Case study onderzoek naar de inzetbaarheid van parametrische ontwerpstrategieën: detentiehuizen

Samuel Cuvelier

Promotor: prof. dr. Ronald De Meyer
Begeleider: ir.-arch. Ruben Verstraeten, ir. Tiemen Strobbe

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: architectuur

Vakgroep Architectuur en Stedenbouw
Voorzitter: prof. dr. Pieter Uyttenhove
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2012-2013

UNIVERSITEIT GENT

Case study onderzoek naar de inzetbaarheid van parametrische ontwerpstrategieën: detentiehuizen

Samuel Cuvelier

Promotor: prof. dr. Ronald De Meyer
Begeleider: ir.-arch. Ruben Verstraeten, ir. Tiemen Strobbe

Masterproef ingediend tot het behalen van de academische graad van
Master in de ingenieurswetenschappen: architectuur

Vakgroep Architectuur en Stedenbouw
Voorzitter: prof. dr. Pieter Uyttenhove
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2012-2013

UNIVERSITEIT GENT

Voorwoord

Eerst en vooral wil ik een aantal mensen bedanken dankzij wie dit werk tot stand is kunnen komen:

Promotor prof. dr. Ronald De Meyer; ten eerst om mij de mogelijkheid te bieden om dit onderzoek te voeren en ten tweede om steeds met kritische blik het onderzoek vooruit te helpen.

begeleiders Tiemen Strobbe en Ruben Verstraeten om mij enerzijds intellectueel bij te staan voor het begeleiden van het onderzoek en anderzijds mij ook met hun technische kennis te helpen bij de praktische uitwerking ervan. In het bijzonder wil Tiemen hier ook bedanken om mij bij te staan bij het appliceren voor de Smart Geometry Workshop in London, wat een hele leerzame en leuke ervaring was.

mijn familie, dankzij hun financiële en mentale steun zijn de afgelopen jaren goed kunnen verlopen met dit werk als eindresultaat.

alle vrienden, die steeds voor de nodige afleiding zorgden om naast de studies ook eens te kunnen ontspannen. In het bijzonder wil ik Glenn bedanken voor de ontspanning en afleiding maar ook voor het vele luisteren en het bijstaan in moeilijke momenten. Zonder hem zou ik het nooit zo ver gebracht hebben.

Samuel Cuvelier, 12/08/2013

Case study onderzoek naar de inzetbaarheid van parametrische ontwerpstrategieën: detentiehuizen

door

SAMUEL CUVELIER

Masterproef ingediend tot het behalen van de academische graad van

MASTER IN DE INGENIEURSWETENSCHAPPEN: ARCHITECTUUR

Academiejaar 2012-2013

Promotor: prof. dr. RONALD DE MEYER

Begeleider: dr. ir.-arch. RUBEN VERSTRAETEN, ir.-arch. TIEMEN STROBBE

Faculteit Ingenieurswetenschappen en Architectuur

Universiteit Gent

Vakgroep Architectuur en Stedenbouw

Voorzitter: prof. dr. PIETER UYTTENHOVE

Samenvatting

Deze masterproef onderzoekt de inzetbaarheid van parametrische ontwerpstrategieën in een vroeg stadium van het ontwerpproces. In het bijzonder wordt de invloed van de relaties tussen ruimtes onderzocht, zo onafhankelijk mogelijk van geometrie. Hiervoor werd een bestaande case van een ontwerp van een detentiehuis als vertrekpunt genomen. Als eindpunt van het onderzoek werd een prototype van een applicatie ontwikkeld die de ontwerper zal bijstaan bij het ontwerpen, met het aantal ruimtes en de relatievoorwaarden als input.

Trefwoorden

Detentiehuizen, generatief ontwerp, floorplanning, nabijheidmatrix, applicatie (C# - java)

Case study: applicability of parametric design strategies: detention houses

Samuel Cuvelier

Supervisor: Ronald De Meyer

***Abstract*: This master dissertation investigates the applicability of parametric design strategies in an early stage in the design process. Specifically the influence of relationships between spaces is a starting point of the research. This will be investigated as independently as possible from any geometrical boundaries. For this research a detention house is the case study. Finally an application has been developed that will aid the designer in the early stage of the design process, with input values: the number of spaces and the relation constraints between spaces.**

***Keywords:* detention houses, generative design, floor planning, adjacency matrix, application (C# - java)**

I. Introduction

After many years of Computer Aided Design (CAD) it is still remarkable that in architecture practices computers are mainly used as tools for drawing and text mockups **Fout! Verwijzingsbron niet gevonden.**, also more and more for visualizations. Still there are advanced systems that can evaluate certain aspects of a building like a Building Information Model (BIM). Research has shown that these models are mainly used in a later stage, after all the important design decisions have been made **Fout! Verwijzingsbron niet gevonden.**. The role of the computer must be redefined to a knowledge-based design assistant. And it must be well identified how and where in the design process de computer can be used in the most useful way **Fout! Verwijzingsbron niet gevonden.**.

The goal of this master dissertation is to investigate how to use the computer in an early stage in the design process. The research is based on known research of layout problems and wants to be a practical expansion with the final goal of developing an application to help de designer.

The idea of the application is to generate a generic model that can be applied to a variable context with variable conditions and constraints specific to the user.

II. Evaluation

This chapter describes the more traditional way how to use the computer in a more intelligent way. It will start from the existing case of the detention house to analyze this model and investigate which evaluative data can be extracted from the model.

A. Building Information Model (BIM)

BIM is a digital representation of physical and functional characteristics of a facility. It's a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life cycle; defined as existing from earliest concept to demolition **Fout! Verwijzingsbron niet gevonden.**.

For this research the BIM of the detention house was modeled to extract the gb.XML file which allows the user to export all the information of the model to a text file that can then be used for evaluation.

B. Evaluation

The model is evaluated to generate the adjacency matrix. This matrix contains all the adjacency relationships between spaces and is the main input for constraints of the application developed in the last chapter. The adjacency matrix can be better visualized by a graph (figure 1). The graph is a tool widely used by designers as a first idea to visualize the relations between spaces.

Other possibilities with this matrix are to evaluate the energy performance of the building by adding more information or attributes to the objects in the model, like wall thickness. If the outside spaces are also modeled; the matrix can evaluate which spaces are adjacent to this outside space.

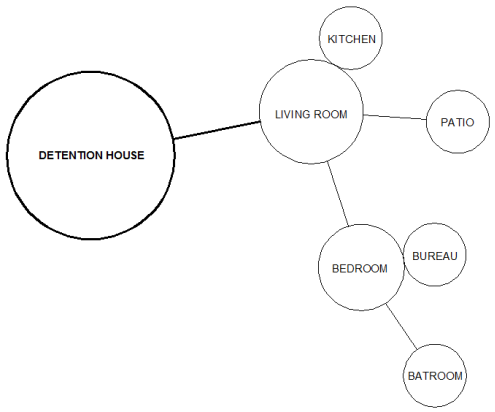


Figure 1 - Graph of an individual unit of the detention house

III. Design

A. Layout Problem

Spatial configurations and layout problems are very common in the architectural design. They are among the most complex problems, since a design task cannot be described as a Well Structured Problem (WSP) which is necessary to be used as input for a computer. Therefore the design task should be simplified into a subtask and the result of that subtask must be well interpreted to be feedback for the entire design.

In spite of the long research history associated with automated layout and space allocation systems, in practice these systems have not been utilized to their full potential **Fout! Verwijzingsbron niet gevonden.**.

B. *Models*

The research to develop a model for the case has been an evolution of different models starting with a first model based on the principles of the shape grammar **Fout! Verwijzingsbron niet gevonden.**. The goal was to try and build a model that created the detention house in different contexts. The result was unpleasing because of the great complexity of the model. Not all creative design decisions can be put into a parametric model. Therefore in the further models only a small part of the detention house (the individual unit) will be taken into account as a representative unit for the whole.

The goal of the further models was to create all possible configurations of a simple cube that can be sliced in any possible way according to what the user wants. On top of these possible configurations, all spaces created can be altered between each other (i.e. permutation) to get all possibilities.

The result was a user dependent model that gave all possibilities. The model was created in grasshopper3d because this program gives a large possibility on parametric design and geometrical representation.

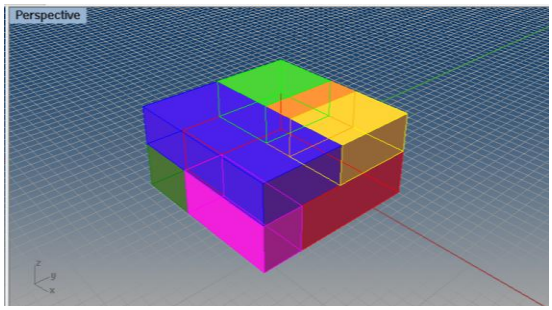


Figure 2 - 1 out of 362880 possibilities

The disadvantage of this model was that it was a geometrically dependent model and the goal is to generate only those models that comply with the given adjacency relations. So next to the model there should be an adjacency matrix to map the model in numbers rather than in geometry.

Another disadvantage of the geometry was that it made the model run very slowly.

Because of the previous reasons the goal was now to create these models purely based on a textual representation of the model. Therefore geometrical constraints were excluded.

IV. String Representation

The string representation is based on the floor planning principles that are also used in electronic chip design **Fout! Verwijzingsbron niet gevonden.**. The plans are created by cutting a rectangular space several times. Each configuration can be represented by a slicing tree that, in turn, can be represented by a Polish expression.

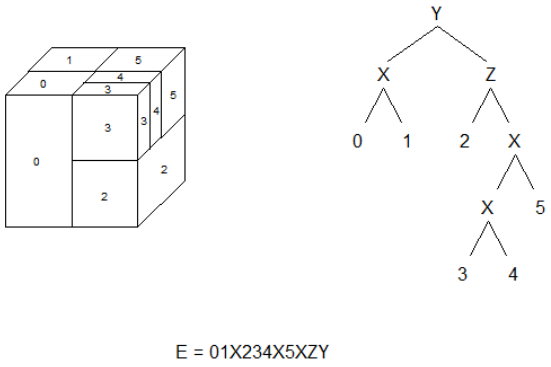


Figure 3 – 3-dimensional example of the representations

The Polish expression (E) consists of the space numbers and the operators (2D: V and H, 3D: X, Y and Z) that are used to divide the spaces. This expression is the starting point. The application generates all possible Polish expressions with a given number of spaces.

A. *Adjacency matrix*

To be able to compare the adjacencies with the given input constraints of adjacencies, it is necessary to create an adjacency matrix from this Polish expression. This is done by first creating a topological matrix that maps all spaces in a matrix that can also be used for a geometrical representation.

This topological matrix can then by evaluated to see which spaces are adjacent to create the adjacency matrix.

V. Application

The application was written in the free software processing, since this allows for creating a better graphical user interface than what grasshopper3d can do. The screen is divided into 2 parts: left the part where the input values can be given; right: the part with the solutions. The solutions are all the possible configurations with those configurations that meet the constraints printed in black.

