The focus of humans is affected by the environment surrounding them. Specific features tend to attract a human's attention more than other features, such as other people, signs, landmarks compared to pavements, grass, etc. Research done in this field can give ideas for design contributing to users focusing on the most useful features for each level of detail. This section will go through different research and discuss their results.

Bertin (1983) lists seven visual variables and their perceptual properties that guide the visual attention of humans: x-position, y-position, size, color value, color hue, shape, and orientation.

Experiments regarding visual search, conducted by Treisman and Gelade (1980) give strong indications that color and orientation are important features that attract attention.

A much more recent paper goes through a lot of previous research and weights visual variables based on results and conclusion from this research (Wolfe and Horowitz, 2004). Their conclusion is that color, motion, orientation, and size is generally the most important attributes for guiding visual attention. Note that color is only counted as one variable, contrary to color value and color hue listed by Bertin (1983). Additionally, motion is not in this original list either.

The results of these papers give solid evidence that color, orientation, and size are important guiding variables.

Garlandini and Fabrikant (2009) give a more thorough analysis of color value, color hue, size, and orientation, by giving participants different tasks for a changing 2D map. The following three tasks were given:

- Detection task: To detect if the map had changed in any way
- Localization task: To see where on the map a change had happened
- Identification task: To say what kind of change happened

Their results show that changing size gave the best score for all three. Both changes in color hue and value gave a little lower score, but still very good. Changes in orientation gave the lowest score. Their results are shown in Figure 3.1.

Osberger and Rohaly (2001) give a very clear idea of how people's attention is affected, by giving results from participants looking at still images and videos. They also point to contrast as an important visual variable and its experiment results give evidence that this is indeed an important visual variable. Different colors result in more contrast, which strengthens the improvement potential of colors.

Again, looking at Figure 1.2 there is close to no use of colors. All the rooms are plain white, which practically eliminates the color variable as a visual variable. The majority of research so far points out color as one of the most important variables for attracting user attention, so a more dynamic use of color could improve the user experience.

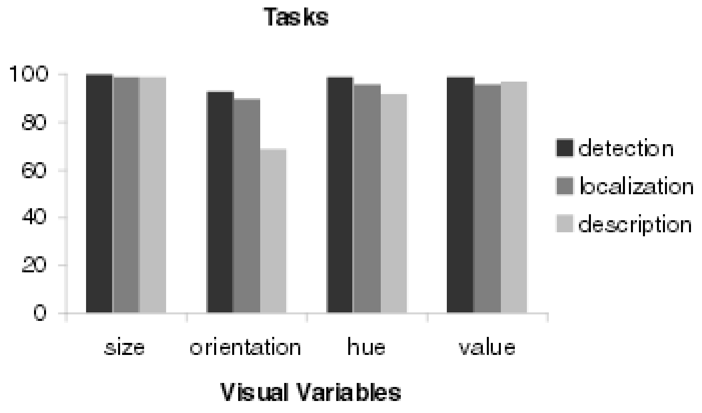


Figure 3.1: Results for the performance of changing visual variables (Garlandini and Fabrikant, 2009, p. 10)

3.5 Level of Detail

Reddy (1998) provides a good summary of common factors that can play a role in deciding which representation to choose for a model. The decision is based on the four following factors:

- Distance: The distance from the object to the viewer
- Size: The number of pixels that is covered by the object
- Eccentricity: The degree of centrality of the object, relative to the display and the viewer's field of view
- Velocity: The velocity of the object, relative to the viewer

There exist more factors, such as giving objects in the environment different priorities. Holloway (1992) makes sure objects with high priority are not affected by polygon reduction, while objects with low priority are. Which factors to use is entirely based on the context of the application.

3.6 Indoor Maps

As mentioned in Section 2.3, indoor maps usually have different floors. This creates multiple problems. For instance, how to display multiple floors in a 2D map. The solution MazeMap is using, and the most common one, is to give the user the possibility to decide which floor that should be displayed by clicking different buttons. Another solution, discussed by Nossum (2011), is to display multiple floors in one map. Another problem for maps with navigation is that Global Positioning System (GPS) often performs too poorly inside buildings, as discussed by Kjærgaard et al. (2010). This might also contribution to why outside maps are more common than indoor maps.

3.7 Polygon Merge

There exist various polygon merge algorithms out there. Žalik (2003) proposes an algorithm for polygon merging, by removing all shared edges between polygons. Polygon amalgamation is described by Zhou et al. (1999) which is to create an overall boundary around a general input of polygons, using a similar idea. Both of these algorithms require polygons that share edges, for example, country borders.

Wigren (2011), summarizes many merging algorithms. One of them is merging disjoint polygons, which means it can also handle cases where polygons don't share edges. It is a straight-forward procedure that draws lines between points in both polygons that are closest and second closest to each other, called twin points, as long as the lines don't cross each other. Then it removes the other edges connected between these points, resulting in a merged polygon.
Another possible approach to merge polygons is the adopt merge operator discussed by Ware et al. (1995). This method uses Delaunay triangulation, which creates triangles between the points in the polygons, as shown in Figure 3.2(a). Then, the triangles should be included in the merged polygon, o3, if its longest edge is less than a decided threshold. This is the four triangles called Ssub in Figure 3.2(b). The result of the merge can be seen in Figure 3.2(d).

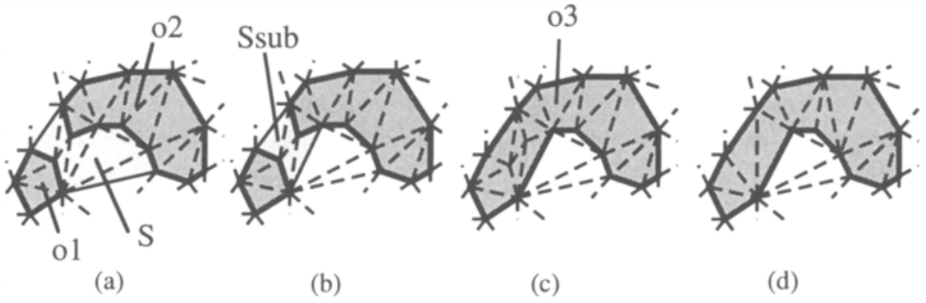


Figure 3.2: Using the adopt merge operator based on Delaunay triangulation to merge two objects, o1, and o2, into one object, o3 (Ware et al., 1995).

3.8 Summary

This chapter has discussed research that can be relevant for our approach and made the reader aware of the current state of the research in these fields. This research can be useful for some implementation decisions.

Part II

Methodology

Chapter 4

Method

This chapter describes the overall implementation process and the decisions that were made. The overall implementation is presented first to give the reader a rough understanding of the pipeline from original rooms to merged rooms. Then each step will be explained in more detail.

4.1 Implementation Overview

The top priority for this thesis was to make a more user-friendly display of indoor maps. As mentioned earlier, the level of detail in current implementations is not very dynamic and results in a lot of information being displayed, independent of zoom level.

As discussed in the introduction, the plan was to merge rooms together. For a further out zoom level merged rooms would be shown and as the user is zooming in, it would gradually be split into smaller merged rooms and finally into the individual rooms. When the merged polygons had been created, they could be stored and that data could be used in the new implementation. Hence, the entire processing pipeline could be a pre-processing stage, and time complexity would not be essential. The following sections describe the implementations in more detail.

4.2 Hallways

The data collected from MazeMap's data API includes the room type of the different rooms. Therefore, it is easy to separate the different kinds. This gives us the opportunity to separate rooms and hallways.

In MazeMap, all rooms are white. It keeps the design simplistic and avoids making unnecessary distractions, in addition to the POIs. However, one possible change is discussed here where a clearer contrast can be helpful for better navigation. As already discussed, hallways are thought of as roads. Hence, hallways are very rarely a destination, but they are very often on the path to the destination. Because of this, spotting a hallway quickly will probably be of great use to the user. In many buildings, it is very easy to spot hallways as the long, thin and central rooms. Figure 1.2 is an example where this is the case. One long hallway is going up and down on the left side of the building and two smaller hallways connected to its right side.

Unfortunately, not all buildings have an equally obvious hallway. Figure 4.1 shows a building with a lot of rooms cluttered together. If a user's destination is one of the rooms in the middle, it is not an obvious task to say which way to go to get to the destination. Recall that Google Maps, Mapwize and Micello also had different colors for their hallways. Therefore, we thought that having a different color than white on the hallways could attract the user's attention to them. This could make it easier for the user to notice how hallways are merged and contribute to answering RQ3.

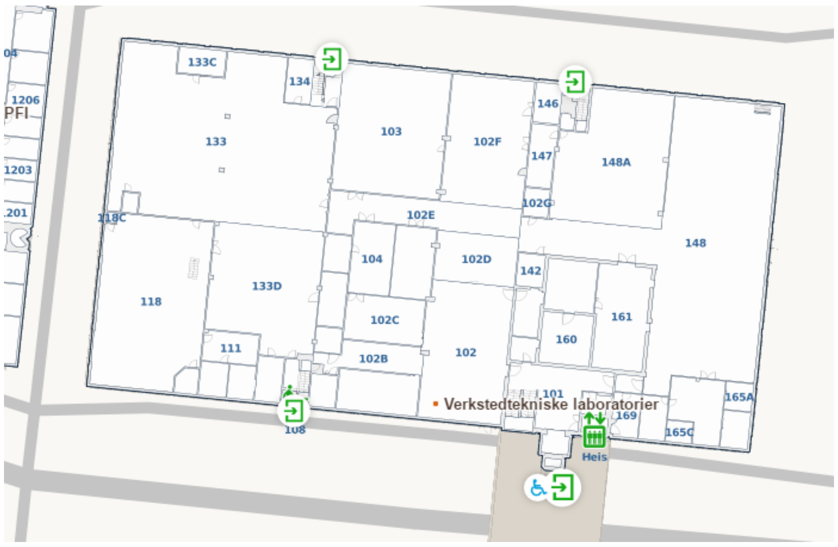


Figure 4.1: Screenshot of a building ("Verkstedtekniske laboratorier") in MazeMap, NTNU Gløshaugen, Trondheim

4.3 Neighbor rooms

Before merging rooms, it had to be decided which rooms should be merged. Initially, only rooms of the same type were considered neighbors, which means that, for instance, offices would only be merged with other offices. But since the room type is usually not available in other APIs than the one used from MazeMap, it was decided that all rooms could be neighbors as long as not one of them is a hallway while the other is not. This makes the implementation more general. Even though other indoor maps may not contain the information about if a room is a hallway or not, hallways may usually be possible to classify quite well through machine learning. This is because hallways normally differ more from other rooms than for instance offices or meeting rooms do.

When finding neighbor rooms, five simplified scenarios with two close rooms were considered. Figure 4.2 shows four scenarios where the rooms are added as neighbors. The fifth scenario that is considered is shown to the right in Figure 4.3. It was decided that these rooms should not be added as neighbors since it would probably not improve the user experience, and it could make merged polygon overlap.

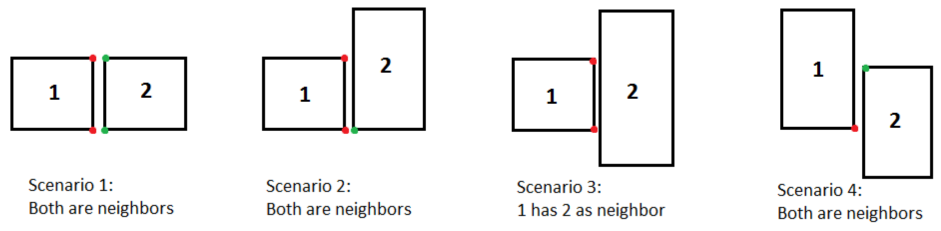


Figure 4.2: Scenarios where rooms will be added as neighbors

4.4 Room Merge

We decided to implement a merging algorithm, similar to the disjoint polygon algorithm presented in Wigren (2011), with some modifications that would adapt it to this application. The adapted algorithm also finds not only the two closest points in both polygons but all points closer than a certain threshold, to be able to deal

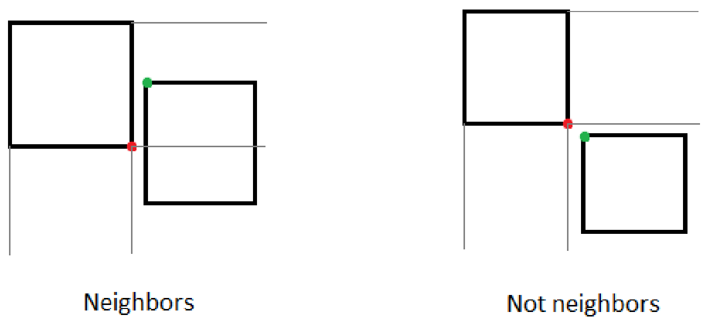


Figure 4.3: Distinguish between if any part of the walls are adjacent or not.

with complex polygons. The reason we chose this approach is that it will be flexible when it comes to additional modifications. We also considered the merge adopt operator discussed by Ware et al. (1995), but there was not much information about how it was implemented and how well it worked. Additionally, we don't know if we would be able to modify it to handle holes in polygons, which is discussed next.

Not all rooms consisted of only one polygon. Some rooms consisted of multiple polygons, where one of the polygons was the outline of the room and the rest were holes inside the room. In addition, the resulting polygon after merging two rooms could contain at least one hole, even if none of the original rooms contained holes. Therefore, holes need to be considered when rooms shall be merged. When deciding how to merge two rooms, the right polygon has to be found in both rooms. In addition, it was necessary to be aware of the possibility that the result could be multiple polygons, to make sure that information is not lost.

4.4.1 Merging Multiple Rooms

It has already been explained how two rooms are merged together. Now the process of merging an arbitrary number of rooms together will be discussed. Recall that we find neighbors of each room. If rooms are neighbors, they will be merged. The neighbors for both of these rooms will get updated so that both lists contain all neighbor rooms to the resulting merged room. When iterated through all rooms, all rooms will be merged to as large rooms as possible following the defined constraints. Figure 4.4 shows an example of merging 4 rooms. The result of this algorithm will be merged rooms which normally are a lot bigger than the original rooms.

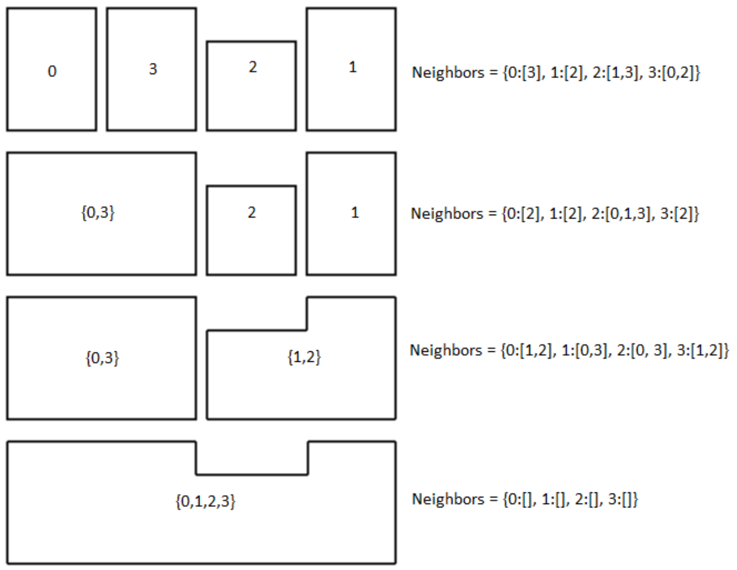

0
3
2
1
Neighbors = {0:[3], 1:[2], 2:[1,3], 3:[0,2]}
{0,3}
2
1
Neighbors = {0:[2], 1:[2], 2:[0,1,3], 3:[2]}
{0,3}
{1,2}
Neighbors = {0:[1,2], 1:[0,3], 2:[0, 3], 3:[1,2]}
{0,1,2,3}
Neighbors = {0:[], 1:[], 2:[], 3:[]}

Figure 4.4: Four rooms merged together, step by step, with the current neighbor list on the right side

4.4.2 Merging Rooms and Hallways

The rooms and hallways would be separated. Rooms would be merged and gradually split. Splitting will be discussed in the next section.
We wanted hallways to behave differently from the rooms. Since they are thought of as roads, gradually splitting a road, wouldn't make sense to the user. Instead hallways will always be fully merged until the user zooms far enough in, where the hallways will be split completely into their original parts. As a result, the hallways will work as roads as long as possible, giving the user a good overview of the hallway network. This is desirable for answering RQ3.