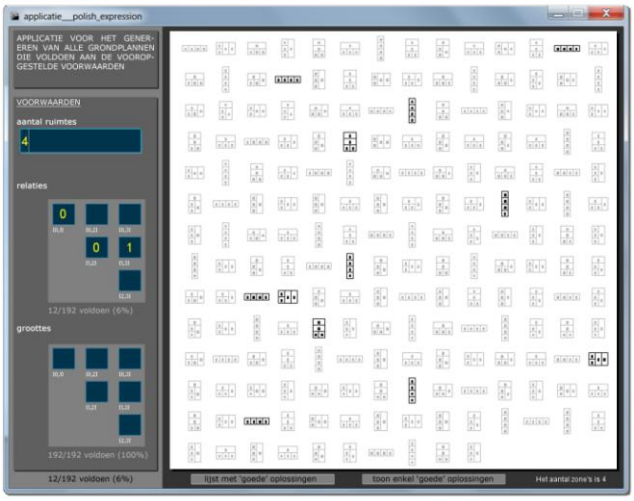


Figure 4 - Application (4 spaces, 3 constraints)

It's a user-friendly application that generates the solution in real-time with every single adjustment of the constraints so that the user can see how the application works to understand the underlying algorithms.

A. *Results*

There are four different kinds of constraints possible. The two most import ones in this research are de relational constraints between spaces: two spaces lie adjacent to one another (0 = 1) or they do not (0 X 1). The two other constraints are 'the greater than' and 'the smaller than' constraints. These last two are added to the application as an example of the way the application can be expanded.

The chart shows the number of possible configurations remaining after the input of a constraint related to the number of spaces on the X axis.

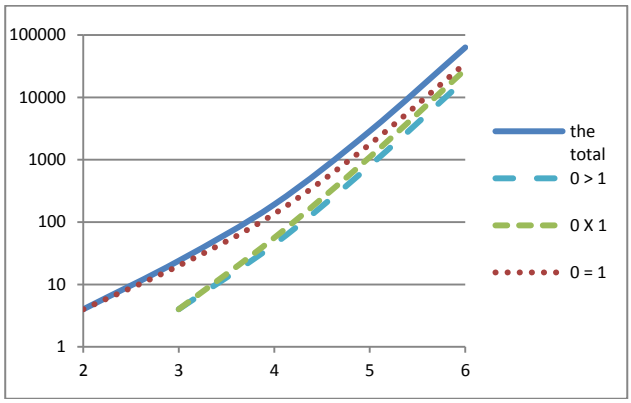


Chart 1 - Solution Space - influence of the basic constraints

As can be seen in the chart the 'being adjacent' of two spaces is the least strict constraint for a low number of spaces. When the number of spaces rises the two relational constraints will converge to each other. The spatial size constraints are more strict because the application is based on a non-geometrical Polish expression that only generates a minimum representation of the floor plan. Here the designer's intelligent interpretation is necessary to adjust the configuration to his own wishes.

B. *Test case*

As a test case an individual unit of the detention house was tested with the application. Given the constraints (right-hand side in figure 5) there were 148 possible solutions from a total of 63360 possible configurations with 6 spaces.

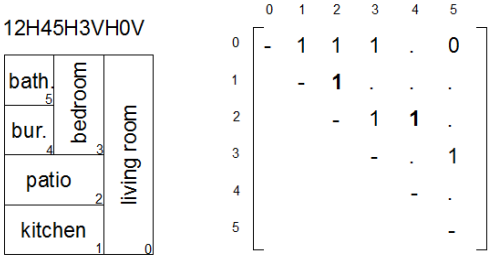


Figure 5 - Example output - plan and adjacency constraints

This is a final result of a playing process with the application. First, the user inputs only the most important constraints. This will generate a large solution space that can then gradually be reduced by inputting more and more secondary constraints. This way, the user sees how the application works and in this way the best result will be acquired.

It is important to note that this generated configuration for a floor plan should not be taken literarily, but should be intelligently fed back to the original design task. The application only gives a suggestion of a geometrical representation.

REFERENCES

[1] Bryan Lawson, *Towards a computer-aided architectural design process: a journey of several mirages*, Computers in Industry 35, pp. 47-57, 1998.

[2] S. Kirsh, *A practical generative design method*, Computer-Aided Design 43 (1), pp. 88-100, 2011.

[3] Y.A. Kalay, *Redefining the role of computers in architecture: from drafting/modelling tools to knowledge-based design assistant.*, Computer-Aided Design, pp. 319-328, 1986.

[4] National Building Information Model Standard Project Committee, *http://www.buildingsmartalliance.org/index.php/nbims/faq/*

[5] R.S. Ligett, *Automated facilities layout: past, present and future*, Automation in Construction, 9(2), pp. 197-215, 2000.

[6] José Pinto Duarte, *Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses*, Massachusetts Institute of Technology, 2001.

[7] Wang, Laung-Terng; Chang, Yao-Wen; Cheng, Kwang-Ting (Tim), *Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon)*, Morgan Kaufmann, 2009.

Inhoudsopgave

DEEL 1: ONDERZOEK

1. INLEIDING

1. Probleemstelling

Na vele jaren van evolutie in het Computer Aided Design (CAD) valt het nog steeds op dat computers in architectuurbureaus veelal gebruikt worden als hulpmiddel om te tekenen en als tekstverwerker (Lawson, 1998). Steeds meer ook om visualisaties te maken. Af en toe worden gevorderde systemen gebruikt om geïsoleerde problemen op te lossen; het gaat hier dan om evaluatietools die bepaalde aspecten voor gebouwprestaties simuleren zoals energieverbruik. Deze laatste toepassingen suggereren een belangrijk potentieel van de computer: namelijk dat we zeer accuraat kunnen weten hoe een gebouw zal zijn lang voordat het gebouwd zal worden. Onderzoek heeft wel aangetoond dat deze Building Information Models (BIM) voornamelijk in een later stadium gebruikt worden, reeds nadat de belangrijkste ontwerpmatige beslissingen genomen zijn (Krish, 2011).

Kalay stelt reeds in 1985 dat de rol die de computer speelt in het architecturale ontwerp opnieuw gedefinieerd moet worden van enkel teken- en modelleertool naar een “intelligent knowledge-based design assistant” (Kalay, 1985). Men moet goed identificeren waar in het ontwerpproces de computer nuttig kan worden ingezet en hoe hij daar gebruikt kan worden.

1.1. Nieuw soort bouwen

Het is belangrijk om de nieuwe evoluties binnen het beroep van architect te implementeren. Dit betekent concreet de nieuwe digitale technologieën te implementeren in de methode waarmee ontworpen wordt, vanaf het eerste conceptuele idee. Het idee van massaproductie moet worden herzien door de principes van 'customization', wat architectuur fundamenteel zal veranderen. Een volledig nieuwe esthetiek zal het resultaat zijn van digitale parametrische ontwerpprocessen die rechtstreeks gekoppeld kunnen worden aan de fabriek met CNC productiemethoden (Oosterhuis, 2012).

Gebouwcomponenten kunnen niet meer gezien worden als passieve objecten. Digitale technologieën vestigen zich in het ontwerpproces. Voorbeelden hiervan zijn: het parametrische ontwerpen, generatieve componenten en 'file to factory' productie. Gebaseerd op simpele regels worden de gebouwonderdelen herbenoemd met onderlinge 'bottom up' bidirectionele relaties.

Er zijn nu al vele jaren voorbij sinds de introductie van de computer en het internet, maar nog weinig is veranderd in de componenten van de gebouwde omgeving, noch in het veranderen van de manier waarop ontworpen wordt. Er zijn wel al programma's ontwikkeld om het gebouw te simuleren in bepaalde contexten zoals bij 'building information models' (BIM). Maar dit wordt vooral gebruikt om bestaande ontwerpen te verbeteren. Nochtans bevatten deze programma's veel meer informatie waarmee gewerkt kan worden. Dit is een gemiste kans en zelfs aan vele universiteiten wordt afgeraden om met de computer te werken voor het ontwerp. Het is natuurlijk niet eenvoudig als beginneling om van dag één in de architectuuropleiding met de computer te werken. Eerst moeten de basisvaardigheden aangeleerd worden, maar waarom zou dit ook niet kunnen met de computer. Het is dan het doel van de onderwijsinstellingen om de juiste programma's te selecteren die de ontwerpvrijheid niet in het gedrang brengen. Vaak zijn programma's zoals Revit voor BIM beperkend in de creatieve vrijheid, maar hier zijn goede alternatieven voor. De vernieuwende geest van Oosterhuis is hoopvol voor de toekomst: *"It is my explicit opinion that students in architectural faculties should play in the very design process with all kinds of new digital and social media, from day one."* (Oosterhuis, 2012)

1.2. Computer als ontwerpassistent

Het is verwonderlijk dat de eigenlijke eerste pogingen om de computer in te zetten als design assistent voorafgaan aan het kunnen tekenen met de computer. Toen, meer dan 40 jaar geleden, had niemand zich kunnen voorstellen dat de computer vandaag in elk architectuurbureau zo een dominante rol zou spelen als goedkoop toestel voor grafisch hoogstaande prestaties. Het is maar recent dat onderzoek zich weer op de designkant gaat

focussen. De vraag blijft vaak of het nuttig is de computer op deze manier in te zetten, aangezien applicaties zich vaak gaan focussen op 1 aspect van het gebouw; daarom is het voor verder onderzoek steeds aangewezen om het gebouw zo compleet mogelijk te beschouwen (Lawson, 1998). Vaak is het moeilijk al deze factoren te kunnen vatten, zoals bijvoorbeeld de kwaliteit van het rondlopen in een gebouw. Deze factor kan opgedeeld worden in bijvoorbeeld lichtinval, uitzichtafstand, breedte van gangen, klimaatcontrole enz. Noch de gebruiker, noch de architect kunnen aan deze factoren een precies gewicht van belangrijkheid hangen, dus kan er ook nog niet voorgesteld worden hoe een computer hiermee zou omgaan.

Het is vandaag logisch dat de computer gebruikt wordt in een architectenbureau. Computers leveren sterke grafische prestaties en in een de architectuurwereld is dit grafische zeer belangrijk als overtuigingsmechanisme voor klanten en voor precieze tekeningen. Weliswaar is deze evolutie ietwat een afleiding geweest om de computer te gebruiken als designassistent. Het onderzoek dat gepaard gaat met de grafische prestaties en dergelijke heeft eigenlijk weinig te maken met architectuur: het is wat de tekeningen bedoelen en hoe ze gebruikt worden wat in de praktijk van belang is (Bell, 1989). Het is dus goed dat de computer ingezet wordt als 'computer aided drafting – tool', maar het doel blijft om te focussen op meer dan enkel dit.

Een probleem is ook dat het ontwerpproces vaak slecht gedefinieerd is. Vaak kan de klant niet echt al zijn wensen in concrete gestructureerde doelstellingen omzetten, wat het dan moeilijk maakt voor de ontwerper om het ontwerpproces op te starten. Op deze manier is het ook moeilijk om de computer rechtstreeks daarbij in te zetten als de doelstellingen niet geconcretiseerd kunnen worden. Vaak begint het ontwerpproces al voordat alles goed begrepen is. Het is ook algemeen aanvaard dat bij elk ontwerpproces een oplossing steeds gepaard gaat met een nieuw probleem of een nieuwe uitdaging.

Ontwerpprocessen kunnen ook moeilijk onderworpen worden aan optimalisatieschema's of andere methoden. Het is door de evolutie in het ontwerpproces en door ervaring en kennis dat de ontwerper verder geraakt. Studies stellen dat het ontwerpproces integratief van karakter is en dat dus één specifiek element niet één specifiek probleem kan oplossen (Lawson, 1998). Het is het samenspel van elementen en indien er daar één in wordt gewijzigd zal de volledige situatie van het ontwerp veranderen. Daarom zal een reeks computerprogramma's die het ontwerp van een gebouw gaan afwegen tegen een aantal criteria niet het verhoopte resultaat geven.