4.4.3 Merging Room Names

Since rooms would be merged together, it would look natural if names are displayed dynamically, corresponding to its underlying geometry. So if room 10, 12, 14 and 16 are merged together, it would make sense to show for example 10 - 16 over the group instead of showing the individual names. This is inspired by the way how hotel rooms are usually displayed on signs. More details would be removed, while we hopefully keep enough detail for people to understand which rooms are in this group. This could make it easier for us to get an answer on RQ2.

4.5 Splitting Room Groups

We had already decided that rooms should be split gradually, so the level of detail will gradually increase or decrease. It will hopefully make it easier for the user to follow if the rooms gradually get smaller as the zoom level get higher. Because the rooms will be changing dynamically, we believed this would not only be better for the user experience but also contribute to answer RQ1.

To make the merged rooms split gradually, we would need some form of structure to organize the rooms in so that we can conveniently split rooms as the user zooms in. For example, a group consisting of eight rooms merged together can then be split into two groups of four rooms each. Afterward, these two groups can be split into four groups of two rooms, and finally, the eight individual rooms are shown. We thought three levels of merging would be appropriate for this application. That means, if a group can be split more than this, we only use its three most split-up levels. We chose the most split-up levels so that the transition to the original rooms is as small as possible, reducing user confusion.

It is important to note that when we talk about ”splitting rooms”, we are technically talking about merging rooms together only to a certain degree, and not actually splitting an already merged group. But it is more natural to think of it as splitting a merged group since we first aim to merge a group entirely to find out which rooms belong together, and afterward, we create smaller merged groups within. Hence, from now on, when ”splitting rooms” is discussed, it is really just rooms being merged together to a certain degree.

4.6 Simplification of polygons

We decided that simplifying polygons on low zoom levels would be beneficial, to remove small details that would be of no interest to the user at this point. This section focuses on algorithms for simplifying polygons.

Merging would already remove a significant amount of details, but some rooms are still quite detailed by themselves. They can contain tiny holes and the polygons can consist of a lot of points. Also, some rooms will be very tiny at a low zoom level. We think details like this would be unnecessary to display at a low zoom level. Hence, we decided to remove small holes and rooms, and simplify polygons by reducing their amount of points on lower zoom levels. This could be an additional contributing factor to answer RQ1 and RQ3.

We would remove small holes and rooms by simply comparing their area to a certain threshold. But polygon simplification would be a more complex problem. There are various algorithms for 2D polygon simplification. Additionally, we thought of making custom algorithms specialized for simple room geometry as well.

However, it became apparent that there was a lot of complex room geometries, where a lot of special cases could show up. Hence, we decided to use a general algorithm that could handle arbitrary geometric shapes. Heckbert and Garland (1997) was our main source for finding simplification algorithms.

4.6.1 Douglas-Peucker Algorithm

We decided to use the Douglas-Peucker algorithm for simplification. It is designed for polylines which could be easily adapted to polygons. We found an existing implementation of Douglas-Peucker (Agafonkin, 2015) so we could focus on the merging which was much more important.

As discussed by Saalfeld (1999) and Da Silva and Wu (2007), Douglas-Peucker might fail topologically if a line is simplified in such a way that the simplified version can interfere with other geometries. However, in this project, this is not considered a problem, since each room usually covers a regularly shaped area which prevents the simplified room from leaving this area. And if inconsistencies should appear, they would only be apparent at a low zoom level. If the user zooms in enough, the original polygons will be displayed, and there will be no inconsistencies.

4.7 Summary

This chapter has given an overview of how the implementation will be, and has created the foundation for diving into the detailed implementation of each step.

Chapter 5

Implementation

This chapter explains how the solution was implemented through text, figures, and pseudocode, to give the reader all the details required to understand the implementation.

At the beginning of each pseudocode, some of the variables that are especially important for the algorithm are briefly explained. The functions used in the pseudocode will be explained in the text, but some simple list operations that are commonly used will be explained here. REMOVE(A, x) is a function that removes element x from list A. CONCATENATE(A, B) is a function that concatenates the two lists, A and B, into one continuous list, and returns this list. ADD(A, x) is a function that adds element x as the last element in the list A. REPLACE(A, x, y) is a function that removes element x from list A, and insert element y where x was inside list A.

5.1 Overall Procedure

This section describes the overall procedure of the merging algorithm. Each step will be very briefly explained, to give the reader an overview of how the entire algorithm is running from start to finish. Every step will be explained in more detail in the following sections. Algorithm 1 describes the function MERGEALLROOMS which contains the major parts of the overall procedure. It takes a list R, which contains all rooms, as input. Each room has a list P of polygons which contain the coordinates, and a list N of neighboring rooms, which is initially empty.

Algorithm 1 Overall algorithm

```
 1: //R = list containing all rooms
 2: //r.N = neighbor rooms of r
 3: function MergeAllRooms(R)
 4:     for each r in R do
 5:         r.N = AddNeighbors(r, R)
 6:     for each r in R do
 7:         for each n in r.N do
 8:             m = Merge(r, n)
 9:             m.N = AddNeighbors(m, R)
10:             Replace(R, r, m)
11:             Replace(R, n, m)
12:     return R
```

It is important to note the difference between a room and a polygon. Figure 5.1 graphically displays how a room is built from points and polygons.

The first step is to find all initial neighbors. The pseudocode for this algorithm is shown in Algorithm 2. Where GetSecondSmallestDist returns the distance between the second closest point in the first polygon to the second, while GetSmallestDist returns the distance from the closest point. OrthogonalsNotIntersecting returns *true* for cases like the one on the left of Figure 4.3 and *false* for cases like the one on the right. Two rooms will be merged if and only if they are neighbors. When neighbors have been generated, the algorithm iterates through all rooms and all neighbors for each room.

Before going any further, the occurring merging case has to be determined. That is, decide if a room consists of one or several polygons. Figure 5.2 shows an example of each case. If a room consists of several polygons, the decision of which polygon to merge has to be made. After this single polygon is found, the algorithm moves on.

The next step is to add any appropriate additional points to each polygon, to make merging easier and avoid too rough approximations.
After any points have been added, the algorithm decides which points in each polygon to use for merging. These points serve as connection pairs, which means that

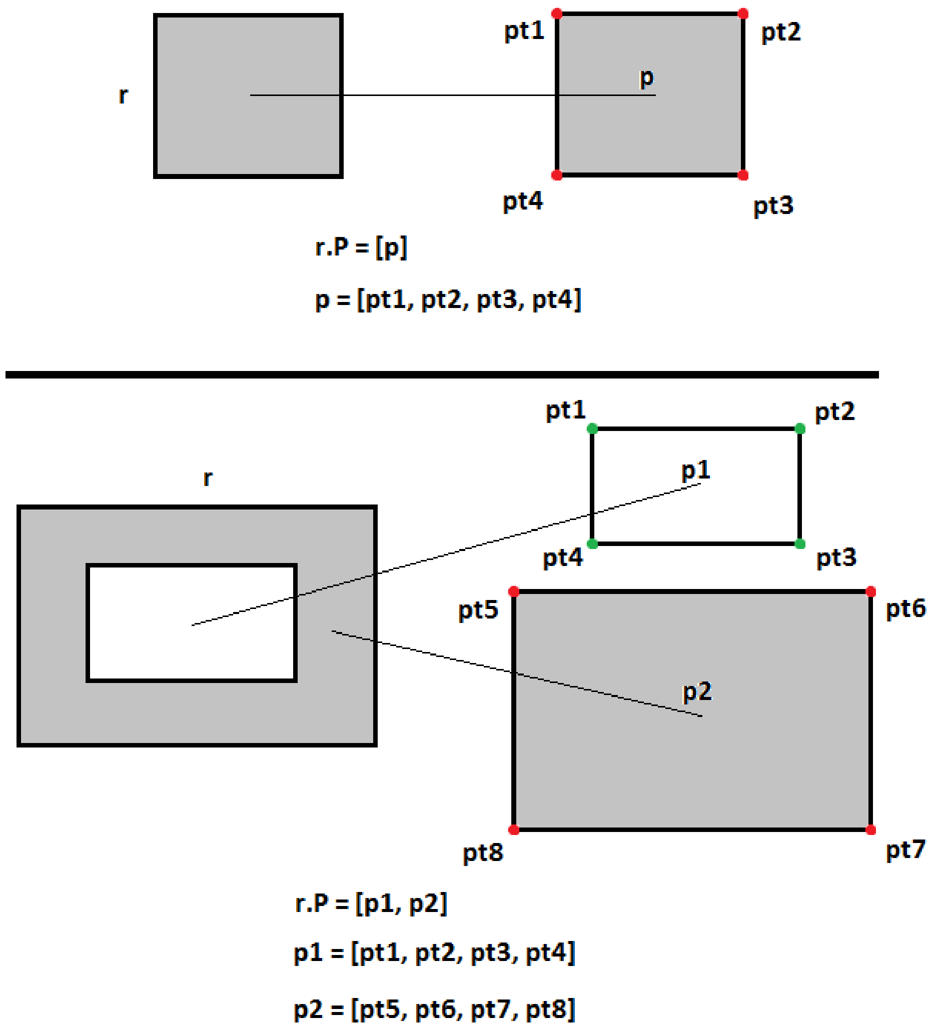


Figure 5.1: Top: Room without holes. Bottom: Room with one hole

the two polygons will be connected together through these points.
Finally, the merging itself is run. Since it is a complex process, it will be explained in detail later. For better understanding, Figure 5.3 describes one scenario of merging two simple polygons. The following sections will present the different steps in more detail.

Algorithm 2 Add neighbor rooms to the neighbor list of each room

```
   //r = room
   //R = list containing all rooms
   //r.N = neighbor rooms of r
1: function AddNeighbors(r, R)
2:     p = GetMergingPolygon(r.P)
3:     for each r_k in R do
4:         if r ≠ r_k then
5:             p_k = GetMergingPolygon(r_k.P)
6:             if GetSecondSmallestDist(p, p_k) < T then
7:                 //Scenarios 1, 2 and 3
8:                 Add(r.N, r_k)
9:             else if GetSmallestDist(p, p_k) < T then
10:                if OrthogonalsNotIntersecting(p, p_k) then
11:                    //Scenario 4
12:                    Add(r.N, r_k)
13:    return r.N
```

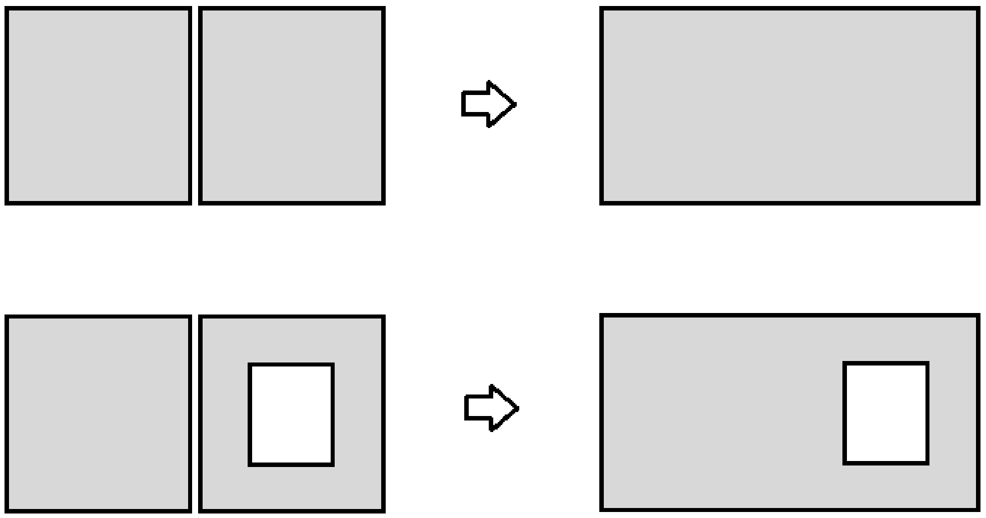


Figure 5.2: Top: Merging without holes. Bottom: Merging with one hole

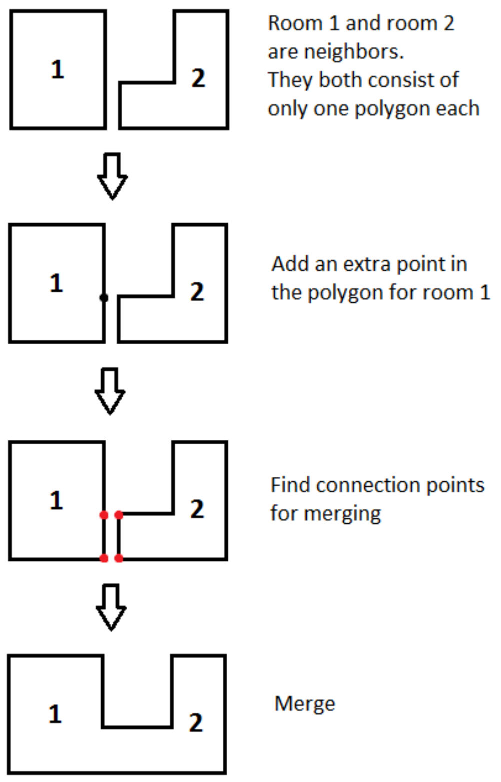


Figure 5.3: Example merging of two polygons

5.2 Premerge

This section explains the procedures that are performed to make all rooms ready to be merged together.

5.2.1 Find Correct Polygons

As mentioned earlier, rooms can consist of an arbitrary number of polygons, so it is important to find the correct polygon when we are going to add neighbors and perform merging. Initially, this was done by finding the area of all the polygons in the room, and select the polygon with the biggest area. This will normally work, but if one room is inside the room it should be merged with, it will not be the best choice. Therefore, the closest polygon is used instead. If more than one polygon is equally close, the biggest polygon should be used. The pseudocode for this is shown in Algorithm 3.

Algorithm 3 Determine if there are several polygons and then find the polygons to merge

```
    //r = room
    //n = neighbor room of r
    //rest = list containing list of holes inside the rooms
    function MERGE(r, n)
        rest = [ ]
        if LENGTH(r.P) == 1 and LENGTH(n.P) == 1 then
            mp = MERGEPOLYGONS(r.P[0], n.P[0])
        else
            if LENGTH(r.P) > 1 then
                $p_1$ = GETMERGINGPOLYGON(r.P)
                REMOVE(r.P, $p_1$)
                rest = CONCATENATE(rest, r.P)
            if n.P.length > 1 then
                $p_2$ = GETMERGINGPOLYGON(n.P)
                REMOVE(r.P, $p_2$)
                rest = CONCATENATE(rest, r.P)
            mp = MERGEPOLYGONS($p_1$, $p_2$)
        ADD(rest, mp)
        r.P = rest
        n.P = rest
        return r
```

5.2.2 Finding Neighbors

The algorithm for finding neighbors is concerned with two rooms at a time. There are four different scenarios that we have defined where it would be desirable to classify the rooms as neighbors. The four scenarios are shown in Figure 4.2. The first three scenarios occur when room 1 has two close points to room 2. The fourth scenario occurs when room 1 only has one point close to room 2.

First, the two closest points in room 1 towards room 2 are found. If the distance from the second closest point in room 1 is below a certain threshold, room 1 and

2 become neighbors of scenario 1, 2 or 3. If only the closest point in room 1 is close enough, another test has to be made to make sure the rooms are placed like in scenario 4. This test checks if the rooms' corners are placed as shown to the right of Figure 4.3, by using the properties of dot products between the points and the other room's line. If that is the case they will not become neighbors.

The upper distance threshold between neighbors will be determined empirically from the existing map.
One important note is that for scenario 3, room 1 will mark room 2 as its neighbor, but not vice versa. Hence, an extra check is performed in the end for all these cases to make sure they are neighbors both ways.

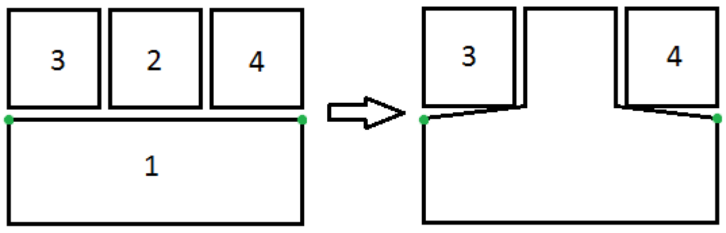
5.2.3 Adding Points

The reason for the addition of extra points in the polygons is not obvious. It makes the merged results look aesthetically nicer, but more importantly, it is also necessary when finding connection points, which will be discussed in the next section. An example where this helps aesthetically is shown in Figure 5.4. The result becomes nicer when the extra points have been added, compared to if they hadn't been added.
By adding extra points, the distance between the points in a pair of corresponding points will be as small as possible and results in shapes that look more like the original shapes.