Men moet zich er dus goed van bewust zijn dat de computer wel gebruikt kan worden als assistent voor ontwerp, maar waarbij er specifieke informatie wordt ingeladen. De andere informatie zal genegeerd worden om zo een bevredigend resultaat te krijgen op dat specifieke niveau. Het is een doel om vervolgens al deze specifieke computertools samen te convergeren om zo ver mogelijk te geraken in het ontwerpen.

2. Doelstelling

Het doel van deze masterproef is op basis van de case van het detentiehuis te onderzoeken op welke manier het mogelijk is om de computer nuttig in te zetten al van bij het begin van het ontwerpproces. Dit onderzoek baseert zich op reeds bestaande studies over 'layout problems' en wil hierop een praktische uitbreiding zijn met als einddoel het ontwikkelen van een applicatie die de ontwerper van bij het begin van het ontwerpproces kan helpen.

Het idee achter de applicatie is om een generisch model te ontwikkelen van de case die verder in dit hoofdstuk toegelicht zal worden. Generisch zodat je hetzelfde onderliggende ontwerpschema kan plaatsen op een variabele site, in een variabele context en met variabele randvoorwaarden specifiek voor de gebruikers.

De applicatie wordt een gebruiksvriendelijke tool, die na een eenvoudige muisklik het gewenste resultaat op een grafisch duidelijk manier kan weergeven. Waar niet enkel ontwerper maar iedereen die op dat moment inspraak heeft in het ontwerp eenvoudig mee aan de slag kan.

3. Opbouw

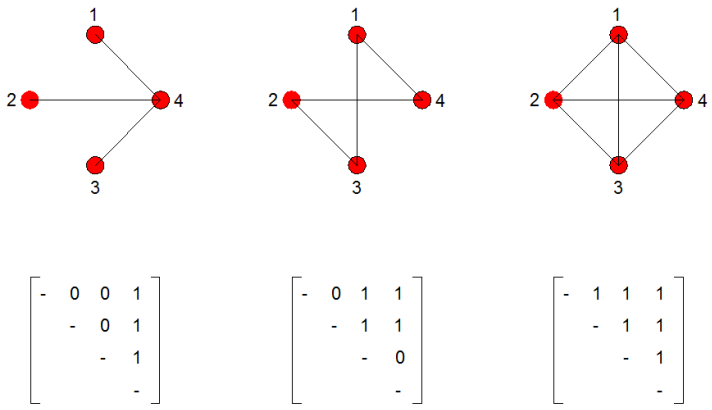
Deze masterproef is uiteindelijk een evolutie geworden van het onderzoek naar de rol van de computer in het ontwerpproces vertrekkend vanop de positie waar hij tegenwoordig het meest gebruikt wordt, namelijk op het einde als presentatietool (maar ook steeds meer als analysetool).

Het vertrekpunt in hoofdstuk 2 is het opstellen van het BIM-model (Building Information Model) van de case: het detentiehuis. Dit is een model dat in de huidige architectenpraktijk vaak opgesteld wordt, omdat dit een volledig geïntegreerd model is waar je een 3-dimensionele voorstelling hebt die rechtstreeks verbonden is met de 2-dimensionele snedes (grondplannen, doorsnedes en gevels). Het belangrijkste in dit model is dat er effectief informatie wordt toegekend aan de objecten van het gebouw. In dit programma (in tegenstelling tot bijvoorbeeld AutoCAD) wordt werkelijk aangegeven dat het om een gebouw gaat, niet om louter geometrie.

Dit model is uitstekend geschikt als analysemodel om een uitgebreid aantal dingen te kunnen analyseren zoals thermische en akoestische prestatie van de scheidende elementen, bouwknopen, evacuatieroutes enzovoort. In dit onderzoek wordt gebruik gemaakt van de functies: ruimtes en wanden een type en naam toekennen. Op deze manier is het dus mogelijk om te kijken welke scheidende elementen welke ruimtes scheiden. Hieruit werd vervolgens een nabijheidmatrix opgesteld. Deze matrix geeft de ruimtes weer welke in het

gebouw naast elkaar liggen. Als verdere uitbreiding is het ook mogelijk te weten welke ruimtes rechtstreeks verbonden zijn via fysiek betreedbare manieren.

De nabijheidmatrix die nu helemaal op het einde automatisch gegenereerd kan worden is eigenlijk een conceptuele schets die je aan het begin van een ontwerp simpel opstelt door eerst een graaf van het ontwerp te tekenen. Deze wordt opgebouwd door elke ruimte in een cirkel neer te schrijven en vervolgens de cirkels met elkaar te verbinden met verschillende soorten lijnen die weergeven welke relaties je al dan niet wil hebben in je gebouw zoals weergegeven wordt in de onderstaande afbeelding. Hieruit wordt vervolgens de nabijheidmatrix afgeleidt. Deze matrix is als uiteindelijk doel van deze masterproef de input voor de applicatie.



Figuur 1 – Nabijheidmatrix opstellen

Voor er hiertoe wordt gekomen, wordt in hoofdstuk 3 de moeilijk bereikbare doelstelling bekeken om een ontwerpopgave te vertalen naar een ontwerp-'probleem' wat nodig is om duidelijk te maken aan de computer wat hij moet doen. Er worden verschillende mogelijkheden getoetst om een generisch model te ontwikkelen dat het ontwerp van de case kan genereren.

Het zal steeds de bedoeling blijven om de grafische representaties weer te geven, aangezien het hier toch om een ontwerpopgave gaat en dit traditioneel zo benaderd wordt en het achterliggende zo ook duidelijker wordt. Maar voor het betere samenspel met de computer wordt in hoofdstuk 4 overgeschakeld naar een tekstuele representatie van ruimtes. Een gevolg hiervan is dat de relaties tussen ruimtes puur op basis van deze tekstregels zullen worden weergegeven in plaats van met geometrische regels. Dit zal als voordeel hebben dat de computer hier beter en vooral sneller mee om kan. Het blijft wel mogelijk om die tekstregel om te zetten in een ruimtelijke schets van het ontwerp om het realiteitsgevoel niet te verliezen.

In het *2e deel* en in hoofdstuk 5 van deze masterproef wordt de ontwikkelde applicatie volledig besproken met alle achterliggende wiskunde. Ook wordt het belang van de visie op

het gebruik van de applicatie geduid. Het is belangrijk al van bij het begin te onthouden dat het niet de bedoeling is om de output van deze applicatie letterlijk te vertalen naar een grondplan. Aangezien een ontwerpproces een niet volledig beschrijfbare opgave is kan deze applicatie ook niet anders dan een deeloplossing zijn van een vereenvoudigd probleem dat afgeleid werd uit de ontwerpopgave.

4. Case: detentiehuizen

De case die behandeld wordt is mijn ontwerp voor een haalbaarheidsstudie met als ontwerpopgave detentiehuizen in een stedelijke context in het kader van de masterstudio architectuur en constructie bij prof. Ronald De Meyer in het schooljaar 2011 – 2012.

4.1. Achtergrond

Het onderzoek kadert in een visie van Hans Claus, gevangenisdirecteur van de gevangenis van Oudenaarde. Het concept van detentiehuizen in plaats van de grootschalige opsluitmachines. "*Ik vertrek niet vanuit de doelen maar eerder vanuit de praktijk. Van hoe zij het samenleven met andere gedetineerden en met het personeel binnen onze cellulaire gevangenissen ervaren."* (Claus, 2009)

4.1.1. *Traditioneel*

Het klassieke model van de negentiende-eeuwse gevangenis is het model van het panopticon: een gebouw met een centrale kern waar een bewaker zit en verschillende deelgebouwen met cellen die uitkomen op die centrale plek. Op deze manier is met een minimum aan personeel en een minimum aan camera's de beveiliging gegarandeerd. In principe heb je zelfs geen personeel nodig die daar fysiek aanwezig zijn. Een voorbeeld van een dergelijke gevangenis is er in Gent: 'de Nieuwe Wandeling' (afbeelding).

Figuur 2 - Gevangenis nieuwe wandeling in Gent

Een gevangenis is de forensische context waarin negativiteit overheerst. Een gevangenis is in wezen probleemgericht. Een detentie berooft gedetineerden niet alleen van hun vrijheid, maar ontneemt hen ook op verschillende manieren hun keuzevrijheid. Dit zorgt voor detentieschade (het ontwrichten van het sociale leven, afhankelijkheid, psychische schade) bovenop de vrijheidsberoving. Zeker wanneer iemand een lange tijd geïnterneerd wordt. Hierbij gaan gedetineerden zich steeds meer gaan richten op de regels en eisen van het sociale leven binnen de gevangenismuren, met een vervreemding van de normen en waarden van de samenleving als gevolg.

Een goede straf zou eigenlijk een intensievere, een tijdelijke residentiële en deels gedwongen vorm van zorg kunnen zijn. (Claus, 2009)

4.1.2. *Visie*

Het idee vertrekt vanuit de getuigenissen van de gedetineerden zelf en ontwikkelt een visie op de “nieuwe huizen” zoals Claus ze noemt. Centraal staat de stelling dat het cellulaire concept het best verlaten wordt. Het voorstel is om enkele kleinschalige huizen te bouwen die opgebouwd zijn rond zorg en begeleiding van de gedetineerden in de plaats van het vertrekpunt van het opsluiten.

Concreet is het doel om in de loop van de tijd 35 gevangenissen te vervangen door 900 kleinschalige huizen. Deze kleinschaligheid impliceert een 10-tal gedetineerden per huis met enkele begeleiders. De bewoners van de huizen zouden eerstelingen zijn, zonder gevangeniservaring; met als uiteindelijk doel de instroom naar de gevangenissen in te dijken. Maximaal een 5-tal detentiehuizen in elkaars omgeving verspreid over het stedelijke weefsel zodat het haalbaar is om bepaalde collectieve functies te voorzien die inzetbaar zijn in de onmiddellijke buurt.

De voordelen zijn: een persoonlijke aanpak, meer het gewone leven benaderen, aanzetten tot het nemen van verantwoordelijkheid, begeleiding op maat, soepeler bezoek, in de buurt van familie enz. Principes van de basiswet die op deze manier beter uitvoerbaar zijn.

Voor een financieel realistische aanpak moet het personeelsaantal en het grondgebruik per detentiehuis best beperkt blijven, aangezien dit ook een punt is dat steeds zwaar zal doorwegen voor dergelijke beslissingen.

4.1.3. *Kritiek*

De voornaamste kritieken zijn net dat financiële aspect, maar ook het maatschappelijke. Hoe goed het idee misschien klinkt, er zijn niet veel mensen die graag willen dat er een detentiehuis in hun buurt gebouwd wordt. Het is net zoals bij windmolens, men zegt: “Ja graag, maar niet in mijn achtertuin”. Zo lang ze er geen last van hebben, kan alles.

In de haalbaarheidsstudie wordt ietwat rekening gehouden met deze kritieken, enerzijds door de oppervlakte van het gebouw te beperken voor de financiële kant en anderzijds door bepaalde delen van het gebouw open te stellen voor de buurt op sommige uren van de dag. Dit enerzijds ter compensatie en om het idee meer aanvaardbaar te maken door iets terug te geven aan de buurt en anderzijds ook om een interactie te kunnen aangaan tussen de buurt en de gedetineerden.