For each point close enough to the other polygon, a corresponding point is added in the other polygon. In Figure 5.4 when room 1 and room 2 are going to be merged, there are no points in room 1 close to room 2, which means the algorithm will choose the points on the very ends of room 1, marked green. By adding the two points in room 1, marked red, the algorithm will instead choose these points to connect to, resulting in a much more desirable shape. The function that does this is called AddPoints. It takes two polygons as input and returns the polygons with added points.

5.2.4 Finding Connection Points

After points have been added, the majority of neighbor rooms should contain points close to each other that can become intuitive connection points. The next task would be to figure out which points to connect together. In most cases, two



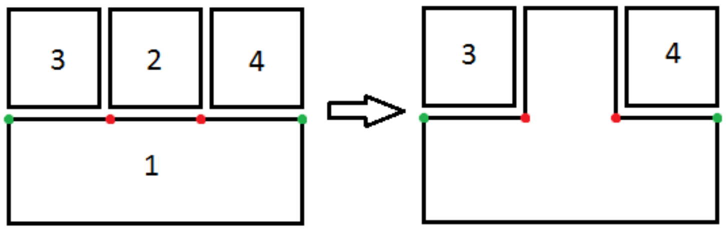


Figure 5.4: Top: Room 1 and 2 are merged straight away, using their original points. This leads to undesirable diagonal lines that almost overlap with room 3 and 4.
Bottom: Two extra points marked with red are added to room 1. This results in a much nicer-looking merging

pairs of connection points will suffice. But more complex cases will arise, so the algorithm should be able to return an arbitrary even number of connection pairs. Algorithm 4 takes both polygons as input and returns a set of points from the first polygon, such that these points can be used as connection points towards the second polygon. Then the algorithm is run again, this time from the second polygon to the first. It finds all points in the first polygon that are closer to the other polygon than the empirically determined threshold mentioned earlier. Afterward, it iterates through all these points and removes redundant points. That is, for every close point, if the two neighboring points to that point in the real polygon are also close points, this point is redundant and not included as a connection point. This process is repeated until all the remaining close points do not fulfill this condition. Figure 5.5 shows how connection points are chosen in both polygons, where their indices are marked in *pairs1* for polygon 1 and *pairs2* for polygon 2. This algorithm makes sure that points 0,3,5,6,7,8 and 10 are not included from polygon 1, and points 0,1,2,4,5,7 and 10 are not included from polygon 2, resulting in the remaining points shown to the right.

Algorithm 4 Get connection points that shall be used to connect the polygons together

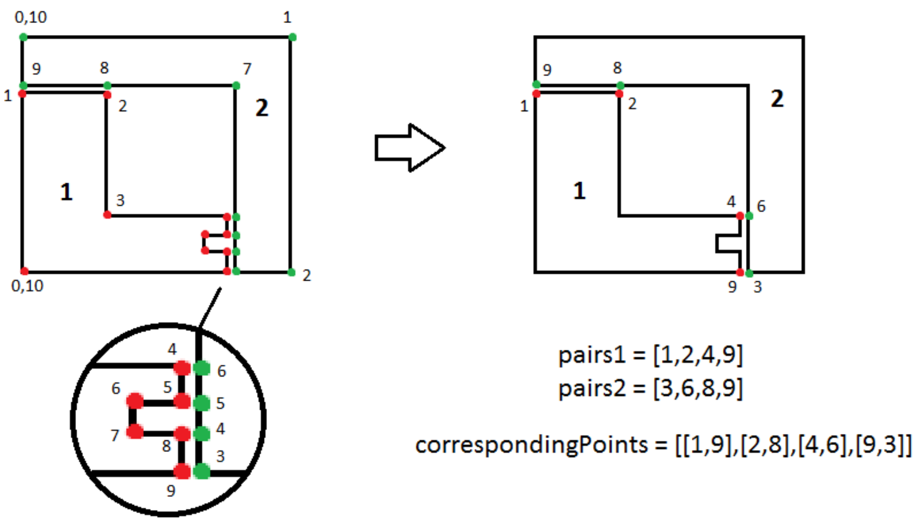
```
    //p_1 = polygon
    //p_2 = polygon
    //pairs = list containing connection points
 1: function GetConnectionPoints(p_1, p_2)
 2:     CP = GetClosePoints(p_1, p_2)
 3:     index = 0
 4:     pairs = [ ]
 5:     for i = 0 to Length(CP)-1 do
 6:         if IndexOf(p_1, CP[i]) $\neq$ IndexOf(p_1, CP[i+1]-1) then
 7:             if index $\neq$ i then
 8:                 Add(pairs, CP[index])
 9:                 Add(pairs, CP[i])
10:             index = i+1
11:     if IndexOf(p_1, CP[index]) $\neq$ IndexOf(p_1, CP[Length(CP)-1]) then
12:         Add(pairs, CP[index])
13:         Add(pairs, CP[Length(CP)-1])
14:     return pairs
```

Additionally, it would be essential to know which points pairs should be connected together. Algorithm 5 arrange the connection points and puts them in a list like the one called *correspondingPoints* in Figure 5.5. In this list, points are arranged in pairs, where each pair will be connected to each other when the rooms are merged. This algorithm first makes sure that if one of the connection point lists is longer than the other, as is the case in Figure 5.6, the missing points (returned by GETUNUSEDPOINTS will be added to the shortest one such that both lists of points will have equal length. Afterward, each corresponding point, based on the smallest distance between them will be added to the resulting *correspondingPoints* list.

Algorithm 5 Find which points that should connect to each other between p_1 and p_2

```
    //p_1 = polygon
    //p_2 = polygon
    //C = list containing corresponding point indices
 1: function GETCORRESPONDINGPOINTS(p_1, p_2)
 2:     C = [ ]
 3:     CP_1 = GETCONNECTIONPOINTS(p_1, p_2)
 4:     CP_2 = GETCONNECTIONPOINTS(p_2, p_1)
 5:     if LENGTH(CP_1) > LENGTH(CP_2) then
 6:         unCP_1 = GETUNUSEDPOINTS(CP_1, CP_2, p_1, p_2)
 7:         for each pt in unCP_1 do
 8:             ADD(CP_2, GETCLOSESTPOINT(pt, p_2))
 9:     else if LENGTH(CP_1) < LENGTH(CP_2) then
10:         unCP_2 = GETUNUSEDPOINTS(CP_2, CP_1, p_2, p_1)
11:         for each pt in unCP_2 do
12:             ADD(CP_1, GETCLOSESTPOINT(pt, p_1))
13:     for each pt_1 in CP_1 do
14:         pt_2 = GETCLOSESTPOINT(pt_1, CP_2)
15:         ADD(C, [pt_1, pt_2])
16:     return C
```

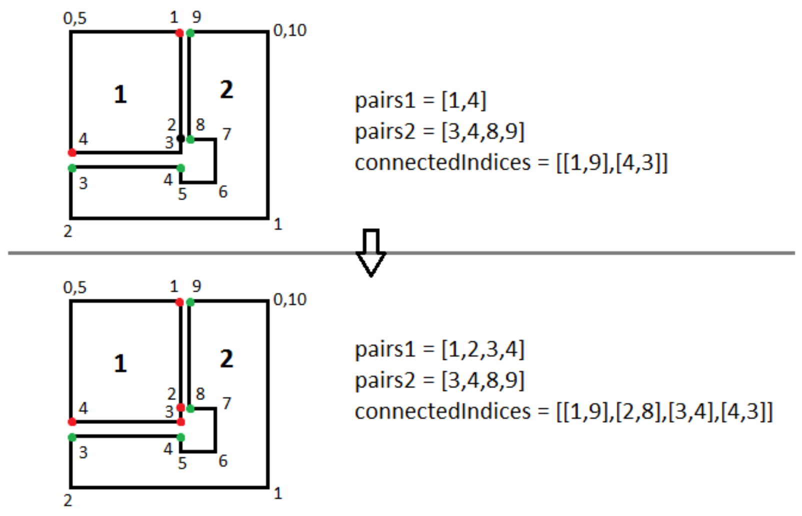
Now we know exactly which points that will be connected together, setting the stage ready for merging polygons.



pairs1 = [1,2,4,9]
pairs2 = [3,6,8,9]

correspondingPoints = [[1,9],[2,8],[4,6],[9,3]]

Figure 5.5: Example of how point indices are connected between two polygons. Note that index 6 and 7 in room 1 is considered close enough to room2, leading to all indices between 4 and 9 being skipped



pairs1 = [1,4]
pairs2 = [3,4,8,9]
connectedIndices = [[1,9],[4,3]]

pairs1 = [1,2,3,4]
pairs2 = [3,4,8,9]
connectedIndices = [[1,9],[2,8],[3,4],[4,3]]

Figure 5.6: Example with different number of pair and its correction

5.3 Merging

This section explains the procedure of merging, which can be performed after the premerging steps are completed. Its main purpose is to make the reader understand the procedure of the most important algorithm in this thesis.

5.3.1 Merge

In the implementation, two different ways of merging are used. One for making resulting polygons containing holes, and one for resulting polygons without holes. Which algorithm that should be used is decided by checking the length of the list with corresponding points. If the list consists of only two pairs of corresponding points the merged polygon should not contain any holes. On the other hand, if the list consists of more than two pairs, the result should be one outer polygon and at least one hole polygon. This is shown in Algorithm 6.

Algorithm 6 Deciding between the two different cases of merging

```
    //p_1 = polygon
    //p_2 = polygon
1:  function MergePolygons(p_1, p_2)
2:      (p_1, p_2) = AddPoints(p_1, p_2)
3:      C = GetCorrespondingPoints(p_1, p_2)
4:      if Length(C) == 2 then
5:          return mergeWithoutHoles(p_1, p_2, C)
6:      else if Length(C) > 2 then
7:          return mergeWithHoles([p_1, p_2], C)
```

5.3.2 MergeWithoutHoles

The pseudocode for merging where the result polygon should not contain holes is shown in Algorithm 7. First, both polygons are divided into two polylines each by the function DivideToPolylines. This function uses the points in *correspondingPoints* to split the polygons into two polylines, where one polyline for each polygon should be used to create the merged polygon. The next step is to find out which of the polylines that should be used. This is done be the function GetPolylineWithPointFarthestAway. In this function, the distance from each point in the polylines to the other polygon is calculated. The polyline that contains the point that is farthest away should be used in the result polygon.

Algorithm 7 Create merged polygon without holes

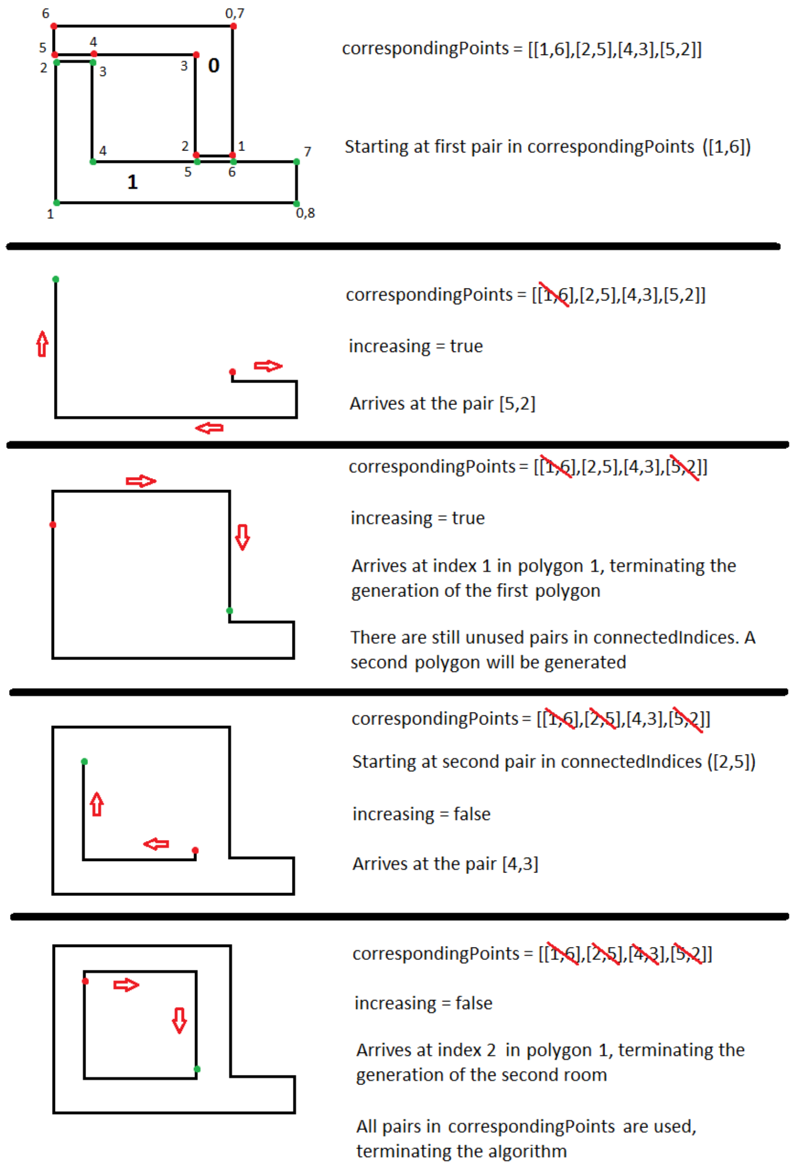
```
    //$p_1$ = polygon
    //$p_2$ = polygon
    //C = list containing corresponding point indices
    //$pl_{ab}$ = polyline number $b$ in polygon $p_a$
1: function MergeWithoutHoles($p_1$, $p_2$, C)
2:     ($pl_{11}$, $pl_{12}$, $pl_{21}$, $pl_{22}$) = DivideToPolylines($p_1$, $p_2$, C)
3:     $pl_1$ = GetPolylineWithPointFarthestAway($pl_{11}$, $pl_{12}$, $p_2$)
4:     $pl_2$ = GetPolylineWithPointFarthestAway($pl_{21}$, $pl_{22}$, $p_1$)
5:     p = Concatenate($pl_1$, $pl_2$)
6:     Add(p, $pl_1$[0])
7:     return p
```

5.3.3 MergeWithHoles

Algorithm 8 describes how two polygons are merged if the result should contain holes. This is also shown in Figure 5.7. First, the indices of the corresponding points are added to arrays. This is done because it makes it easier to keep track of which points that are used and not. The algorithm iterates through points while adding them to the result polygon. It starts with the first corresponding point in the first polygon, then it iterates to the other corresponding point in the same pair, which means the corresponding point in the other polygon. The function GetOuterIndex is just used to find the index of the pair in the *corresponding-Points* to find the correct corresponding points. After that, the function IsIncreasing checks if the next point should be the one in clockwise or counterclockwise direction. How this is done is explained in the next section. Based on the result from IsIncreasing the algorithm iterates through the points in the other polygon in either clockwise or counterclockwise direction until it finds another corresponding point. Then the corresponding point of this point is found, and IsIncreasing is run again to find to correct direction to iterate in, which should be the same as last time. The algorithm iterates through points until it reaches the point it started on. This means that the first polygon is complete, and can be added to the resulting room. This procedure is repeated until all of the corresponding points are used, which means that all the polygons are created.

Algorithm 8 Create merged polygon with holes

```
//C = corresponding point indices
//P = array consisting of both polygons
//i = index of point in either first polygon or second polygon
//$i_p$ = index of polygon. 0 for first polygon and 1 for second polygon
//$i_c$ = index of current corresponding point indices pair
//I = array consisting of $I_1$ and $I_2$
//$I_k$ = list with indices of connection points in room k, k is either 0 or 1
//p = polygon which is created and added to the resulting room, r
function mergeWithHoles(P, C)
    i = C[0][0]
    $i_p$ = 0
    p = r = $I_1$ = $I_2$ = [ ]
    for c in C do
        Add($I_1$, c[0])
        Add($I_2$, c[1])
    I = [$I_1$, $I_2$]
    while I[0].length $>$ 1 or p[0] $\neq$ P[$i_p$][i] do
        Add(p, P[$i_p$][i])
        if p.length $>$ 1 and p[0] == P[$i_p$][i] then //polygon p is complete
            Add(r, p)
            p = [ ]
            i = I[0][0]
        else if contains(I[$i_p$], i) then //p[i] is a corresponding point
            $i_o$ = getOuterIndex(i, $i_p$, C)
            increasing = isIncreasing(C, $i_o$, $i_p$, P)
            remove(I[$i_p$], i)
            $i_p$ = Modulo($i_p$ + 1, 2)
            i = C[$i_o$][$i_p$]
            Remove(I[$i_p$], i)
        else
            if increasing then
                i++
            else
                i - -
    //r is room outline including holes
    return r
```



correspondingPoints = [[1,6],[2,5],[4,3],[5,2]]

Starting at first pair in correspondingPoints ([1,6])

correspondingPoints = [[~~1,6~~],[2,5],[4,3],[5,2]]

increasing = true

Arrives at the pair [5,2]

correspondingPoints = [[~~1,6~~],[2,5],[4,3],[~~5,2~~]]

increasing = true

Arrives at index 1 in polygon 1, terminating the generation of the first polygon

There are still unused pairs in connectedIndices. A second polygon will be generated

correspondingPoints = [[~~1,6~~],[~~2,5~~],[4,3],[~~5,2~~]]

Starting at second pair in connectedIndices ([2,5])

increasing = false

Arrives at the pair [4,3]

correspondingPoints = [[~~1,6~~],[~~2,5~~],[~~4,3~~],[~~5,2~~]]

increasing = false

Arrives at index 2 in polygon 1, terminating the generation of the second room

All pairs in correspondingPoints are used, terminating the algorithm

Figure 5.7: Example of merging with hole

5.3.4 IsIncreasing

The function IsIncreasing returns true if the result polygon after merging should be clockwise, and false if it should be counterclockwise. The function starts at the first point in the connection pair, i.e. the merging point in the first polygon, and iterate through points in a clockwise direction until it arrives at another point that is in *correspondingPoints*. For each point, it is checked if the closest distance from the point to the other polygon is less than a threshold. Equivalently, the points in the other room are checked, but instead of iterate in a clockwise direction, it iterates through points in a counterclockwise direction. If any of the distances calculated is greater than the threshold the function will return false. If all distances are small enough it will return true. An example is displayed in Figure 5.8. The pair in *correspondingPoints* which is considered consists of point 1 in the left polygon and point 9 in the right polygon. If the left polygon is the first polygon, the polylines will be between point 1 and point 2 in the left polygon, and between point 8 and point 9 in the right polygon, as shown in the left image. Since all points in this polyline are close enough, the function returns true, and the result should be in clockwise direction. On the other hand, if the right polygon is the first polygon, the polylines will be from point 9 to point 3 in the right polygon and from point 4 to point 1 in the left polygon, as shown in the right image in Figure 5.8. Then the function will return false since at least one point is too far away, and the merged polygon should then be in a counterclockwise direction.

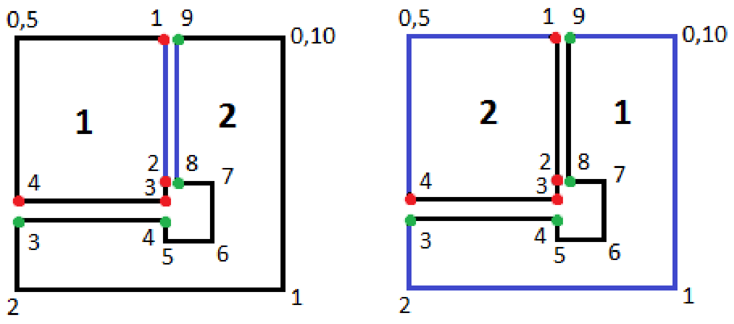


Figure 5.8: Display the difference in polylines checked in isIncreasing if the order of rooms is switched.

5.4 Room Splitting

As we already have discussed, we wanted to implement gradual splitting of merged rooms. Hence, different levels of merging were implemented. At a low zoom level, lots of rooms would be merged together in large groups, while at a higher zoom level, there would be smaller groups and more details. Initially, when creating groups, the groups has the same number of rooms, but since the rooms have very different sizes, we decided that we needed some sort of partition algorithm so we could split large room groups into smaller room groups as evenly as possible based on area. An example of both uneven and even splitting is shown in Figure 5.9.

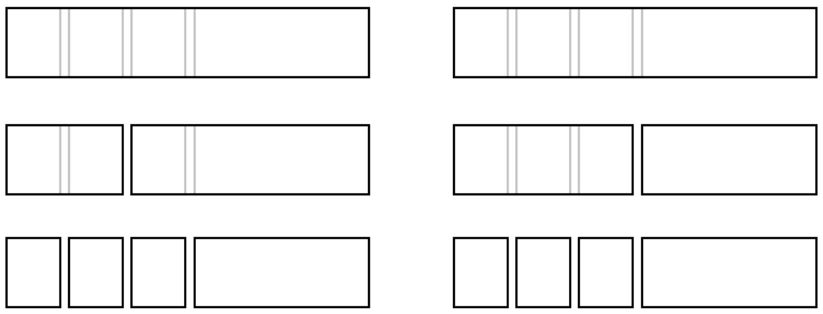


Figure 5.9: Left: Room splitting based on number of rooms. Right: Room splitting based on area. Splitting on area is more even than splitting on number of rooms

5.4.1 Dynamic Programming

Initially, a dynamic programming approach similar to the bin-packing problem (Korf, 2002) was considered. This algorithm puts a given amount of objects with given volumes into packs, such that a minimum amount of packs are used. This means that the objects are packed as evenly as possible. In our version of the problem, rooms would be objects, their areas would be their volumes and all merged rooms would be in the same pack. An additional constraint would be that the algorithm would need to handle ordering since only neighboring rooms can be merged. A merged line of rooms could be evenly split into smaller and smaller groups.

The biggest problem with this approach was that rooms aren't necessarily ordered in one line. They can be cluttered in many ways. This algorithm only works correctly when a room group has two end-rooms: rooms with only one neighbor, but this was far from always the case. If not, it is not guaranteed that every room in the sequence can be merged with the next room and the previous room, rendering the algorithm prone for errors. Figure 5.10 shows some scenarios where room groups have more than two end-rooms. To handle this, we would need to find sub-groups in these groups such that each sub-group only had two end-rooms. This proved to be a very difficult problem since rooms can be positioned arbitrarily relative to each other. Hence, we decided to find another approach that could solve this problem.

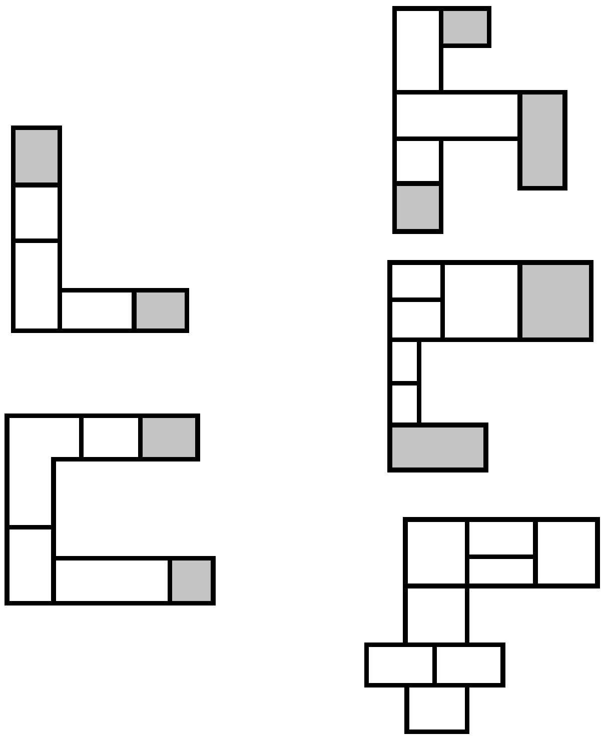


Figure 5.10: Left side: Room groups that will be correctly split. Right side: Room groups that will most likely be incorrectly split. Rooms marked gray have only one neighbor.

5.4.2 Tree Partitioning

Since the dynamic programming algorithm had no guarantee to give the correct results, a better splitting algorithm was needed. We decided to use a tree data structure instead. The reason for this was that a tree can branch out when a room has several neighbors so that it is guaranteed that a node can be merged with its parent and children.
One catch was that cycles can arise since a group of rooms can be in a circle. This could lead to a graph instead of a tree, and this would create the possibility of the

algorithm trying to merge rooms with themselves, which would make the algorithm crash. By simply adding rooms as nodes to the tree only when the rooms hadn't been visited before, no cycles would be created, and the tree structure would be maintained. The disadvantage of having a tree structure is that we're locked to one form, so if this form leads to undesirable splitting, there's nothing we can do about it. However, it is much better than encountering recursive merging, with graphs.

Because of the convenient structure of the tree structure, the splitting could be done by traversing the tree and remove edges between nodes, such that the total area of each resulting sub-tree was as equal as possible. Hence, we decided to use tree partitioning. Our splitting algorithm that creates the tree structure is a hybrid of BFS (breadth first search) and DFS (depth first search). That is, all neighboring rooms of a room is added as its children, and then it moves downward to one of the children and it adds all the neighbors of that child, and it keeps going.
The ordering and splitting procedure can be split into the following steps:

1. Create a tree out of the group of rooms, where each node is a room. Every node has a reference to its parent and its children.

2. Modify the area of each node, such that each node has the sum of the area of itself and all its descendants

3. Split the tree into two sets of two and three sub-trees.

4. Determine which set of sub-trees that have the most even area distribution, and choose these sub-trees.

5. Repeat step 2 to 4 for every sub-tree until the area of the sub-tree is below a fixed minimum threshold, or the sub-tree only contains a single node.

Step 1 and 2 are graphically described in Figure 5.11.

Figure 5.12 shows how step 3 works, and how the resulting room groups look like. The choice between these sets is determined by Formula 5.1 and 5.2. a_i is the area of sub-tree i and A_2 is the area of the set of sub-trees in the two split. In the three split, a_j and A_3 are used instead. d_2 and d_3 are the largest differences in area between one of the sub-trees compared to the area of the entire set of trees, for a split in two and three, respectively. D is the minimum of these two differences.

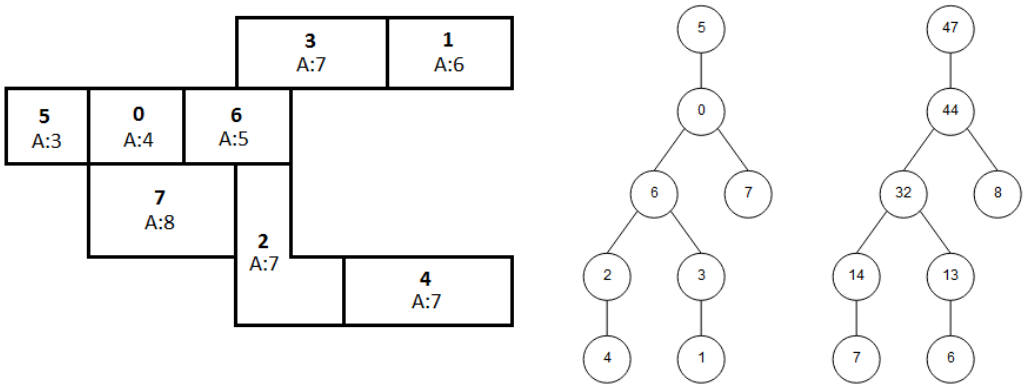


Figure 5.11: Left side: The original rooms that have been merged together. Middle: The tree structure that shows how the rooms are built into a tree. Right: The tree structure where each node contains the sum of the area of itself and all the nodes below it.

If D equals d_2, splitting in two gives the most even result. Otherwise, splitting in three is most even.

$$d_2 = \max\left(\left|a_i - \frac{A_2}{2}\right|\right) \quad d_3 = \max\left(\left|a_j - \frac{A_3}{3}\right|\right) \tag{5.1}$$

$$D = \min\left(d_2, d_3\right) \tag{5.2}$$

We had to decide which thresholds to use when splitting room groups. If we used a global one, it could be good if most groups consisted of similar sized rooms. But if one group had large rooms, they would be split immediately, or if one group had small rooms, they would never be split. Local thresholds for each group was another possible solution, but it could lead to disproportionately different groups, for example, if one group with large rooms is side by side with a group with small rooms.

Since the sizes of rooms varied greatly, this could lead to a lot of smaller groups never being split if a global threshold was used, which we believed was a much worse trade-off than potentially disproportionately different groups, so we decided to use local thresholds. These thresholds were calculated for each merged level, using the formulas in 5.3, where A is the total area of the groups and N is the number of rooms in the group. T1 is the upper threshold for the smallest merged

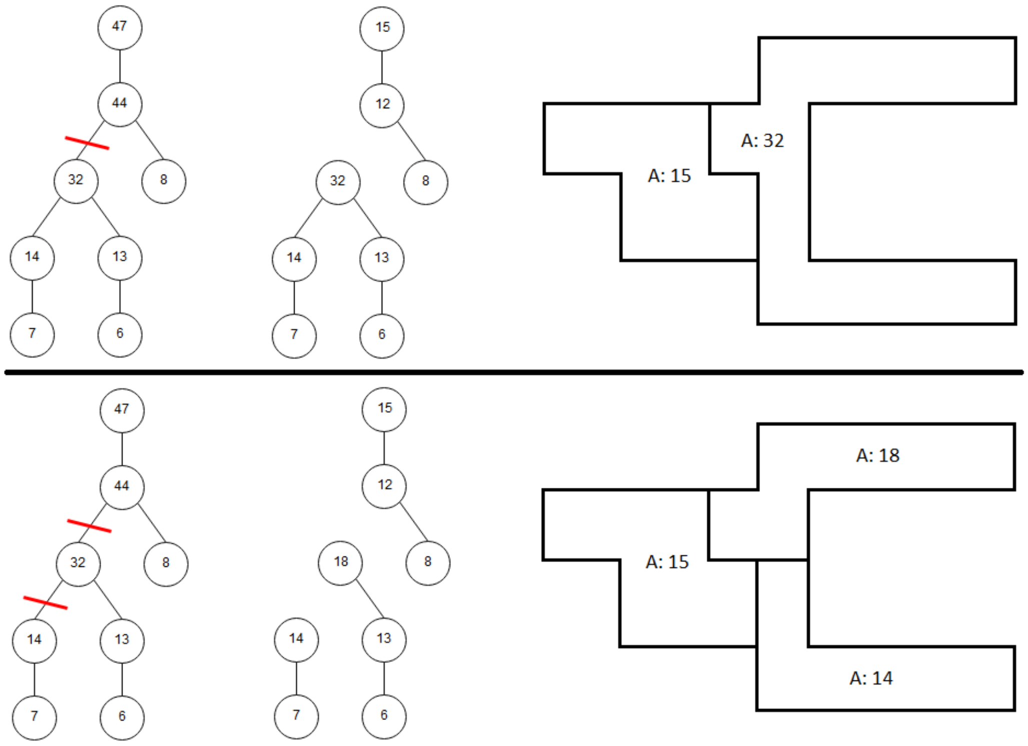


Figure 5.12: Top: Tree is split into two and the resulting group split is shown. Bottom: Same as top, except that the tree is split in three

groups, T2 is the upper threshold for the next level, and T3 for the top level. The constants in the denominator are empirically found so that it fits our application.

$$T3 = \frac{A}{\frac{N}{10}} \quad T2 = \frac{A}{1.1 + \frac{N}{10}} \quad T1 = \frac{A}{2 + \frac{N}{10}} \tag{5.3}$$

The number of times a room group could be split varied a lot. For example, a small group of two rooms could be split once, while a group of twenty rooms could be split, say four times. We wanted room groups to be as equal as possible, so we decided to split all groups as much as possible. Then for each group, we would choose to only use the three sets of sub-groups that were most split. If a group was split less than three times, we would show the least split group several times.

One disadvantage with the tree partitioning algorithm is that the order of how the rooms are added to the three can have a large impact on how even the splitting can get. We choose the room with fewest neighbors to increase the chance of creating a deep and narrow tree instead of a wide and shallow tree. If a tree is wide and shallow it is much more likely to give uneven splitting. This can happen even when choosing the root node with the fewest neighbors. Figure 5.13 shows an example of this.