4.2. Het ontwerp

Het gaat om een haalbaarheidsstudie van 1 detentiehuis in de stedelijke gordel rond Gent. Het doel was wel om een generisch model te ontwikkelen zodat deze flexibel is naar verschillende contexten en sites, maar ook flexibel naar de verschillende soorten strafuitzittingsmodellen. Het ontworpen model zou vervolgens ook gebruikt kunnen worden voor een ander segment in de zorgsector zoals gemeenschapshuis, bejaardentehuis enz. Dit generieke leent zich ook uitstekend als uitgangspunt voor dit onderzoek, waar het ontwerp tot zijn essentie zal moeten worden uitgepuurd om vervolgens als generisch model terug op te bouwen om te komen tot een ontwerp aangepast aan situatie, locatie en ook aangepast aan de invloed van de gebruiker.

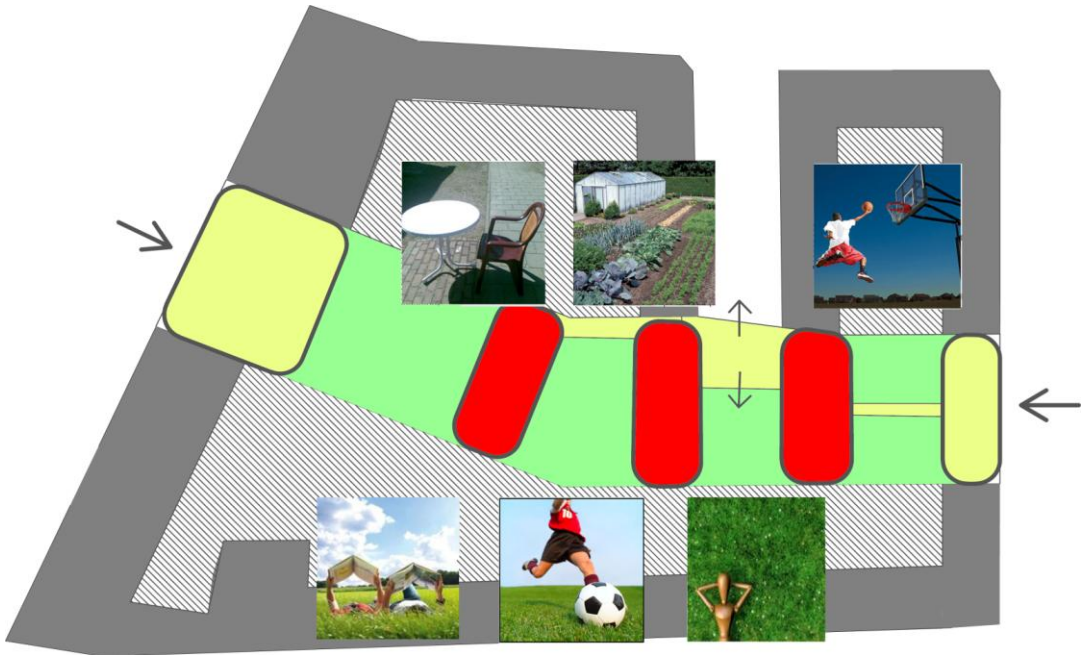
Voor dit ontwerp werd het zwaarste type bestraften gekozen, ook het moeilijkste: de strafuitboetingshuizen. Deze mensen mogen het detentiehuis nog niet verlaten en zijn dus afhankelijk van alle voorzieningen binnen het detentiehuis zelf. Mits een bepaalde dubbele functie van bepaalde delen van het gebouw kan er wel een interactie bestaan tussen gedetineerden en buurtbewoners; wat wel noodzakelijk is voor de aanvaarding in de buurt en ook als begin van een integratie in de maatschappij.

4.2.1. *Programma*

- Diensttoegang, personeelstoegang, bezoekerstoegang, bewonerstoegang, fouilleerruimte, wachtruimte
- Administratieve ruimte, medisch kabinet
- Kleed- en eetruimte personeel
- Individuele verblijfsruimte (+individueel sanitair en kitchenette)
- Collectieve verblijfsruimtes, sportruimte, waslokaal
- Therapielokaal, lokalen voor beroepsopleiding, crea-ruimte
- Parkeergelegenheid

4.2.2. *Concept*

Het globaal idee is dat de gedetineerden daar minstens 4 jaar zitten en niet steeds dag in dag uit op dezelfde plek hoeven te zijn. Het ontwerp bestaat uit het creëren van verscheidene plekken die een variabele manier van ‘de dag doorbrengen’ kunnen teweegbrengen. Dit wel steeds binnen eenzelfde schema van de dagorde, wat bevorderend is voor de herontwikkeling van de gedetineerde.



Figuur 3 - Case detentiehuis: conceptplan

4.2.3. *Plannen en Beelden*

Bijlage A

2. EVALUATIE

Dit hoofdstuk beschrijft de meer traditionele benadering van hoe de computer ingezet kan worden aan het einde van het ontwerpproces om vervolgens een model te ontwikkelen dat ingezet kan worden voor analyse en om ten slotte achteraf aanpassingen te doen aan het ontwerp. Men kan bijvoorbeeld de bouwfysische of de akoestische prestaties van de schil nagaan door middel van een analyse van de scheidende delen enerzijds tussen verschillende ruimtes en anderzijds ook naar de buitenomgeving toe. Nadien kan men met deze vergaarde informatie aanpassingen doen aan het ontwerp. Dit is geen directe interactie met het ontwerp maar wel al ietwat interactief.

Deze werkwijze is al interessanter dan louter het gebruik van de computer als visualisatietool, aangezien men ook dit 'building information model' (BIM) kan gebruiken om mee te ontwerpen. Er wordt onmiddellijke feedback gegeven op verschillende niveaus van constructie en bouwkunde waardoor je een directe terugkoppeling kan maken naar je ontwerp voor aanpassingen. Maar naar eigen ervaring met software zoals Autodesk Revit is gebleken dat deze ontwerpomgeving creatief beperkend is in de eerste fasen van het ontwerpproces aangezien er al te snel bepaalde gedetailleerde beslissingen moeten worden genomen en ook het creëren van niet-traditionele vormen is moeilijk.

Aangezien dit onderzoek verder bouwt op een reeds bestaand ontwerp is dit wel een geschikt vertrekpunt om omgekeerd te bekijken hoe het ontwerp kan worden geanalyseerd

en eventueel vereenvoudigd tot bepaalde analytische schema's die dan op hun beurt als input kunnen dienen voor het verdere onderzoek om op die manier meer tot het ontwerp te komen.

1. Building Information Model

In het domein van architectuur, bouwkunde en constructie (AEC: architecture engineering and construction) is het bijna onmogelijk om een werkend prototype van een ontwerp te maken. De bouw zelf is meestal het prototype, wat fouten en kostenoverschrijding veroorzaakt. Het is tegenwoordig wel mogelijk om een voorafgaande planning en analyse te maken door middel van een Building Information Model (BIM). Dit is een geïntegreerd model dat het gebouw weergeeft met een digitale representatie van de fysische en functionele eigenschappen. Dit model is een bron van gedeelde kennis om van bij het begin van het bouw- en ontwerpproces als steun te dienen om ontwerpmatige en constructieve beslissingen te nemen.

1.1. Achtergrond

Het idee van BIM bestaat reeds sinds de jaren 1970 en de term kwam eerst aan bod in een paper door (van Nederveen, 1992). Het meer populaire gebruik van de term BIM kwam er sinds het uitbrengen van het paper "Building Information Modeling" door (Autodesk, 2003). Het eerste gebruik van BIM was deel van het 'virtual building' concept van ArchiCAD (door Graphisoft). (Eastman, 2011)

1.2. Definitie

"Building Information Modeling (BIM) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition." (National Building Information Model Standard Project Committee)

Een traditioneel ontwerp van een gebouw bestaat hoofdzakelijk uit 2-dimensionele tekeningen (grondplannen, doorgesneden, gevels enz.). Building information modeling wil dit uitbreiden voorbij het 3-dimensionele (breedte, hoogte, diepte) met tijd als 4^e en kost als 5^e dimensie. BIM betekent dus veel meer dan puur geometrie. Het betekent ook ruimtelijke

relaties, lichtanalyses, geografische informatie, hoeveelheden en eigenschappen van gebouwcomponenten.

BIM geeft een ontwerp weer als een verzameling van objecten die drager zijn van geometrie, relaties en attributen. De ontwerptools in de BIM-omgeving laten toe om verschillende zichten op het ontwerp te bekijken. Deze verschillende zichten zijn automatisch samenhangend. Een wijziging van een object in één zicht, slaat deze nieuwe informatie in het object zelf op en zal dus in alle zichten gewijzigd worden, aangezien alle objecten gegenereerd worden op basis van één enkele definitie. BIM is ook parametrisch, aangezien wanneer er bepaalde objecten van elkaar afhankelijk zijn en één object wordt gewijzigd, ook het andere object mee verandert in functie van de ingestelde afhankelijkheid.

Voor alle partijen die deelnemen aan het gebouwproces biedt BIM een virtueel informatiemodel dat door iedereen gebruikt kan worden om zijn eigen nodige informatie eruit te halen.

Het gebruik van BIM is toepasbaar voor de volledige cyclus van het project, van conceptfase in het ontwerpproces tot ondersteunende processen zoals kostenbeheer, constructie, project management en facilitair management.

1.3. Toekomstpotentieel

BIM is een relatief nieuwe technologie en zal dus in deze industrie typisch redelijk traag opgenomen worden als veelgebruikte werkwijze. Toch zijn de huidige gebruikers zeker dat BIM zal groeien en een belangrijke rol zal spelen in het documenteren van gebouwen. Het gaat hier dus niet over het gebruik van BIM in de architectenpraktijk, want daar bestaat nog veel argwaan onder andere over de beperkte ontwerpomgeving die creatief zeer beperkend is.

De voorstanders stellen dat BIM volgende verbeteringen biedt:

- Verbeterde visualisatie
- Verhoogde productiviteit door het gemak van informatievergaring
- Verhoogde coördinatie van constructiedocumenten
- Inbedden en verbinden van informatie uit veel verschillende velden
- Verhoogde leveringssnelheid
- Verminderde kost

1.3.1. *Green Building XML*

Green Building XML (gbXML) is een deel van BIM dat zich richt op de groene kant van het ontwerpen. Het wordt gebruikt als input voor verschillende energiesimulaties.

Het is ontwikkeld als bestand om de interoperabiliteit tussen verschillende programma's die gebruikt worden in de bouwindustrie te garanderen. Deze interoperabiliteit vermindert algemeen de tijd om een gebouw te ontwikkelen en garandeert dat wanneer een gebouw gebouwd wordt het aan zijn ontwerpintenties kan tegemoetkomen.

Verder in dit hoofdstuk wordt de gbXML file gebruikt in het onderzoek om vanuit het model van het detentiehuis bepaalde informatie te extraheren om een matrix op te stellen die de relaties tussen de ruimtes weergeeft. Deze informatie kan dan omgekeerd gebruikt worden als input voor een ontwerp.

2. BIM: detentiehuis

Dit gedeelte bespreekt de opbouw en het gebruik van het model van de case in de BIM-omgeving van het programma Revit van Autodesk.

2.1. Opbouw model

In principe wordt een BIM-model compleet geïntegreerd opgebouwd naarmate het ontwerp vordert, met alle mogelijkheden die het programma biedt om informatie op te slaan in de objecten van het ontwerp. Vervolgens wordt het complete model geëxporteerd naar het gb.XML bestand. Het nadeel hiervan, voor dit onderzoek, is enerzijds dat er dan zoveel informatie wordt geëxtraheerd dat het onoverzichtelijk wordt welke informatie waar te lezen is en anderzijds is dit ook een indirecte analyse.