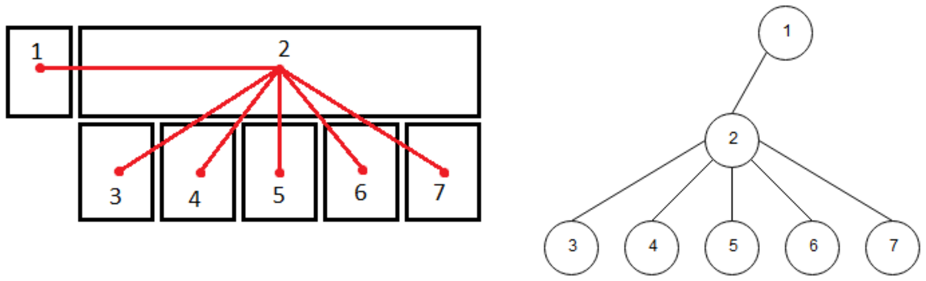


Figure 5.13: The tree is created in an order such that it becomes wide and shallow. No matter which of the edges we cut, the area distribution will always be very uneven.

5.5 Merging Room Names

Since rooms are merged together, showing all room names can be confusing since the room names don't correspond with the underlying polygon. Hence, an adapted name displaying scheme was necessary. A better idea was to show all room names in a group in some shortened way. If the name was too long, it could occupy too much space on the map and overlap with other names. We decided to display the names in form of a range. For example, this group consists of rooms 120 to 130, similar to hotel room signs. Additionally, since some room names could be quite long, we decided to only show a certain postfix of the last room, to shorten the name length. In a group, the name with the lowest ASCII value and the name with the highest ASCII value would be extracted and used in the resulting range name. Figure 5.14 shows an example. Now the room names and polygons made much more sense together.

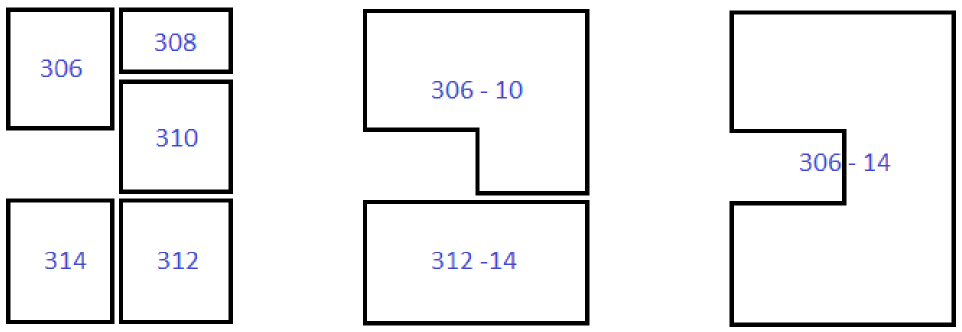


Figure 5.14: Room names are merged together and sorted so that it shows a range of what room names are in this group

5.6 Rendering

Fast rendering is important for the user experience, so the user won't become impatient and annoyed by slow response. In this section, we describe how we are rendering polygons and names.

5.6.1 Polygon Rendering

We decided to render canvas elements for our approach since the user interactions affect the entire screen (dragging and zooming). There was no need for us to be able to manipulate specific parts of the screen, which is one of the strengths of SVG.

We were more uncertain about using immediate mode rendering (updating the entire screen at once) or tiled rendering for our solution. We thought that if the rendering was fast enough, updating the entire screen would be desirable. However, if the rendering proved to be slow enough to annoy the user, we would use the tiled rendering instead.

The graphics were separated into different layers. Original rooms, doors, stairs, outlines, etc. This made it very convenient to give each type of graphics custom properties.

The Leaflet library supported both immediate mode rendering and tiled rendering. Additionally, the Leaflet library contained a feature which allowed us to apply a

Graphics Layers	Zoom Level Interval
Outlines	16.5 - 25
Simplified Merged Large Rooms	17 - 18
Simplified Merged Hallways	17 - 18
Unmerged Large Rooms	17 - 18
Merged Hallways	18 - 21
Unmerged Rooms	18 - 20
Merged Large Rooms	18 - 18.5
Merged Medium Rooms	18.5 - 19.5
Merged Small Rooms	19.5 - 20
Rooms	20 - 25
Stairs	20.5 - 25
Hallways	21 - 25
Doors	21 - 25

Table 5.1: Listing of the different layer groups and which zoom levels they are displayed on

rendering technique for each layer group so that we could experiment with different combinations.

Table 5.1 shows the different graphics layers and when they are displayed. The further the user zooms in, the higher the zoom level gets. On low zoom levels, larger merged room groups are displayed, while as the zoom level increase, these groups gets split into smaller groups, until the original rooms are shown. Simplified means that the polygons have been significantly simplified as an attempt to get rid of small details. These are displayed at the lowest zoom levels.

5.6.2 Name Rendering

Names were rendered as Leaflet markers. Name markers were organized in corresponding layer groups as the polygon layer groups, which was stated in the previous section and shown in Figure 5.14. That means there is a corresponding *Merged Large Room Names* layer group to Merged Large Rooms in Table 5.1, *Merged Medium Room Names* layer group to *Merged Medium Rooms* in the same table, and so on.

Collision Grouping

One major difference between the polygon layers and the marker layers was that marker layers included collision detection.
When showing room names, there is a risk that they will be cluttered and overlap each other, because of long names of groups or rooms being close to each other, as displayed in Figure 5.15. This would negatively affect the user experience and should be avoided. Luckily there was a built-in functionality in the Leaflet library that handled collisions. If names collided, one of them would not be rendered. The additional neat thing about this functionality was that it made us able to modify the margin between the names. So on a lower zoom level, the margin could be increased to decrease the level of detail, while the margin would decrease as the user zoomed in.
One undesirable effect that appeared from using this feature, was that it didn't care about what names it chose to show. So if a name pops up at low zoom level, this name might disappear again when zooming in, because of another name close by just happened to be rendered instead. This inconsistent rendering can be confusing for the user.

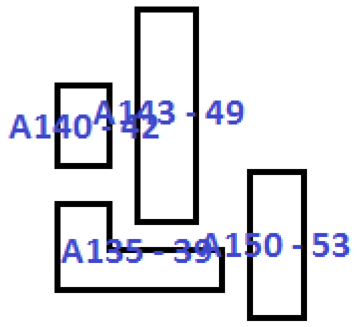


Figure 5.15: Names of room groups are overlapping each other, which makes it very difficult to read what it says

5.7 Testing of Implementation

It was important to test the implementation to verify its correctness and performance. Undesirable behavior or slow performance would be quickly noticed by the user and impact the experience in a negative way. This section describes different ways we tested the technical aspects of the algorithms, both correctness, and performance.

5.7.1 Polygon Merge Testing

Merging of polygons was the main task of this thesis, so it was crucial to get the merging right. One obvious way to test merging was to simply look at the result and see if it made sense according to what we expected. If it didn't there would usually be noticeable artifacts, such as walls overlapping rooms, that could be analyzed.
During an occurrence of incorrect merging, we would identify the two polygons that created the undesirable behavior. That is if it looks correct before these two polygons are merged, and incorrect after these two are merged, we know these two polygons are causing the problem.
The most common reasons for undesirable merging is when wrong connection points are connected together, so this would be the first thing to be analyzed. A marker would be placed at each connection point, displaying its index. The points involved in the connections that didn't make sense were the cause of the problem, and the code could be evaluated from this point.
If an error occurred during polygon merging, we rendered the two polygons that the program tried to merge, at the point of time the error was thrown. Then we could follow the procedure that was just explained.

5.7.2 Performance Testing

Our map solution should be fast and smooth enough to use so that people will have an overall better experience. If choppy movements and slow rendering of graphics occur the user can get impatient and annoyed. Hence, it is essential to test the performance of the solution and remove unnecessary processing.
The JavaScript performance tab could give us useful information about what processes were running at certain times. This could be very helpful for identifying unnecessary processes and optimize the solution. For example, if a compiler is running while rendering graphics on the map, there's undesirable behavior in the rendering.

5.8 Summary

This chapter has given a detailed description of the implementation, to give the reader an understanding of how the implementation works.

Chapter 6

Experiment

We needed empirical data to find out what users thought of our solution, compared to a more classical solution. Hence, we designed an experiment to get useful feedback from users. This section explains how the experiment was set up and why the specific test decisions were made.

The application was very subjective, so there was no obvious way to design user tests. It would be desirable to get quantitative data from wherever it was possible, but we realized that most things could only be measured in a qualitative way. Still, we decided to design user tests that had a mix of both.

The tests would be performed on two solutions. Our solution, showed in `http://folk.ntnu.no/morteano/Dynamic/Map.html`, would be one of them, while the other solution would be more similar to how MazeMap is represented today, and can be found in `http://folk.ntnu.no/morteano/Classic/Map.html`. Our solution will be referred to as the *new solution*, while the old solution will be referred to as the *classical solution*.

6.1 Quantitative Testing

Typical tasks for a user navigating in an indoor map, are finding a specific room and figure out directions to get to a specific destination. Hence, we included the following user tasks in the test:

1. Given a room name, find the room on the map, by only moving and zooming with the mouse.

2. Given two rooms on the map, draw a path between the two rooms, simulating where you would go in real life.

We believed these were highly relevant tasks since navigation and orientation are what a typical user would be doing in a digital map. To get quantitative data, every test subject would be timed on every task. Hopefully, we would see a pattern in the time spent between the test subjects. Every test would start from the same specific point on the map, and have the same initial zoom level. Each test would be performed on both solutions, so they could be compared with each other. To reduce memory bias, half of the subjects would first test on the classical solution and then the new solution, while the rest would test vice versa. Task number one would be performed twice with two different rooms, to see if the test subject got more used to the map or not, after the first time.

The main purpose of task one was to test the significance of the room labels (POIs). We expected that people who were searching for a room would mainly focus on the labels. This had the potential to give clear feedback about how people used the different label representations, and give an indication to RQ1.
In task two, we expected the hallways to play a large role. Hallways would be natural to use when walking around in large buildings, so we hoped this would make the user notice the different hallway representations in the two solutions, and that the time results would show a pattern and give some results to RQ3.

We chose the ”Realfagsbygget” building as test building since it has a lot of different rooms and the hallway network is quite large. Most of the room names in ”Realfagsbygget” follows the same standard where the first symbol is a letter, the next specifies which floor it is in followed by a dash and then the room number, for instance, B1-177. We assumed that it could be confusing if the merged names also used a dash to separate the rooms, for instance, B1-177 - 79. Therefore, the dash was removed from the names in the solutions used in the experiment. In addition, the digit showing the floor number was also removed, because it was not necessary when only the first floor was used. The result was, therefore, for instance, B177 - 79. These simplifications were done to try to make sure the test subject focused on what we wanted to test instead of being confused about the room names.

Until now, room labels and hallways has been mentioned as being in focus in the quantitative testing, but the main part of our thesis, room merging, hasn’t been in focus. We couldn’t come up with any quantitative way to test this, so we decided to test this qualitatively.

6.2 Qualitative Testing

After the test subjects had completed the tasks, we would have an interview with them, to collect qualitative data. This would be executed in the form of an interview with questions, where they had to give explanations for their answers. We believed that having a simple questionnaire would be difficult for the test subjects to answer because it would be hard to create questions that could be answered by ticking boxes, and at the same time give us insightful knowledge.
In every interview, one of us would explain the tasks and ask questions, while the other would take notes and take care of the practicality of the tests. We also aimed to record every interview so we could make sure that we didn't miss any useful information.
Even though we were going to test room names and hallways with specific tasks, we also included questions about these in the interview, since they would be a natural point to bring forward for discussion. This could be helpful to avoid making wrong conclusions, based purely on quantitative data, since we didn't know how useful the quantitative data would be.

We made the following questions to cover as much as possible in the interview. The questions were concrete and the expected answers should give clear hints of which solution they preferred with that topic in mind unless they weren't sure. We expected cases where people would give answers to several questions during their explanations, so there would be no point asking questions that were already answered.

Q1: What are the biggest differences you noticed between the two solutions?

Q2: Did you notice that rooms are being merged together?

Q3: Did you notice that hallways are being merged together?

Q4: Did you understand the merged room names?

Q5: Do you think it is helpful that rooms are being merged?

Q6: Do you think it is helpful that hallways are being merged?

Q7: Do you think it is helpful that names are being merged?

Q8: Is merging of information confusing?

Q9: Which of the solution did you prefer and why?

Q10: Do you have any suggestion to improvements in the solutions?

Q11: Do you have any other comments about the solutions?

Q1 was going to check if they noticed any differences between the maps, to see what features drew the most attention.
Q2, Q3, and Q4 aimed to document their understanding of the new map.
Q5, Q6, and Q7 aimed to see if these features were actually helpful for navigating on the map to give an indication to RQ2, RQ3, and RQ1, respectively.
Q8 was more of a general question about more details compared to fewer details.
Q9 was the pivotal one where they had to decide between one of the two, and give valid reasons for their choice.
Q10 and Q11 were there to get potential extra useful ideas that we might had not considered ourselves.

During the interviews, the users had the two solutions side by side, so they were able to compare the two solutions when answering.

6.3 Participants

We wanted to perform the tests on people who had experience with MazeMap, as we expected they had the most useful feedback to give us. Additionally, we wanted to test on people who were not that familiar with MazeMap, to see if we could find notable differences. The first group would consist of employees of MazeMap, while in the other group, people would preferably have no more than limited experience with MazeMap. It would be desirable with an even number of test subjects from each group since that would lead to an equal amount of tests starting with the classical solution and tests starting with the new solution.

We knew MazeMap had at least ten employees, so we decided we wanted to interview as many of them as possible and we would interview other people so that the total number of participants would hit 20. It would be most desirable to get ten of each, so we could directly compare the two samples and see if there were notable differences between experts and casual users.

6.4 Summary

In this chapter, the design of the experiment has been presented, where the aim is to collect useful empirical data about our solution.

Part III

Evaluation, Results, and Discussion

Chapter 7

Results

In this chapter, results are presented, to discuss the rate of success of our solution. The results are divided into technical results and results from the experiment, for better structuring.

7.1 Technical Results

In this section, strictly technical aspects of the thesis are presented and reasoned with.

7.1.1 Imperfect Data

The data we fetched from the server wasn't always as we would expect. This was especially noticeable with doors. Instead of a door being drawn as a door polyline, a door-shaped cut in the wall is drawn. One example is shown in Figure 7.1, where the correct doors are rendered as black polylines, while the wrong door is included in the room instead of being its own polyline.
This can happen because MazeMaps algorithm sometimes incorrectly interprets the floor plan, and generates a polygon with a door-shaped cut instead of generating an individual door.

7.1.2 Polygon Merging

The merging algorithm results in correctly merged polygons in the majority of test cases. When tested on the first floor of the entire campus of Gløshaugen, most buildings had all their rooms merged correctly. Figure 7.2 shows how a group of

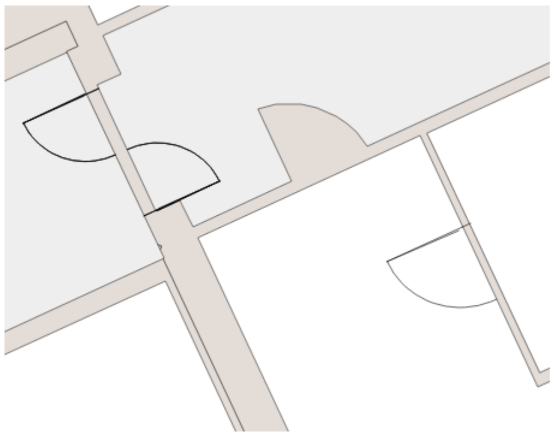


Figure 7.1: The two doors to the left and the bottom door are correctly drawn as individual polylines, while the door at the center is incorrectly integrated into the wall

rooms is gradually split until the original rooms are shown. Figure 7.3 compares the classical and new solution, with focus on the hallways. To get a better understanding of the overall difference, Figure 7.4 shows a screenshot from the new solution beside a screenshot from classical solution.

Of the total of 46 buildings tested, only two buildings gave undesirable merging results, which is a building success rate of 95.7%. But this percentage is not a informative value, since there are many polygons that are merged together inside each building. In the two buildings that failed, there was only one merging case in each that failed. Hence, the rate of success is much higher in the context of every single merging case.

Both of the buildings that failed, completed the merging process, but they had an inverted merge result, shown in Figure 7.5 and Figure 7.6. The reason this happens is that there are special cases where an equal number of connection points is found in both polygons to be merged, but these points do not correspond to each other in the way that was desirable, leading to incorrect connections between the two polygons. Otherwise, the algorithm proved successful both on rooms and hallways.