Voor dit onderzoek is ervoor gekozen om puur te kijken op vlak van relaties tussen ruimtes en op welke manier ze gescheiden zijn. Het gevolg voor het model is dat alle ruimtes enkel als eenvoudige dozen gemodelleerd zijn met ofwel een volle muur (zonder gedetailleerde opbouw) ofwel een glazen muur als scheidend element. In deze twee types muren zijn ook deuren geplaatst om een idee te hebben van welke ruimtes fysiek in elkaar overlopen en welke ruimtes enkel grenzen aan elkaar. Daarnaast is er ook een type onzichtbare wand gedefinieerd om in elkaar overlopende ruimtes te kunnen modelleren en scheiden, aangezien Revit niet de mogelijkheid biedt om twee verschillende types van ruimtes binnen eenzelfde volume afzonderlijk te benoemen.

Het is belangrijk om ook de buitenruimtes en de naburige gebouwen te modelleren om een zo volledig mogelijk beeld te krijgen van de relaties. Buitenruimtes moeten ook als volume gemodelleerd worden, niet als 'open ruimte' omdat het model hier anders gewoon geen rekening mee gaat houden. Dit is belangrijk als er gekeken wordt naar de lichtinval, maar ook naar de relaties: door te kijken welke ruimtes grenzen aan dezelfde buitenruimte. Dit is

een 3-dimensionele benadering met als gevolg dat ook verticale visuele relaties bekeken kunnen worden, wat een belangrijk inzetpunt was in het ontwerp van de case.

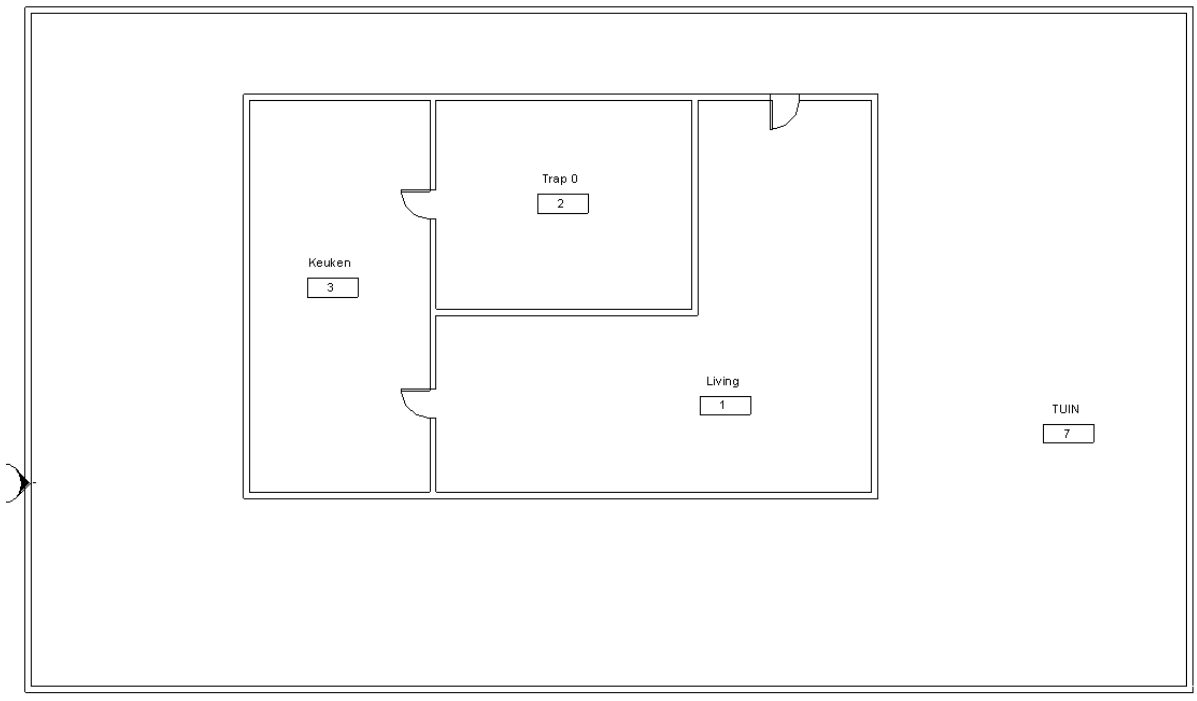
Natuurlijk kan dit onderzoek nog verder uitgebreid worden door steeds meer informatie in het model te steken en ook deze te gebruiken voor de analyse. Hierover meer in een verder hoofdstuk.

2.1.1. *Voorbeeldmodel*

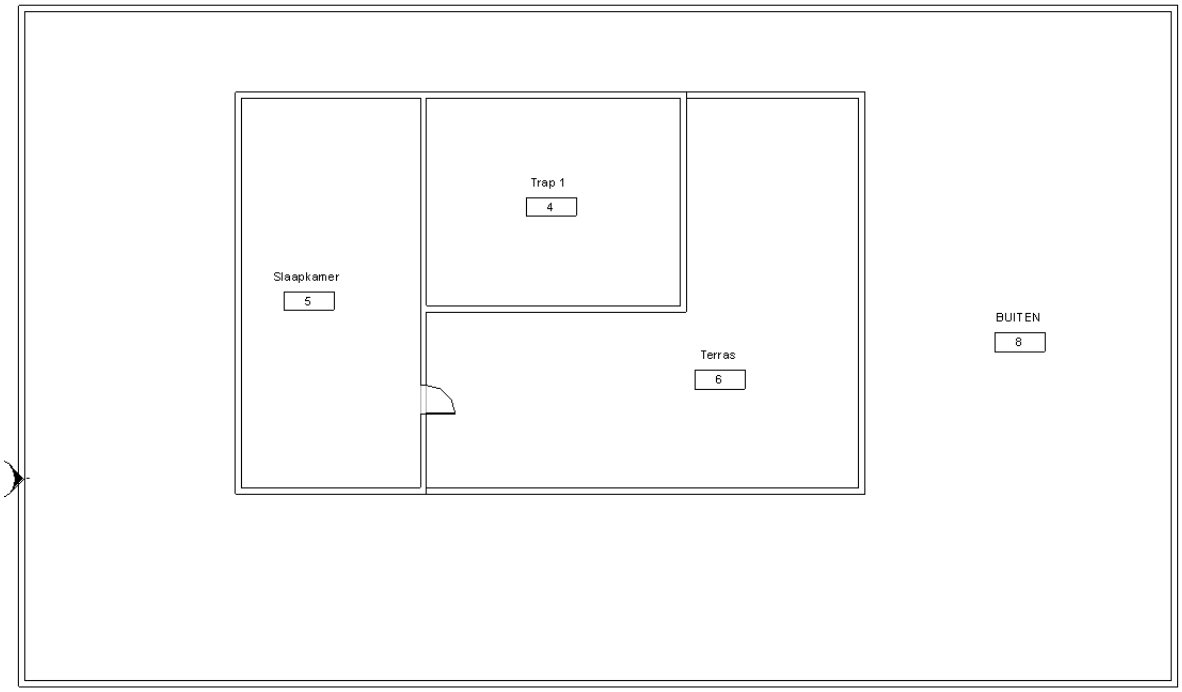
Als voorbeeld wordt onderstaand eenvoudig model van een simpele woning kort besproken om de principes duidelijk te maken die in de case eventueel minder duidelijk zouden zijn, aangezien het model van de case van het detentiehuis al zeer uitgebreid en complex is.

Onderstaande figuren geven het model weer in de Revit-omgeving. Het is duidelijk in de grondplannen dat het om een simpel model gaat met enkel volle wanden en wat deuren. Er is ook voor gekozen om de wanden over de 2 verdiepingen te laten doorlopen om in het geanalyseerde model een duidelijke relatiematrix of nabijheidmatrix (en. adjacency matrix) te bekomen. In de 3D weergave is zichtbaar hoe de buitenomgeving ook als volume gemodelleerd wordt. In dit geval zijn wel lage wanden als representatief object gekozen, maar dit heeft geen invloed op het uiteindelijke resultaat aangezien deze enkel nodig zijn om de ruimte TUIN en de ruimte BUITEN te kunnen definiëren.

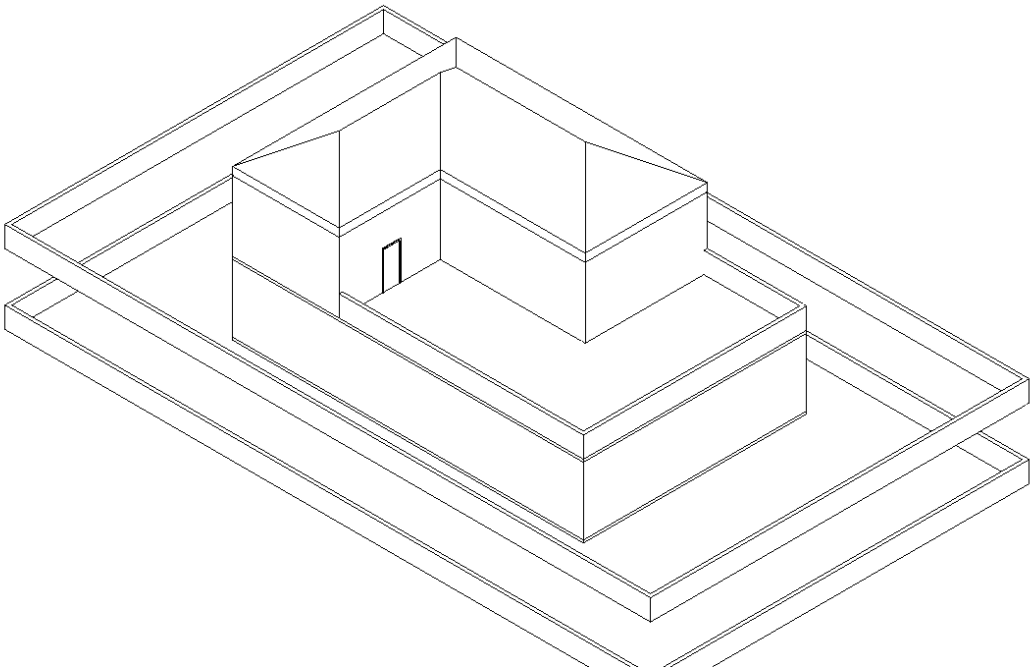
Een resultaat zal wel zijn dat TUIN en BUITEN een relatie hebben, maar dat is gemakkelijk interpreteerbaar. Hierover meer in volgende paragraaf.



Figuur 4 - Voorbeeldmodel: gelijkvloers



Figuur 5 - Voorbeeldmodel: 1e verdieping



Figuur 6 - Voorbeeldmodel: 3d-weergave

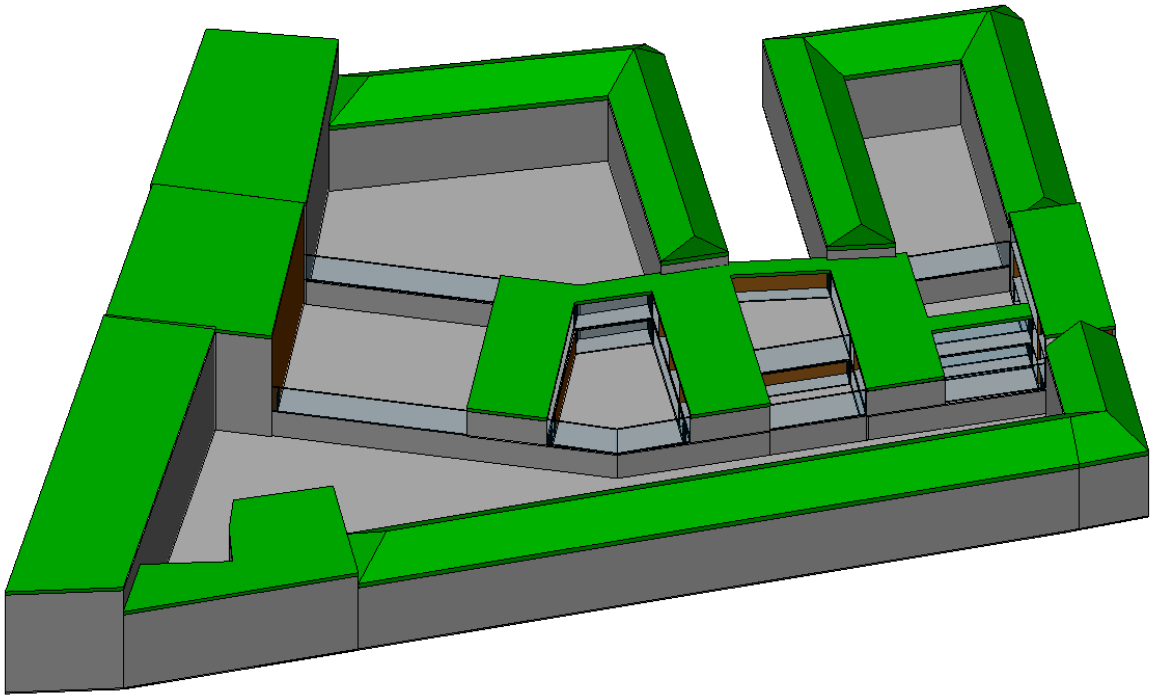
2.1.2. *Model van het detentiehuis*

Bij de opbouw van het model van het detentiehuis in Revit is vertrokken van de grondplannen van het bestaande ontwerp. Achteraf is gebleken dat voor het gemak van de analyse een vereenvoudigd grondplan makkelijker geweest zou zijn. Maar voor het meer realistische idee van het vertrekken van het bestaande gegroeide ontwerp model is dit wel een betere benadering.

Op onderstaande beelden is zichtbaar hoe het Revit-model in vorm wel het ontwerp benadert mits enkele vereenvoudigingen. Waar het voorbeeldmodel slechts uit 8 ruimtes bestond, bestaat dit vereenvoudigd detentiemodel al uit 22 ruimtes. Vereenvoudigd omdat niet alle ruimtes als aparte ruimtes gemodelleerd zijn om het overzicht wat te bewaren. Maar in principe als de analysetool werkt voor dit vereenvoudigd model kan dit zeker ook werken voor een volledig model.

In dit model is ervoor gekozen om de daken ook een ander type scheidingswand te geven, zodoende kan je hiervoor ook een aparte klasse maken bij de analyse om te zien welke volumes een plafond hebben dat grenst aan de buitenruimte zonder de volledige buitenruimte boven elk volume te moeten modelleren. Dit zou handig kunnen zijn indien je een evaluatie wilt maken van de energieprestatie of indien er een akoestische analyse wordt gedaan kan hierdoor gekeken worden welke ruimtes onder het dak zitten en dus hinder kunnen ondervinden van regen.

Figuur 7 - Detentiehuis render AUTOCAD



Figuur 8 - Detentiehuis 3d-model REVIT

3. Evaluatie

Onderstaande figuur is een sectie van het grondplan op het gelijkvloers om de opbouw meer gedetailleerd te visualiseren.

Er zijn 4 types wanden zichtbaar: buitenwand, binnenwand, transparante wand en onzichtbare wand. Het onderscheid tussen de verschillende wanden is eigenlijk overbodig om puur de relaties te bekijken tussen de verschillende ruimtes. Toch wordt dit gedaan om de mogelijkheden van het model duidelijk te maken. Het onderscheid tussen binnen- en buitenwanden is om een extra evaluatie op energieprestatieniveau toe te laten. Zoals eerder vermeld, zijn de transparante wanden naast de energie redenen voor de zichten en lichtinval en zijn de onzichtbare wanden voor doorlopende ruimtes (binnen en buiten).

Overal zijn deuren getekend om de fysieke toegangen duidelijk te maken. Dit is ook niet nodig voor de puur relationele matrix maar dit kan dan wel gebruikt worden voor analyse van de kortste weg tussen ruimtes of voor evacuatiedoeleinden.

Figuur 9 - Detentiehuis REVIT gelijkvloers (fragment)

Als het model klaar is wordt het geëxporteerd naar een gb.XML bestand dat de mogelijkheid biedt om via tekst het ontwerp te evalueren.

3.1. gb.XML

Onderstaande figuur geeft de informatie weer van één scheidend oppervlak zoals kan worden geëxtraheerd uit het gb.XML bestand ter verduidelijking. Elke informatie die in het BIM-model wordt toegekend aan dit object wordt hier weergegeven: de geometrie, het typemuur dat je zelf gekozen hebt en het belangrijkste voor het analyseren van relaties is de 2 ruimtes welke aan dit oppervlak grenzen. Aan elk kruispunt wordt een wand opgedeeld in Revit zodat elk uniek wandnummer slechts 2 ruimtes scheidt. Ook kan gezien worden in dit voorbeeld dat deze wand een deur bevat. Deze informatie kan in verdere analyses nog gebruikt worden, maar niet binnen het kader van dit onderzoek.

```
<Surface id="su-151" surfaceType="InteriorWall">
  <Name>N-14-21-I-W-151</Name>
  <AdjacentSpaceId spaceIdRef="sp-14-circ_c-d_0">
  </AdjacentSpaceId>
  <AdjacentSpaceId spaceIdRef="sp-21-Living">
  </AdjacentSpaceId>
  <RectangularGeometry>...</RectangularGeometry>
  <PlanarGeometry>...</PlanarGeometry>
  <Opening id="su-151-op-1" openingType="NonSlidingDoor">...</Opening>
  <CADObjectId>Basic Wall: _detentiemuur binnen [203572]</CADObjectId>
</Surface>
```

Figuur 10 - gb.XML informatie van 1 oppervlak als voorbeeld

3.2. Applicatie

Om de informatie van de gb.XML file te kunnen lezen en verwerken tot de informatie die nodig is voor het vervolg van dit onderzoek werd een kleine consoleapplicatie geschreven in programmeertaal C#. Dit om te illustreren wat de mogelijkheden zijn met de informatie die in een BIM-model gestoken wordt.

De applicatie is opgebouwd met een XML reader, wat geïntegreerd zit in C#. De reader leest doorheen het volledige bestand en slaat de informatie van de verschillende 'nodes' op. Een node is een object van het model. In het voorbeeld is een node het oppervlak. Dit oppervlak bevat dan verschillende informatiedragers of 'attributes'. Bijvoorbeeld wordt er bij de surface het attribute "id" opgeslagen met als gevolgd dat de naam "su-151" zal opgeslagen worden in een lijst.

In deze lijst worden steeds het oppervlak en vervolgens de aangrenzende ruimtes opgeslagen. Met als gevolg dat uit die lijst een matrix kan worden opgesteld welke aantoont welke ruimtes naast elkaar liggen. Dit gebeurt door middel van een bestaande lege vierkante matrix met overal nullen, waar er een 1 geplaatst wordt op de positie waar een relatie bestaat tussen 2 ruimtes. Als voorbeeld, als er een relatie is tussen ruimte 4 en 7 zal op locatie [4,7] in de matrix een 1 staan. Het gaat over een 3-dimensioneel model dus de relaties werken ook verticaal.

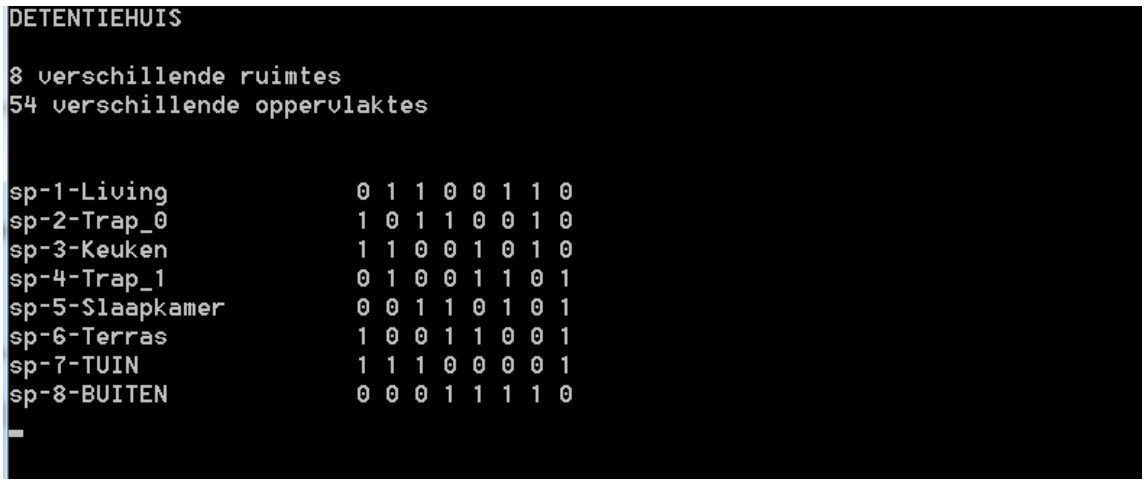
In het verdere verloop van dit onderzoek zal op het gebruik van de nabijheidmatrix met een 3-dimensioneel model verder ingegaan worden. Verticale relaties zullen anders moeten weergegeven worden dan relaties binnen eenzelfde niveau om het onderscheid te kunnen behouden.