Another less important but notable thing was that the distance threshold between

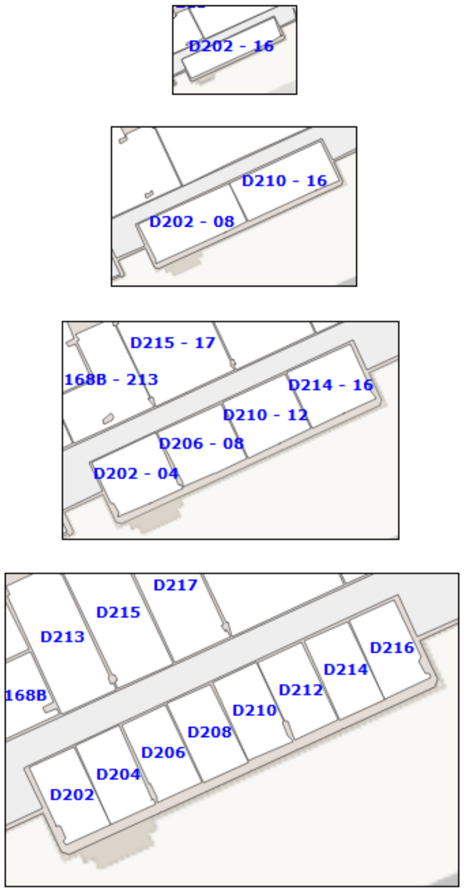
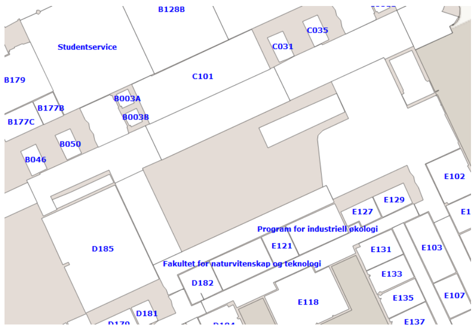


Figure 7.2: A room group is fully merged on top and gradually split as the user zooms in, until all the original rooms are shown. All images are in their original scale

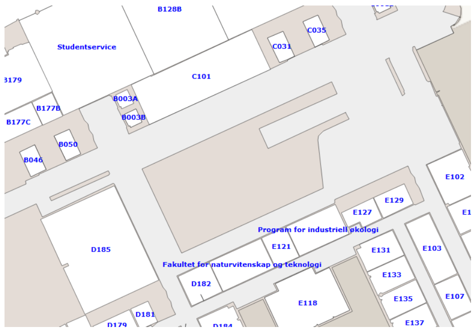
polygons for merging wasn't optimal for all cases. The most notable case is the one shown in Figure 7.7, where one hallway is not merged together with its two neighboring hallways, resulting in two clear gaps in an otherwise fully connected hallway network. The distances over these gaps were larger than this threshold, creating these undesirable gaps.

7.1.3 Name Merging

The merging of names was an overall success. The algorithm managed to handle the majority of cases and produce merged names such that it ranged from the low-



(a) Hallways in classical solution



(b) Merged hallways in new solution, colored gray

Figure 7.3: Comparing the hallways in the classical and new solution

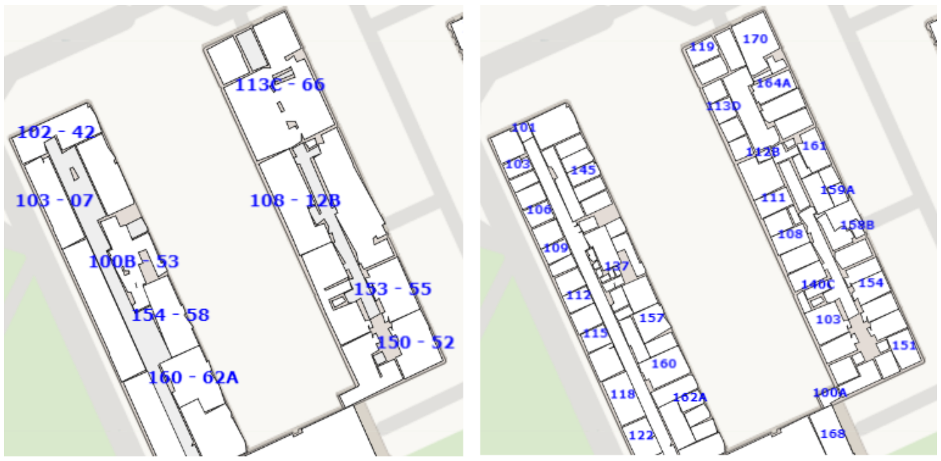


Figure 7.4: Comparison of the new and classical solution

est name to the highest name. There was one case where the algorithm gave an undesirable result. It created a room name called ”4.111 - K26”, which shows no intuitive relation. This happens because there is a room group that contains rooms following the norm ”4.1xx” and one single room named ”K26”. The algorithm checks if more than the last three characters is different. Since ”K26” only has three characters, this test mistakenly passes and the room is marked as related to the other rooms.

One direct effect of removing merged room names where the first and last name is not related, is that some areas can seem more empty than others. This happens be-

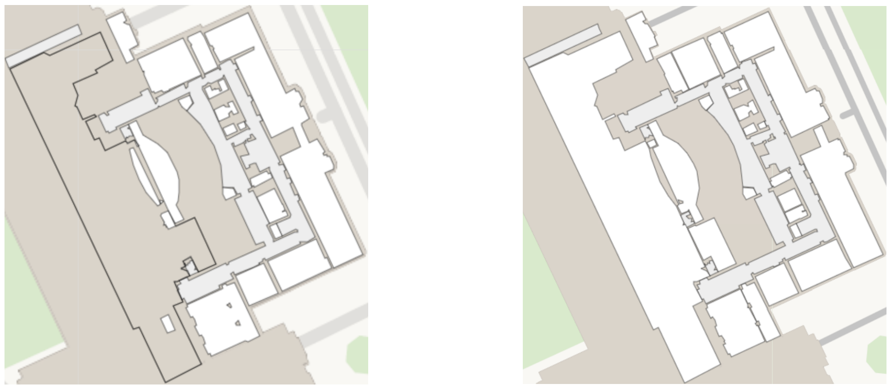


Figure 7.5: Left: Undesirable merging result. Right: Rooms before merging

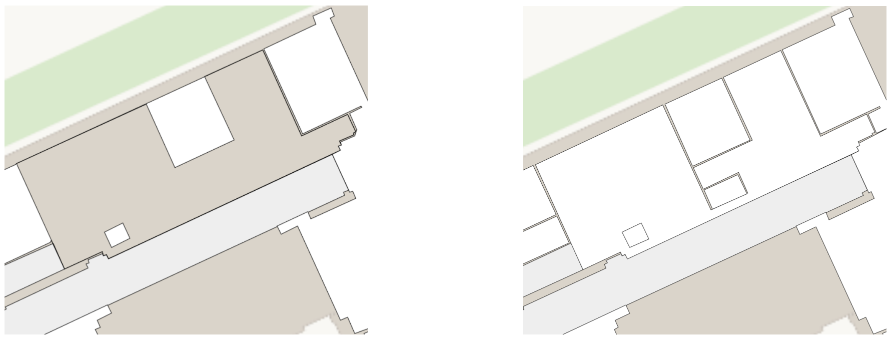


Figure 7.6: Left: Undesirable merging result. Right: Rooms before merging

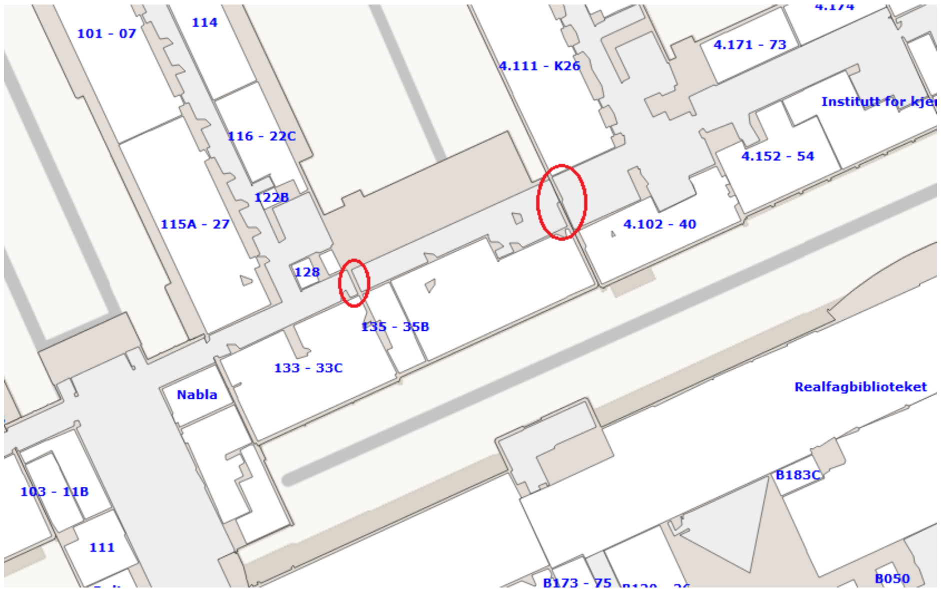


Figure 7.7: Two gaps between the hallways appear because the distance is higher than the threshold

cause these kinds of names are set as an empty string, so they're on the map, even though they can't be seen. This prevents names that are close by to be rendered because of the collision grouping.

7.1.4 Polygon Splitting

The splitting algorithm that splits polygons for different zoom levels was a success. It handled all cases without any errors or artifacts. This shows that the tree partitioning algorithm was an excellent choice for creating different levels of detail.

There were a few cases where the rooms could have been split more evenly. Figure 7.8 shows two examples where the areas of the rooms are quite uneven. As explained earlier, a room won't be split unless it exceeds a certain threshold. In the case to the left, it just so happens that one room is split because it is slightly larger than the limit while another room is not, because its area is just below the limit, leading to one large room beside several small rooms. The case to the right of Figure 7.8 occurs since the function *CreateTree* creates a wide and shallow tree, resulting in an uneven splitting.

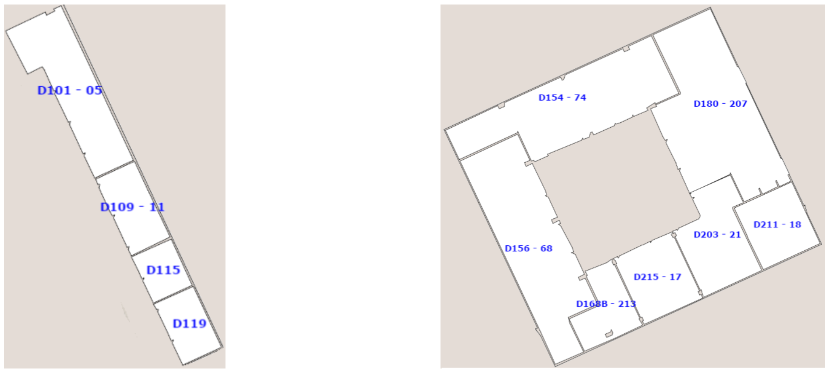


Figure 7.8: Two cases of uneven merging

We believe the issue with uneven splitting is not a major issue, and it wouldn't affect the user experience significantly.

7.1.5 Reduction of Points

Even though the main focus has been on a qualitative visualization of the map, the solution also gives possible quantitative technical improvements. The simplification and merging lead generally to a large drop in the number of points to render. Table 7.1 contains the number of points for the different zoom level intervals. Since the classical solution uses the same amount of points as the zoom level interval from 21 to 25, the number of points is noticeably reduced for zoom levels below 21 in the new solution. This can possibly result in reduced rendering time, but since it has not been tested in this project, we can't draw any conclusions from it.

Zoom level interval	Points in rooms	Points in hallways	Sum of points	Reduction of points [%]
17 - 18	1814	775	2589	89.1
18 - 18.5	7124	2168	9292	61.0
18.5 - 19.5	9888	2168	12056	49.5
19.5 - 20	10706	2168	12874	46.0
20 - 21	17607	2168	19775	17.1
21 - 25	17607	6247	23854	0

Table 7.1: Displaying the number of point for different zoom level intervals

7.2 Experiment Results

There were twenty participants in the experiment. Nine of them were MazeMap employees, while the rest were other employees or students. All of them were generally proficient with a computer.
We ended up recording only the twelve first participants since we were sitting in an enclosed environment with each one. The remaining eight participants were tested in a reading hall with other students in it, so we dropped recording, because of background conversations. The quantitative results of the experiment are shown in Table 7.2 and Table 7.3, which show how long time the users needed to complete each of the tasks. Table 7.2 shows the results where users used the classical solution first, and then the new solution, while Table 7.3 shows the results of the users who started with the new solution.
The rest of this section will present the results from the questions presented in Section 6.2.

7.2.1 MazeMap Employees vs Casual Users

There were no distinct trends or differences between the results from the MazeMap employees and the casual users, both in the quantitative and qualitative results. This was unexpected since we anticipated that the MazeMap employees would give better quantitative results, and also give more detailed feedback because they work with indoor maps daily. In both groups, there were fast and slow people and similar observations and comments were mentioned. In general, there were no standout opinions from one of the groups compared to the other. Hence, all results are weighted equally.

	Classical solution			New solution		
	Locate B175	Locate E143	Draw path 1	Locate A142	Locate 4.126	Draw path 2
	14.71	10.95	10.85	9.00	22.28	10.14
	8.63	31.71	26.14	10.91	46.75	36.66
	58.08	97.52	141.35	7.57	75.63	57.20
	18.78	17.34	24.97	9.06	20.91	28.67
	70.17	13.28	32.71	17.18	35.42	33.81
	21.90	12.88	57.20	6.55	15.11	14.58
	47.75	34.87	48.54	42.37	13.37	16.12
	11.33	37.19	36.30	26.68	27.73	30.00
	48.34	4.01	26.69	14.93	58.49	25.37
	29.94	17.84	32.04	69.14	13.48	10.42
Mean	**32.96**	**27.76**	**43.68**	**21.34**	**32.92**	**26.30**
STD	**21.59**	**26.89**	**36.64**	**20.09**	**21.18**	**14.50**

Table 7.2: The time, in seconds, each user used to complete the tasks during user testing when solving the three first tasks in the classical solution and the three last tasks in the new solution

7.2.2 Answers to the questions

The results of the each question, that was mentioned in Chapter 6, are summarized here. Table 7.4 shows a shorter version of the results.

Q1: What are the biggest differences you noticed between the two solutions?

When the test subjects were asked which differences they noticed, 18 of them mentioned the merged room names, while 5 noticed that the polygons get merged as well. The difference hallway colors were noticed by 3 participants, and 2 of the participants said they were too busy completing the task to notice any differences at all.

Q2: Did you notice that rooms are being merged together?

After completing the tasks, 5 of the participants had noticed that rooms were merged together in one of the solutions, while 15 didn't notice this difference. Of those 15, 2 said that it was probably a good thing that they didn't notice since it indicates that it happens in a smooth and intuitive way, that doesn't distract them.

	New solution			Classical solution		
	Locate B175	Locate E143	Draw path 1	Locate A142	Locate 4.126	Draw path 2
	29.32	10.60	25.93	25.45	18.98	32.04
	24.75	8.22	48.47	18.44	28.94	40.42
	33.64	13.86	43.16	36.75	8.83	52.30
	21.18	13.55	25.63	10.72	16.96	71.24
	48.93	22.42	60.61	48.89	31.43	21.95
	31.25	9.68	57.86	10.19	17.31	36.57
	23.59	12.25	30.42	13.87	43.89	27.93
	35.71	28.05	29.51	18.25	9.44	57.26
	63.19	13.04	42.99	8.98	9.23	46.83
	41.79	5.75	57.26	184.52	10.07	27.55
Mean	**35.34**	**13.74**	**42.18**	**37.61**	**19.51**	**41.41**
STD	**12.95**	**6.70**	**13.71**	**53.19**	**11.77**	**15.47**

Table 7.3: The time, in seconds, each user used to complete the tasks during user testing when solving the three first tasks in the new solution and the three last tasks in the classical solution

Q3: Did you notice that hallways are being merged together?

None of the participants noticed that the hallways were being merged during the tasks.

Q4: Did you understand the merged room names?

15 test subjects said they understood the merged room names, while 5 said that they didn't understand it. In addition, 1 of the participants that understood it, said that it was not very intuitive, and meant it could be improved.