3.2.1. *C# code*

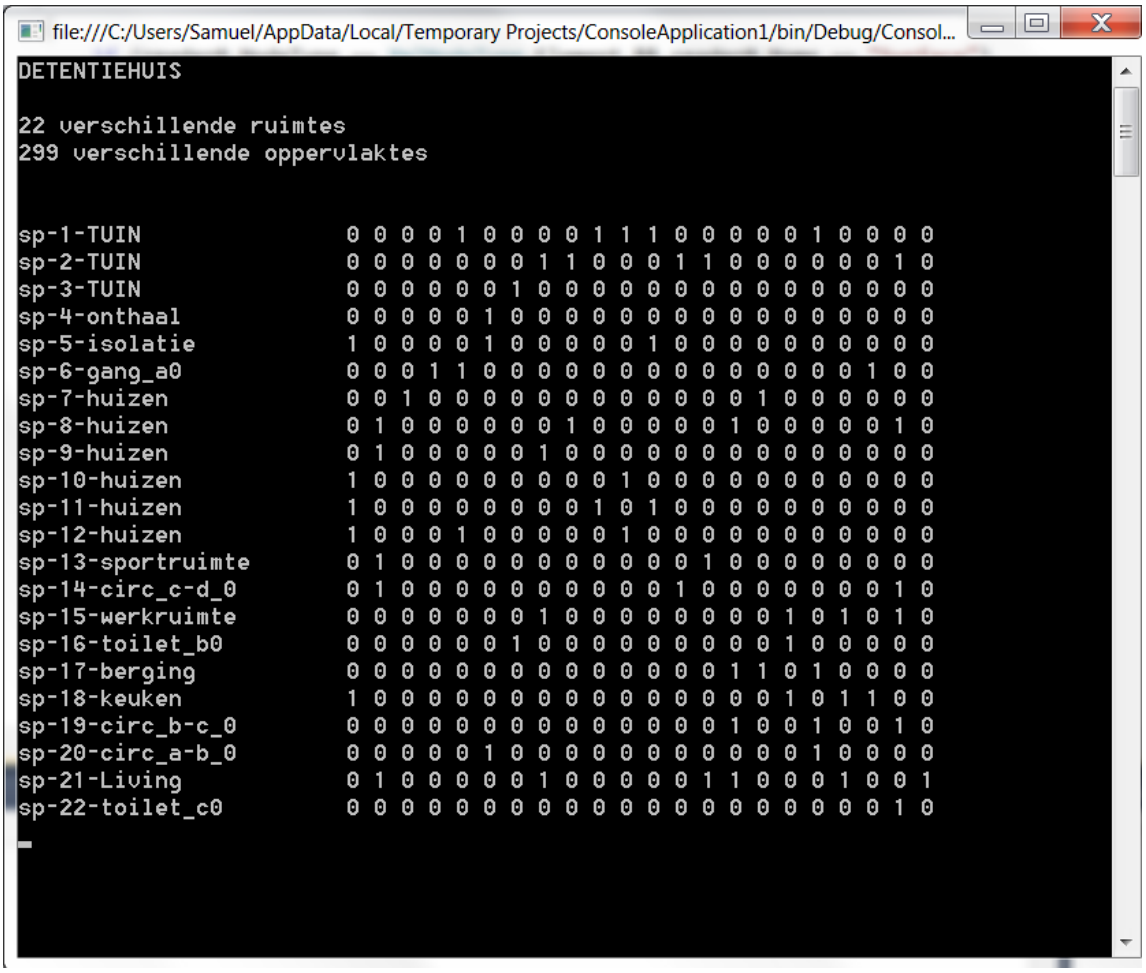
Zie bijlage B.

4. Resultaat

De onderstaande afbeeldingen geven het resultaat van de applicatie voor zowel het voorbeeldmodel (2.1.1) als het model van het detentiehuis (2.1.2). De nabijheidmatrix is een vierkante matrix, symmetrisch ten opzichte van de nu nog zinledige diagonaal (het zou mogelijk kunnen zijn om de oppervlaktes van de ruimtes op de diagonaal weer te geven). Voor de visualisatie in het consolescherm was het niet mogelijk om op de horizontale as de legende van de ruimtes weer te geven, maar door de symmetrie heeft deze dus dezelfde legende als de verticale as.



Figuur 11 - Voorbeeldmodel: adjacencymatrix



Figuur 12 - Model detentiehuis: adjacencymatrix

4.1. Mogelijkheden

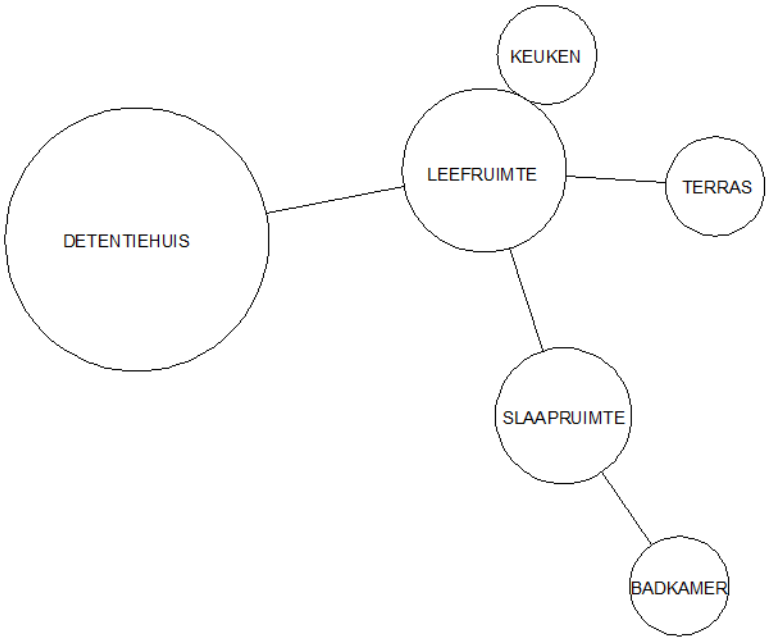
De mogelijkheden voor het verdere gebruik van dit principe om informatie uit een BIM-model te halen zijn talrijk. Naast louter de nabijheidmatrix waar je de relaties tussen de ruimtes kan afleiden zijn er veel meer mogelijkheden, in functie van welke informatie of attributes je toekent aan de objecten in het model.

Zoals eerder aangehaald zijn voorbeelden hiervan evaluatie van energieprestatie en akoestiek door middel van toekennen van eigenschappen aan de scheidende elementen. Andere mogelijkheden zijn evacuatie of kortste-weg evaluaties door deuren in wanden te modelleren enz.

Door eenvoudige aanpassingen aan het script van de applicatie kan je bepaalde informatie extra meegeven met de matrix. Er kan bijvoorbeeld een cijfer 2 gezet worden tussen ruimtes die met een deur verbonden zijn, een cijfer 1 voor gewone horizontale relaties en een cijfer 3 voor een verticale niet fysiek betreedbare relatie, dit is een relatie tussen 2 ruimtes boven elkaar. Daarnaast kan je ook de verticaal fysiek betreedbare relaties met nog een ander cijfer aanduiden, dit is bijvoorbeeld voor trappen of liften.

4.2. Besluit

Dit resultaat van de nabijheidmatrix kan op een meer visuele manier weergegeven worden door middel van een graaf. Dit is een figuur waarbij elke ruimte in een cirkel wordt getekend en onderling wordt verbonden door middel van al dan niet gekleurde lijnen om bepaalde relaties te leggen. Dit is een manier die veel gebruikt wordt door ontwerpers in een vroeg stadium van het ontwerp om de eerste ideeën en relaties van het opgelegde programma vast te leggen en te analyseren.



Figuur 13 - Voorbeeldgraaf: woonunit detentiehuis

Dit idee is wat meegenomen wordt naar het vervolg van het onderzoek, om deze relaties van ruimtes als vertrekpunt te nemen voor een generatief ontwerp. Dit is het omgekeerde doen van wat tot nu toe gedaan is: in plaats van te vertrekken van een ontwerp en zo een vereenvoudigde matrix van relaties op te stellen, wordt het de bedoeling om nu net deze relaties te gebruiken als beginvoorwaarde voor een ontwerp, als sturende factor voor het genereren van grondplannen in een bepaalde context.

3. ONTWERP

Dit hoofdstuk zal het resultaat van hoofdstuk 2 als beginpunt nemen: de relaties tussen ruimtes als eerste stap in een ontwerpproces. Dit beginpunt wordt op verschillende manieren benaderd om hieruit een ontwerp tot stand te doen komen. Er wordt vertrokken vanuit de case van detentiehuizen en verder in het hoofdstuk wordt er meer toegespitst op grondplan lay-out van het kleiner onderdeel van de individuele verblijfsruimte om de complexiteit niet te groot te maken. Dit allemaal met 3-dimensionele representaties van elk model.

1. Kader

1.1. Ontwerpproces

Het is moeilijk om juist te omschrijven wat een ontwerpproces is. De meeste literatuur (Kalay, 1985) is het erover eens dat ontwerp een doelgericht proces is dat gericht is op het bedenken van voorwerpen en omgevingen die binnen bepaalde randvoorwaarden vooropgestelde doelen willen bereiken. Omdat er geen exacte formules bestaan die deze

doelen en randvoorwaarden kunnen omzetten in een samenhangende fysieke vorm is ontwerpen een iteratief, educatief 'trial-and-error' proces dat zeer sterk afhankelijk is van kennis en ervaring.

Op deze manier heeft ontwerpen veel gemeen met algemene 'problem-solving' processen zoals ze beschreven werden door onderzoekers als Newell en Simon die sinds de jaren 1960 veel onderzoek hebben gedaan naar artificiële intelligentie. In een vroeg onderzoek hebben ze een programma GPS-I opgesteld, als onderzoek om te begrijpen welke processen er onder de menselijke intellectuele, aanpasbare en creatieve mogelijkheden liggen (A., Shaw, & Simon, 1959). Dit is een synthetisch proces: een computerprogramma schrijven dat intelligentie en ervaring vraagt. Het is daarentegen wel interessant te kijken welke verschillen het programma geeft ten opzichte van menselijke 'problem-solving'.

Volgens bovenstaande theorieën kan dus gesteld worden dat er voor elk probleem een oplossingenruimte kan gedefinieerd worden die alle mogelijke oplossingen van het probleem bevatten. 'Problem-solving' kan dus gedefinieerd worden als een zoekproces door al de alternatieve oplossingen van deze ruimte met als doel één of meerdere oplossingen te vinden die voldoen aan bepaalde vooropgestelde doelen (Kalay, 1985). Het woord zoeken wordt metaforisch bedoeld: enerzijds om het proces van zoeken en het evalueren van alternatieve oplossingen te beschrijven ten opzichte van het gehele probleem en anderzijds ook ten opzichte van kleinere subproblemen.

De oplossingenruimte van een ontwerpprobleem bestaat uit een reeks bestaanstoestanden die elk een specifieke oplossing representeren. Één of meer van deze toestanden geven de huidige staat van het ontwerp weer (zodat er ruimte is voor alternatief). Het proces kan dus gezien worden als een opeenvolging van acties die de huidige toestand van een ontwerpproces vooruithelpen van de ene toestand naar de volgende.

Deze ideeën worden in dit hoofdstuk toegepast, waar duidelijk gesteld moet worden dat dit slechts een aanzet is voor een ontwerp en moeten kritisch worden teruggekoppeld naar het veel complexere ontwerpproces.

Het ontwerpproces is een dynamisch wederkerig proces, dat wel beschreven kan worden als het oplossen van een ontwerpprobleem, maar het woord probleem is hier te technisch, aangezien men bij ontwerpen het 'probleem' niet concreet kan beschrijven zoals nodig is voor een computerprogramma. Een ontwerpproces is een proces dat zoekt naar een bevredigend resultaat. Dit kan worden bereikt door creatief om te gaan met de doelstellingen en gebruik te maken van ervaring en intelligentie (kennisbasis).

1.2. Layout 'Problem'

Ruimtelijke configuraties en 'layout problems' zijn alomtegenwoordig in het architecturale ontwerp. Ook zijn ze een van de meest complexe problemen. Het komt voor op

verschillende schalen: van de schaal van de stad tot het grondplan van een gebouw tot interieuroplossingen.

Dergelijk probleem duikt op in het ontwerp en de toewijzing van ruimtes in een nieuw gebouw of in het herschikken van ruimtes in een reeds bestaand gebouw. Tijdens de conceptuele ontwerpfase wordt de toewijzing van ruimtes in een nieuw gebouw gebruikt om alternatieve oplossingen voor de configuratie van het gebouw te testen. Grondplannen kunnen geëvalueerd worden op het beste gebruik van de ruimte om elementen te bepalen zoals optimaal aantal niveaus, de omtrek van het grondplan enz. In een bestaand gebouw kan het gebruikt worden om een bestaand 'space-management' probleem op te lossen. (Liggett, 2000) haalt volgend voorbeeld aan: wanneer in een bedrijf de grootte van een projectgroep toeneemt of afneemt moet de nabijheid van alle werknemers voldoende blijven met zo weinig mogelijk verandering van werkomgeving.