Q5: Do you think it is helpful that rooms are being merged?

15 participants said that room merging was helpful, 3 meant that it didn't matter or they were unsure if it was helpful and 1 said it was a disadvantage since it was confusing and made it difficult to know if a room would be split further on a higher zoom level. 1 of the participants didn't answer this question.

Q6: Do you think it is helpful that hallways are being merged?

When the participants were asked if merging of hallways was helpful, 11 of them said that they thought so, while 3 said that something helped, but they couldn't tell if it was only the color, or that the merging also helped. 6 of the participants didn't answer this question.

Q7: Do you think it is helpful that names are being merged?

Of the 20 people that participated in the experiment, 19 said that the merging of names improved the user experience, while the last participant said that he/she thought it didn't at that moment, but it had potential to improve the user experience.

Q8: Is merging of information confusing?

2 of the participants said that the merging was confusing. 4 said that it was initially confusing, but they understood it after using it for a short while. 14 of the 20 participants said that the merging was not confusing.

Q9: Which of the solution did you prefer and why?

19 of the 20 test subjects preferred the new solution. The last participant said he preferred the new solution if he should find a path between two locations, but preferred to use the classical one to locate rooms.

Q10: Do you have any suggestion to improvements in the solutions?

The suggestions from this question are briefly presented here. These will then be further discussed in the next chapter.

The most suggested improvement that was made, was suggested by 9 of the participants and was to only display block names consisting of the prefix of room names at a low zoom level. Which means that since ”Realfagbygget” consists of blocks with different prefixes, it could initially display an *A* above the block that contains rooms that start with A, a *B* above the block that has rooms that start with B, and so on. In addition, three of the test subjects suggest that these blocks could have more differences, for instance, a color around each of the blocks, or different fonts or color on the names.

The second most suggested improvement was to change the hallway color, which 6 users suggested. 3 of those suggestions was to make the color more different from the rooms to make them more visible, while 2 of the users just did not like the gray color because they initially associated it with an area that was not accessible. The last person wanted the hallways to be changed back to white because that is what he is used to from other inside maps.

Another commonly suggested improvement was to make merged rooms different from unmerged rooms. This was suggested by 4 of the participants, where one of them suggested it because it can make the users be able to differentiate between a room that can be split further and a room that is unmerged. The 3 other test subjects suggested to show the walls inside merged rooms, but with a thinner line or less noticeable color than the other walls.

Another suggestion that was made by 4 of the participants was to show three digits in the last part of a merged room name. For instance, instead of displaying ”B173-77” it should display ”B173-177”.

In addition, 2 of the test subject suggested adding a feature to zoom on name click. Which means that if the user clicks on a merged name (or potentially group name) the solution should zoom in on the rooms that the name belongs to.

One of the participants felt that the algorithm for making merged room names was too naive. He would prefer if instead of just being a range from the lowest number to the highest, the names only included number they contained. For instance, if a group consisted of B173, B174 and B176 he meant that the name should not be B173-176 since B175 is not in the group. Instead, he suggested that the name should be something like B173/174/176 or B173-174,176.

In addition, some adjustments were suggested. For instance, 4 of the participants mentioned that they felt like it was displayed to few names at the lowest zoom level with names, and one said that the stairs should be displayed at lower zoom levels as well.

Q11: Do you have any other comments about the solutions?

The answers to the last question were normally either a summary of what they already had said or something they forgot to answer to one of the other questions. Which means that the information we got from this question is already mentioned.

Questions	Answers
Q1	18 noticed merged room names, 5 noticed merged rooms, 3 noticed hallway color, 2 didn’t notice any differences
Q2	5 noticed, 15 didn’t notice
Q3	20 didn’t notice
Q4	15 understood, 5 didn’t understand
Q5	15 said yes, 3 said it didn’t matter/unsure, 1 said it was a disadvantage, 1 didn’t answer
Q6	11 said yes, 3 said that something helped, 6 didn’t answer
Q7	19 said yes, 1 said no
Q8	14 said no, 4 said it was initially confusing, 2 said yes
Q9	19 preferred new, 1 preferred classical

Table 7.4: Results from questions listed in Section 6.2

7.2.3 Research Questions Revisited

The results of the experiment were in general very positive. 15 people understood the room names very quickly and 19 participants meant they were helpful when looking for rooms, which indicates a clear positive answer to RQ1. Most people did not notice the merging of rooms and no people noticed the merging of hallways, but after understanding the merging, 15 people thought room merging was helpful and 11 people thought merging of hallways was helpful. This indicates that the experiment has given us an overall positive answer to RQ2. Only 11 of 20 thought hallways were helpful, but of the 9 other, 3 said that something helped, but it was possible that it was just the color of the hallways, while the last 6 participants didn’t answer this question. This leans closer to a positive answer to RQ3 than a negative one, but it is not as decisive as RQ1 and RQ2. This shows that after they understood how the merging worked, the majority of the participants, meant it was helpful for the user experience. The result that 19/20 participants preferred our solution in favor over the classical solution, is a strong indication that our dynamic approach is very promising.

7.3 Summary

This chapter has explained the technical results and the results from the experiment. In the technical results, it was explained how well the solution performed, while the results of the experiment informed about the measured times for each task and the answers to the questions.

Chapter 8

Discussion

There are room for improvements based on the results presented in the previous chapter. Possible improvements will be discussed here. This chapter will be organized in the same fashion as the previous chapter, but it is instead separated into three parts: Discussion about the technical aspects, discussion about the experiment, and finally a general discussion about all the improvements suggested by the participants.

8.1 Technical Discussion

In this section, some technical problems and possible solutions are discussed.

8.1.1 More Robust Merging

The merging algorithm fails when the wrong points are chosen as corresponding points. Before the merging happens, we could perform an extra check and measure the distances between each corresponding point pairs. If one or more of these distances is longer than the distance threshold, we know some points are incorrectly connected, and we could trigger a special case handler that deals with a case like this.
One possible way this handler could work is to add additional points again, in the same manner, we do in the main algorithm. This might make sure that every close point will have a close corresponding point in the other polygon.

8.1.2 More Even Splitting

The splitting works fine most places, but as discussed before, there are scenarios where the splitting becomes uneven. Recall that our tree creation algorithm is a hybrid of BFS and DFS. This can potentially lead to wide and shallow trees if a room has a lot of neighbors.
A better solution would be to use pure DFS when creating the tree. Then the tree would always become equally deep or deeper than the current tree, and a split would be more even.

8.2 Experiment Discussion

This section discusses possible improvements with the experiment itself

8.2.1 Measured times

The resulting times from the tasks varied greatly. The standard deviation was very large, the largest being 53.19 seconds. Also, there was no detectable pattern in one of them compared to the other. For example, the mean was significantly better in the new solution when locating room ”E143”, while the mean in the classical solution was much better when locating room ”4.126”. This doesn't give us any useful information.

People are different and some of them took their time to search, while others were more quick and efficient. Also, it wasn't unusual that participants got lucky and stumbled upon their target room pretty quickly without doing any systematic search.
In the tasks where they had to draw paths, some people didn't worry about drawing rough paths and finished quickly, while others were more careful and made thorough checks to see if it was possible to go a certain place.

As a summary, the time of doing a task was very dependent on the participant, which lead to large variations and no visible pattern, so we cannot conclude anything from the quantitative data.

8.2.2 Execution of Experiment

The experiment went generally well. There were no mistakes with the quantitative tests, and the majority of the interviews gave us an answer to all the question.

However, we were inexperienced with conducting interviews, so it wasn't unusual that the questions were asked in different orders based on how the participant answered a question. Sometimes, participants didn't give a clear answer to a question and they would start talking about other aspects, and we didn't ask the same question again. This lead to some questions not having clear answers, which was our fault. If we had followed the question list in a more strict manner, and not moved on before we knew we had a proper answer, this problem could be avoided.

Another weakness with our interviews was that not all of them were recorded. We realized later, during analysis, that the recordings were useful for parts where people gave ambiguous answers, so we could listen to them again and try to understand what they really meant. We could have taken our time to find a private group room to conduct the interviews with recording, instead of interviewing in an open environment.

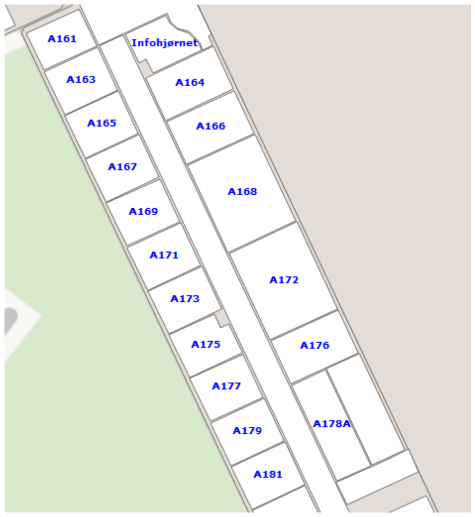
A third weakness, albeit not as important as the other two mentioned, was the way we described the tasks to the participants. We had no written template, so our explanations varied a little, and we also forgot to mention some factors now and then, while they were doing a task. One example was one participant who was contemplating if his target room was on a different floor because we forgot to mention that we were only testing one floor.

8.3 Suggested Improvements

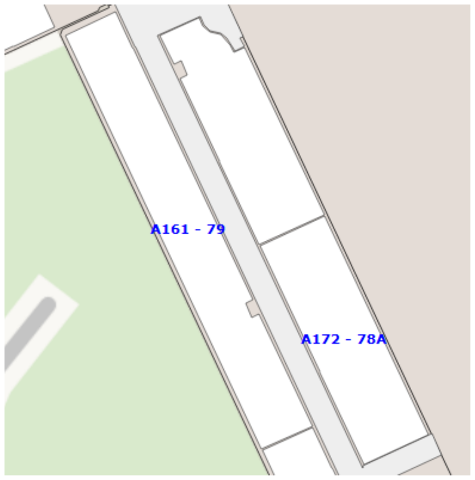
This section is discussing the improvements suggested by the participants

8.3.1 Name Range Representation

As mentioned earlier, some test subjects meant that the name representation could be improved, by not removing digits from the last name. They were confused and thought that, for example, 170 - 84 meant room 170 all the way down to room 84, instead of room 170 to room 184. We chose to represent it like this, to shorten the length of the marker, which especially had a visible effect with longer room names, say "A1-164 - 68", instead of "A1-164 - A1-168".
However, their main complaint was about the digits and not the letters. That is, they seemed to understand that B179 - 181 meant that the last name was B181 and not 181. So the payoff by keeping that extra digit, while creating less confusion, might be greater.



(a) Rooms ordered in odd/even convention



(b) Both groups merged together

Figure 8.1: Example of ordering conventions

In the tests, rooms with a dash (-), got stripped such that the dash was removed, to avoid confusion with the dash we used as the range operator. However, in real life, we can’t do this modification. This might be a weakness with our test since we don’t know if multiple dashes would cause additional confusion. But other possibilities instead of using a dash could be a double dash ”–”, an arrow ”$\rightarrow$”, a double arrow ”$\leftrightarrow$”, or others.

Another issue that arises with room name ranges, is ordering conventions. Different ordering conventions of house numbers are presented in (Rose-Redwood, 2008). These are for outdoor planning, but the same rules can be applied to indoor room ordering. An example is shown in Figure 8.1. For instance, it is not obvious which of the two groups room A176 is in before the user zooms in. We haven’t tested our application with any specific convention, but we think it is important to be aware that people might be used to different conventions.

8.3.2 Distribution of Merged Room Names

Another complaint was about the distribution of merged room names. Some of the participants mentioned that they would like that more names are shown at the higher levels. This is partly due to us choosing a specific margin between names and the removal of groups containing one or more rooms with a completely different name compared to the rest of the rooms. Figure 8.1 shows a group containing ”Infohjørnet”, ”A164”, ”A166” and ”A168”. The first name is drastically different from the others, so ”A164 - Infohjørnet” would be of no help to the user, so this name is not shown. Because names like this are removed, it can leave noticeable gaps in the distribution of names.

One way to avoid this problem could be to never merge any rooms that have characteristic names like this. No merged names would be removed, so the distribution would be as even as possible. It can possibly lead to more uneven merging, if rooms like this appear in the middle of larger groups, preventing the whole group of being merged together, and much smaller groups are formed instead.

Additionally, experimenting with the margin for each zoom level can also give greater benefits, and is much easier to modify.

8.3.3 Difference Between Merged and Unmerged Rooms

There were some people who had trouble understanding if a room was merged or not. They raise an interesting point, because we expected the names to reveal if a room was merged or not, based on the presence of the range operator (-). However, this was not clear for everyone.
We believe that using thin lines inside merged rooms, as some suggested, might be a decision that defeats the purpose of merging rooms in the first place since it would be like showing all the original rooms again. To make it beneficial the lines would have to draw as little attention as possible, while still being noticeable for the user if he’s studying a specific merged room.
Another potential way of giving a hint is to use a different range operator, like the ones presented in Section 8.3.1. An operator like this stands out more than a normal dash, so it could be helpful for making the same distinction.

8.3.4 Hallway Color

The results from the user tests give split opinions about the hallway color. The participants that complained about the color being too similar to the background had a point since it is important that people can easily see the difference. But at the same time, the color shouldn't be too strong either since that would draw all of the attention toward the hallways.
Based on all this, it seems like changing the color to another subtle value, for example, light yellow, that wouldn't draw any more attention than the gray color but would give a clearer contrast to the background, could be a generally more accepted solution.

8.3.5 Block Names

Almost half of the participants suggested that we should give a more clear separation between building areas. The building in the test consist of blocks, where each block is marked by a letter, so it made sense when we thought about it. But we didn't consider this when we chose to use this building. Figure 8.2 shows how the building is organized. The suggestion proposed by the participants was to mark the building in the same way on the lowest zoom levels.

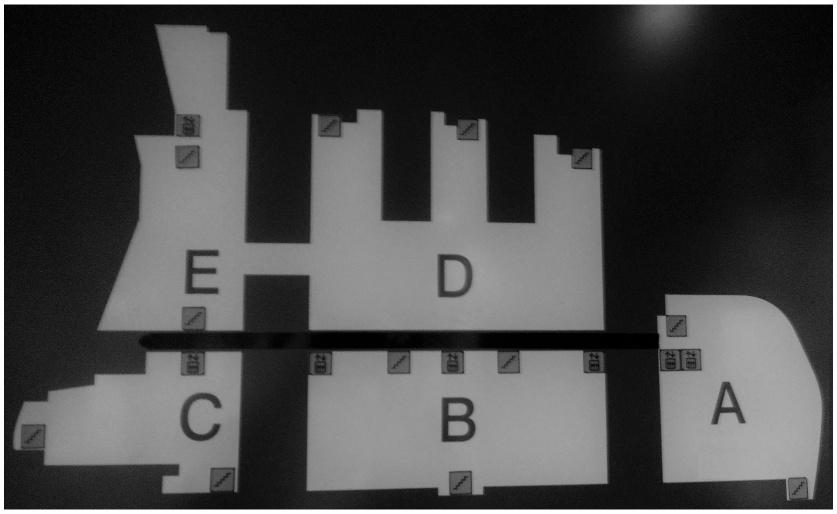


Figure 8.2: A map showing the grouping of the different room names

Figure 8.3 shows how block names could be shown to guide the user to where to look. It is very appropriate for the largest building at the bottom. But for the blocks on the top, it is only confusing. It just so happens that almost all of these building have rooms that start with ”1”, which is of no help to the user. Hence, the idea of block names is good if buildings are well organized, but if there is no clear organization of building blocks, it would only cause additional confusion.

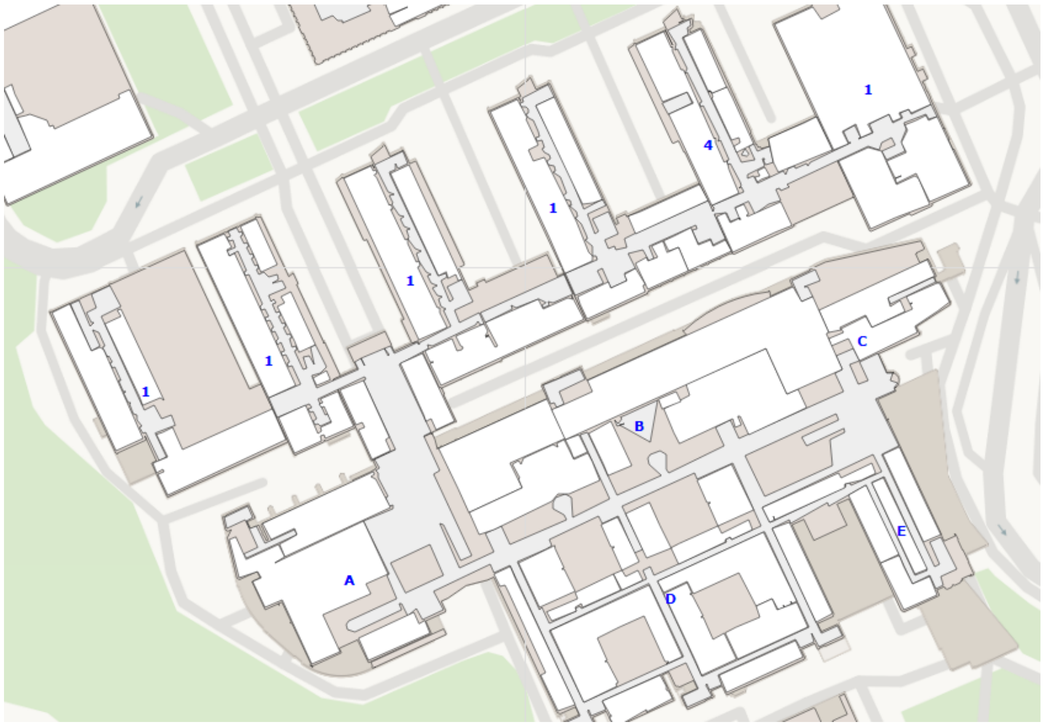


Figure 8.3: Example of displaying block names of buildings, screenshot from `http://folk.ntnu.no/morteano/Blocknames/Map.html`

An alternative approach is to give building blocks the name of the actual building which is already done by MazeMap, as shown in Figure 8.4. On a higher zoom level, this name could be replaced by different building blocks. If we managed to do this only when a building was organized like ”Realfagbygget” while still keeping the building name for buildings without sub-blocks like ”Kjemiblokk”, we could get the best of both worlds.

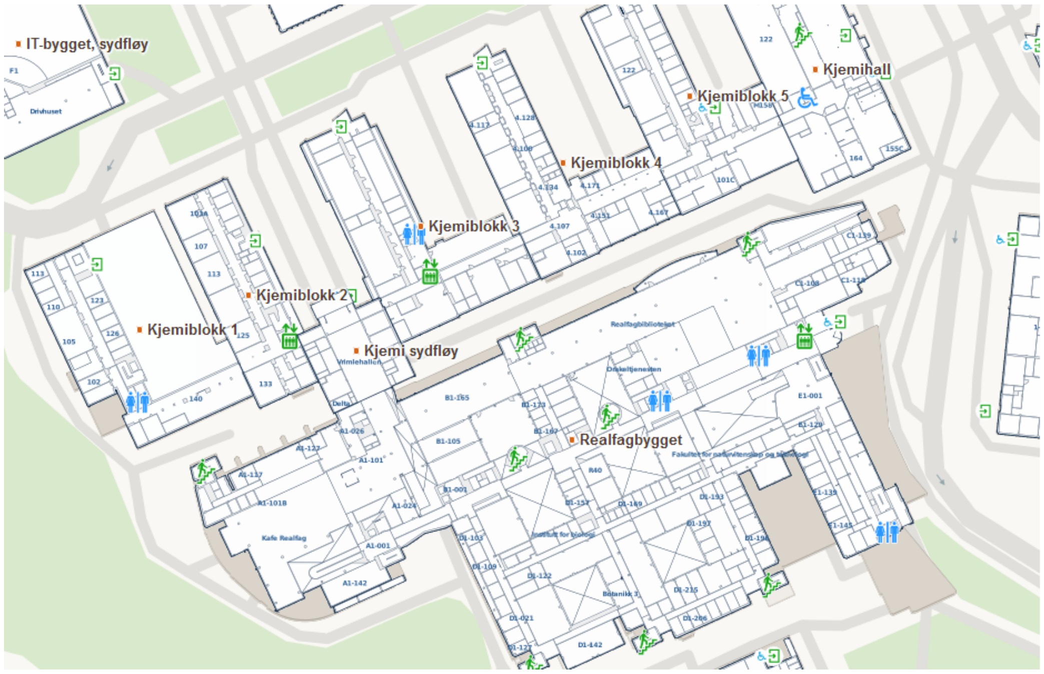


Figure 8.4: MazeMap showing names of buildings (”Realfagbygget”, ”Kjemiblokk 1”, etc.

8.4 Summary

In this chapter, potential improvements have been discussed, both to the implementation and to experiment. The discussion tells that there are a lot of potential improvements for the solution, but many of them are based on subjective opinions, so there are no right or wrong answers. But further testing with these improvements would be beneficial.

Chapter 9

Evaluation, Conclusion and Future Work

This chapter concludes the work that is done according to the research questions and suggests possibilities for future work and how it could be evaluated.

9.1 Evaluation

Wohlin et al. (2003) mention four categories of validity concerns for experiments, which is further explained in Wohlin et al. (2012). These will be discussed here to assess how valid our results are. The first validity concern is the internal validity, which is concerned with factors that may affect the dependent variables without the researcher's knowledge. Based on this it was decided that half of the participants first performed tasks in the classical solution, then performed tasks in the new solution, while the other half first used the new solution, then the classical one. In addition, to make sure that we got the test subjects opinions as correct as possible, it was decided to use an interview instead of a questionnaire. Which means that the test subjects could talk freely instead of only choosing one of the suggested answers.

The second validity concern is the external validity, which is related to the ability to generalize the results of the experiments. This was considered when the tests were created and when the users were selected. The tests consisted of the general tasks that normally are done in maps, and the participants in the experiment were either potential users of the solution or expert users.

The next validity concern is conclusion validity, which is concerned with the possibility to draw correct conclusions regarding the relationship between treatments and the outcome of an experiment. One of the most common threats to conclusion validity is low statistical power, which means that the sample size is too small. To avoid this, a sample size of 20 participants was selected, though it would most likely be beneficial to test on more people.

The last validity concern is construct validity, which is related to the relationship between the concepts and theories behind the experiment and what is measured and affected. This has also been considered, and one of the threat could possibly have had an effect, which is hypothesis guessing. This is when since the participants knows, or guesses, the desired end-result it may affect their actions. Therefore, since some of the users probably understood that the experiment tried to test if merging could be beneficial, it is possible that they would be more positive to the solution that uses merging.

9.1.1 Sample Size

Green and Thorogood (2013), page 122, state: ”If addressing a fairly specific question, the experience of most qualitative researchers is that in interview studies, little ’new’ comes out of transcripts after you have analysed 15 or so with a relatively homogeneous group of participants.” Since there were no notable differences between the answers from the experts compared to the casual users, we can with great confidence state that we had a relatively homogeneous group of 20 participants.

However, Mason (2010) which analyzes 560 qualitative studies used in PhD theses, states the mean sample size is 31, which is over 50% more than our sample size. It also argues that interviewing skills is an important factor, and states that a skilled interviewer conducting 10 interviews might get more informative results than a novice interviewer conducting 50 interviews.

Since this was our first professional interview experiment, and there were flaws in the interviews conducted, we think it will be necessary to conduct more interviews before we can say with certainty that these indications are correct.

9.2 Conclusion

In this project, a hypothesis and a number of research questions have been defined based on motivations raised from literature regarding the level of detail on maps. The research questions specifically regard the context of indoor maps, a field that has not been under much research. The results of the experiment show that of the 20 participants, 12 meant that room merging was helpful, 7 said that it didn't matter or that they were unsure if it was helpful, while 1 said it was a disadvantage. When they were asked if they felt that the name merging was helpful, 19 of the participants said that is was, while 1 said it was not helpful at that point, but that it had potential to be helpful. Therefore, even though the experiment only had 20 participants, it indicates that both merging of rooms and merging of names is helpful for the user. RQ1 received a clear positive answer, RQ2 received a generaly positive answer, while RQ3 didn't get a clear positive or negative answer, but it leaned more towards positive than negative. These results strengthen our hypothesis that a more dynamic level of detail based on zoom level, will contribute to a more efficient navigation and improved user experience for indoors maps.

9.3 Future Work

This section discusses future work and possible future improvements.

9.3.1 Testing Additional Floors

Our solution has only been tested on 46 different floors. There exist a lot more floors which can potentially contain more special cases that make our merging algorithm fail. We already know now that it fails in some rare cases, and these cases can happen in other floors. Also, new cases we don't know about might show up. Hence, testing on additional floors would be necessary to further test the robustness of the merging algorithm.

9.3.2 Additional and Improved User Tests

Even though we got positive results from the experiment, we only interviewed twenty people. Based on the discussion, we need to perform additional tests to be able to draw a conclusion. Since we have some experience from the first experiment, we have an idea of what we can improve in future experiments:

- Write down the explanations for the tasks, to make sure we don't forget to mention anything
- Record all interviews
- Ask questions in a strict order and not move on until a question is properly answered

9.3.3 Summary

As a summary, the empirical results give strong hints that our solution is promising, so there's no reason to stop at this point. We have laid the foundation for a field of research that has not been under much focus before. If the technical issues could be improved to make the implementation more robust, and more extensive empirical testing could be performed, this approach might be beneficial to integrate into existing commercial indoor map solutions.

Bibliography

Vladimir Agafonkin. Douglas peucker algorithm implementation. `http://mourner.github.io/simplify-js/`, 2015.

Karl-Heinrich Anders. Level of detail generation of 3d building groups by aggregation and typification. In *International Cartographic Conference*, volume 2, 2005.

Jacques Bertin. Semiology of graphics: diagrams, networks, maps. *Madison, WI: The University of Wisconsin Press, Ltd*, 1983.

Sayantan Biswas. Level of detail image. `http://www.sayantanbiswas.com/works/chn_Tower/LOD3.jpg`, 2009.

ROGER J Brooks and ANDREW M Tobias. Choosing the best model: Level of detail, complexity, and model performance. *Mathematical and computer modelling*, 24(4):1–14, 1996.

Per Christensson. Rendering definition. `https://techterms.com/definition/rendering`, 2016.

Andy Cockburn, Amy Karlson, and Benjamin B Bederson. A review of overview+ detail, zooming, and focus+ context interfaces. *ACM Computing Surveys (CSUR)*, 41(1):2, 2009.

Adler CG Da Silva and Shin-Ting Wu. Consistent handling of linear features in polyline simplification. In *Advances in Geoinformatics*, pages 1–17. Springer, 2007.

David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

Simone Garlandini and Sara Irina Fabrikant. Evaluating the effectiveness and efficiency of visual variables for geographic information visualization. In *International Conference on Spatial Information Theory*, pages 195–211. Springer, 2009.

Judith Green and Nicki Thorogood. *Qualitative methods for health research*. Sage, 2013.

Jan-Henrik Haunert and Alexander Wolff. Optimal simplification of building ground plans. In *Proceedings of XXIst ISPRS Congress Beijing*, pages 372–378, 2008.

Paul S Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. Technical report, DTIC Document, 1997.

Richard Holloway. Viper: A quasi-real-time virtual-environment application. Technical report, Technical Report TR92-004, University of North Carolina at Chapel Hill, 1992.

Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual analytics: Definition, process, and challenges. In *Information visualization*, pages 154–175. Springer, 2008.

Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. Indoor positioning using gps revisited. In *International conference on pervasive computing*, pages 38–56. Springer, 2010.

Richard E Korf. A new algorithm for optimal bin packing. In *AAAI/IAAI*, pages 731–736, 2002.

Peter Lindstrom, David Koller, William Ribarsky, Larry F Hodges, Nick Faust, and Gregory A Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM, 1996.

Alexandra Lorenz, Cornelia Thierbach, Nina Baur, and Thomas H Kolbe. Map design aspects, route complexity, or social background? factors influencing user satisfaction with indoor navigation maps. *Cartography and Geographic Information Science*, 40(3):201–209, 2013.

Mark Mason. Sample size and saturation in phd studies using qualitative interviews. In *Forum qualitative Sozialforschung/Forum: qualitative social research*, volume 11, 2010.

George Mouratidis. Context dependent multimodal routing in indoor/outdoor environments based on indoorgml and openstreetmap. 2015.

Alexander Salveson Nossum. Literature survey ba8204.

Alexander Salveson Nossum. Indoortubes a novel design for indoor maps. *Cartography and Geographic Information Science*, 38(2):192–200, 2011.

Alexander Salveson Nossum. Exploring new visualization methods for multistorey indoor environments and dynamic spatial phenomena. 2013.

Justin O'Beirne. What happened to google maps? `http://www.justinobeirne.com/essay/what-happened-to-google-maps`, 2016.

Wilfried M Osberger and Ann M Rohaly. Automatic detection of regions of interest in complex video sequences. In *Photonics West 2001-Electronic Imaging*, pages 361–372. International Society for Optics and Photonics, 2001.

Sergio Solano Pinedo, Marco Moreno Ibarra, and Miguel Torres Ruiz. Genetic algorithms for map generalization. 2013.

Paulo Pombinho, Maria Beatriz Carmo, and Ana Paula Afonso. Adaptive mobile visualization–the chameleon framework. *Computer Science and Information Systems*, 12(2):445–464, 2015.

Arto Puikkonen, Ari-Heikki Sarjanoja, Merja Haveri, Jussi Huhtala, and Jonna Häkkilä. Towards designing better maps for indoor navigation: experiences from a case study. In *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*, page 16. ACM, 2009.

Verena Radoczky. How to design a pedestrian navigation system for indoor and outdoor environments. In *Location based services and telecartography*, pages 301–316. Springer, 2007.

Martin Reddy. Specification and evaluation of level of detail selection criteria. *Virtual Reality*, 3(2):132–143, 1998.

Nicolas Regnauld. Contextual building typification in automated map generalization. *Algorithmica*, 30(2):312–333, 2001.

José Ribelles, Angeles López, and Oscar Belmonte. An improved discrete level of detail model through an incremental representation. In *TPCG*, pages 59–66, 2010.

Reuben S Rose-Redwood. Indexing the great ledger of the community: urban house numbering, city directories, and the production of spatial legibility. *Journal of Historical Geography*, 34(2):286–310, 2008.

Alan Saalfeld. Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1): 7–18, 1999.

Henrik Sandström and Johannes Svensson. *Indoor Maps*. Skolan för datavetenskap och kommunikation, Kungliga Tekniska högskolan, 2010.

M Sester, P van Oosterom, F van Harmelen, and S Mustière. Generalization of spatial information. *Dagstuhl seminar 09161*, 2009.

K. Stuart Shea. *Cartographic Generalization*. NOAA TECHNICAL PUBLICATIONS, 1988.

K Stuart Shea and Robert B McMaster. Cartographic generalization in a digital environment: When and how to generalize. In *Proceedings of AutoCarto*, volume 9, pages 56–67, 1989.

James J Thomas and Kristin A Cook. A visual analytics agenda. *IEEE computer graphics and applications*, 26(1):10–13, 2006.

Anne M Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive psychology*, 12(1):97–136, 1980.

Peter Van Oosterom. The gap-tree, an approach to 'on-the-fly'map generalization of an area partitioning. *GIS and Generalization, Methodology and Practice*, pages 120–132, 1995.

J Mark Ware, Christopher B Jones, and Geraint Ll Bundy. A triangulated spatial model for cartographic generalisation of areal objects. In *International Conference on Spatial Information Theory*, pages 173–192. Springer, 1995.

Torbjörn Wigren. Clustering and polygon merging algorithms for fingerprinting positioning in lte. In *Signal Processing and Communication Systems (ICSPCS), 2011 5th International Conference on*, pages 1–10. IEEE, 2011.

Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer, 2003.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

Jeremy M Wolfe and Todd S Horowitz. What attributes guide the deployment of visual attention and how do they do it? *Nature reviews neuroscience*, 5(6): 495–501, 2004.

Shin-Ting Wu, Adler CG da Silva, and Mercedes RG Márquez. The douglas-peucker algorithm: Sufficiency conditions for non-self-intersections. *Journal of the Brazilian Computer Society*, 9(3):67–84, 2004.

Borut Žalik. An envelope construction of a set of polygons in a land cadastre. *Computers & geosciences*, 29(7):929–935, 2003.

Xiaofang Zhou, David Truffet, and Jiawei Han. Efficient polygon amalgamation methods for spatial olap and spatial data mining. In *International Symposium on Spatial Databases*, pages 167–187. Springer, 1999.