Het doel is om een bevredigende ruimtelijke samenhang van elementen te verkrijgen die overeenkomen met de vooropgestelde randvoorwaarden en doelstellingen. Dit wordt zeker niet enkel tegengekomen in de architecturale wereld, maar ook in velden van het grafisch ontwerp tot het ontwerpen van de opbouw van een elektrisch bord. In sommige van deze velden zijn de problemen goed gedefinieerd en ligt het voor de hand om de computer in te zetten om oplossingen te genereren.

In architectuur spreekt men liever niet van een ontwerpprobleem omdat je het niet enkel kan vatten in randvoorwaarden en doelstellingen. Het is veel complexer dan dat; alleen al de zoveel complexe criteria die niet volledig goed te definiëren zijn. Daarnaast is er dan ook de creatieve geest van de ontwerper die ook zijn onbepaalde verlangens heeft. Zolang de computer niet kan denken zoals de mens zal tot nader order de computer de taak van architect niet kunnen overnemen. Het is aan ons om hiervan goed bewust te zijn en de computer in te zetten waar het wel kan en dan het resultaat goed te interpreteren.

Ondanks deze moeilijkheden is er sinds de vroege jaren 1960 wel al veel onderzoek gebeurd. Ligett besluit daarentegen wel dat er nog weinig ontwikkeling is binnen de architecturale praktijk: *"In spite of the long research history associated with automated layout and space allocation systems, in practice these systems have not been utilized to their full potential."* (Liggett, 2000). Veel onderzoek focuste op optimalisatie, om die randvoorwaarden op te lossen die wel goed gedefinieerd kunnen worden binnen een layout-"probleem", terwijl de creatieve designer component genegeerd werd. Van bij het begin was er vanuit de architectenwereld verzet tegen die computationele automatisering. Steadman bijvoorbeeld, haalt de reactie aan van editors in de jaren 1970 op een paper geschreven door William Mitchel over zijn werk om een computerprogramma te ontwikkelen dat architecturale grondplannen kan genereren: *" This work is strictly non-architectural, … it has nothing to do with architecture"* (Steadman, 1983).

1.2.1. *Grondplan*

Een grondplan genereren is het plaatsen van ruimtes binnen een gebouw. In de praktijk zijn de inputgegevens meestal een lijst met ruimtes, bepaalde regels en doelen (al dan niet concreet gedefinieerd) en ook de intentie van de architect. De regels kunnen geometrisch of topologisch zijn.

De doelen zijn bijvoorbeeld: het uizicht vanuit een kamer met de privacy hieraan gekoppeld, de bouwfysische, akoestische of structurele eisen enz. Per ruimte kan er dan ook een prioriteit gekoppeld worden aan deze doelen om zo te kunnen duidelijk maken wat per ruimte als belangrijker beschouwd wordt. Bijvoorbeeld: de ontwerper wil in de leefruimte meer belang hechten aan licht en zicht dan aan thermische prestatie, maar in andere ruimtes wordt dit vervolgens gecompenseerd om toch de nodige eisen te halen voor het energieprestatieniveau van het gehele gebouw.

Bij het ontwerpen van een nieuw gebouw wordt vaak vertrokken van een graaf, een puur topologische schematisering. Dit is wenselijk om zo de ruimtelijke relaties te kunnen analyseren en wijzigen zonder enige geometrische randvoorwaarden. In dit hoofdstuk zal blijken dat geometrie in bepaalde gevallen wel onmisbaar zal zijn om tot bepaalde vooropgestelde doelstellingen te komen. Verder in hoofdstuk 4 wordt hierop teruggekomen om toch enkel met de relaties een ontwerp te genereren dat puur uit tekst zal bestaan en waarvan nadien een geometrische representatie kan worden gemaakt.

1.2.2. *Doelstelling*

De ontwerpruimte is vaak niet duidelijk gedefinieerd en kan zelfs veranderen tijdens het ontwerpproces. De ontwerper heeft eigen specifieke intenties en een brede waaier aan kennis en ervaring die niet expliciet kan worden vertaald naar een simulatie. Dus zal hij op een dynamische manier aan de simulatie moeten kunnen deelnemen:

“interactive and responsive use”

Door onmiddellijke feedback worden de regels en de onderliggende heuristieken duidelijk voor de ontwerper en zijn de effecten van de ontwerpbeslissingen transparant.

1.3. Shape Grammar

Een ‘shape grammar’ bestaat uit een reeks vormregels en een sturende motor om het proces te sturen en de juiste regels te selecteren. Een regel bepaalt hoe een bepaalde vorm kan getransformeerd worden en bestaat steeds uit 2 onderdelen: het linkse deel en het

rechtse deel met daartussen een pijl. Om deze regel dus te kunnen toepassen zal steeds eerst de bestaande vorm (of deel-vorm) gelijk moeten zijn aan het linkse deel, om deze vervolgens te kunnen omzetten in het rechtse deel.

Vormen, labels en gewichten kunnen gecombineerd worden om een 'shape grammar' vorm te geven. Meer nog kunnen deze grammars verschillende van deze componenten combineren om zo een andere blik te verkrijgen op de beschreven ontwerpen en op deze manier een volledig samengestelde grammar te verkrijgen. (Stiny, 1981)

In de verschillende grammars die in de literatuur vermeld worden is er steeds 1 gemeenschappelijk doel: namelijk het verlangen om een schema af te leiden dat zou kunnen worden gebruikt om een massaproductie te verkrijgen van betaalbare huizen, gebruikmakend van industrialisering om de kosten te drukken (Duarte J. , 2005). In het kader van de nieuwe technologieën met 'file to factory' principes is het tegenwoordig ook mogelijk om aan deze massaproductie ook een mogelijkheid te koppelen voor de gebruiker om zijn eigen woning te 'customizen'.

1.3.1. *Soorten Grammars*

Er zijn 2 soorten grammars: de analytische en de originele.

De analytische grammars worden ontwikkeld om de historische stijl of taal van bepaalde ontwerpen van architecten te beschrijven, oorspronkelijk van architecten die niet meer leven om hun werk te kunnen voortzetten. De eerste grammar werd ontwikkeld om het geheel van architectonische artefacten te beschrijven voor de Palladiaanse villa's. De jaren nadien zijn er nog verschillende grammars ontwikkeld met hetzelfde soort doel. Deze analytische studies gebruiken een reeks bestaande ontwerpen om de taal van de ontwerper te representeren en om deze om te zetten in de regels van de grammar. De grammar wordt vervolgens getest door de regels te gebruiken om bestaande en nieuwe ontwerpen te genereren.

De originele grammars bestaan uit het maken van nieuwe en originele ontwerpstijlen 'from skratch'. Het gebruik hiervan is nog niet zo diep onderzocht als het gebruik van analytische grammars.

1.3.2. *Duarte – Siza's Grammar*

Een belangrijk voorbeeld van 'shape grammars' is de actieve grammar voor de huizen van architect Siza in Malagueira onderzocht door José Pinto Duarte in zijn 2^{e} thesis.

De grammar van Siza's huizen in Malagueira is gebaseerd op de analytische studies zoals hoger beschreven werd. Maar toch is deze grammar ontwikkeld voor een bestaand project van een levende architect. Deze werd ook ontwikkeld met de steun van de ontwerper en kan dus wel gezien worden als een natuurlijke uitbreiding van zijn werk (Duarte J. , 2005). Het

positieve van het in leven zijn van de architect is tweevoudig. Enerzijds is het mogelijk om de architect en de bewoners samen met de bestaande ontwerpen in te zetten als bron van informatie om de regels af te leiden. Anderzijds is het mogelijk om de 'grammar' te gebruiken om nieuwe huizen te genereren en ook te bouwen in die taal.

In wat volgt wordt kort de grammatica van Siza's huizen beschreven om een idee te krijgen van hoe deze opgebouwd is zonder veel te diep hierop in te gaan. Voor de volledige grammatica wordt verwezen naar (Duarte J. P., Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses, 2001).

De grammatica zelf kan worden opgedeeld in 2 families van huizen, afhankelijk of ze een tuin vooraan hebben of achteraan. De voortuin familie bevat 5 typehuizen. Sommige types hebben ook subtypes die verschillen op basis van de details van het grondplan. Elk subtype heeft dan een reeks variaties in functie van het aantal slaapkamers die gaan van 1 tot 5 enz.

1.3.3. *Duarte – Order & Diversity*

Het doel van Duarte's eerste thesis, als voorafgaande studie voor de 'shape grammar' van Siza, was om orde en diversiteit te verkrijgen in het systematisch ontwerp van straatgevels binnen een modulair systeem van wonen. (Duarte J. P., Order and Diversity Within a Modular Search for Housing: A Computational Approach, 1993)

De experimenten die hij gebruikt zijn boeiend voor het vervolg van dit onderzoek. De experimenten zijn zo ontwikkeld met als doel: het ontdekken van de beperkingen van de ontwerper en om diversiteit te genereren alsook de perceptie hierop. Deze experimenten gebruiken een computer die het ontwerpproces van de testpersonen volgt. Samen met het luidop vertellen wat gedaan wordt, wordt op deze manier een opeenvolging van regels geregistreerd die het ontwerp beschrijven. Vervolgens wordt op basis van een modulair systeem en op basis van deze experimenteel vastgelegde regels een 'shape grammar' ontwikkeld.

1.3.4. *Detentiehuis regels*

Het proberen vertalen van een ontwerp naar regels is niet eenvoudig. Daarom is het wel handig om zoals bij Duarte het ontwerp proberen luidop te beschrijven en alles neer te schrijven. Het is belangrijk om, met het oog op een parametrisch model, met zoveel mogelijk variabelen rekening te houden, aangezien deze de graad van aanpasbaarheid van het model zullen bevorderen. In wat volgt is een opsomming van een aantal regels die het ontwerp zouden moeten beschrijven. Niet elke regel is makkelijk toepasbaar. De regels die parametrisch vertaald kunnen worden staan in vet gedrukt. Uit paragraaf 2 van dit hoofdstuk zal blijken dat het ontwerp van deze case al te complex is om op te bouwen als parametrisch model volgens enkel deze regels:

- site opdelen loodrecht op de langse richting in parallele zone's in functie van de randen = verschillende buitenruimtes.
- parallelle randen van deze zone's zijn volumes die kunnen worden ingevuld met gebouwen.
- **tussenafstanden tussen de woonvolumes niet te groot (tussenliggende gangen) maar groot genoeg (inkijk/uitzicht) vertrekkende van 1 volume gedefinieerd door de context.** (te groot en groot genoeg zal nog meer specifiek gedefinieerd moeten worden om in het parametrische model in te geven).

- **buitenruimtes lopen zoveel mogelijk fysiek door en zo weinig mogelijk visueel = niet al te grote openingen; maar ruim genoeg als overdekte rookplaats of terras**
- gebouwen nemen dus op gelijkvloers niet de volledige oppervlakte in, op de verdiepingen erboven wel
- **bij de keuken hoort een overdekte buitenruimte die groot genoeg is (binnen het volume)**

- tussen de volumes lopen onderling 2 gangen (gelijkvloers en 1e verdieping); eventueel maar 1
- **als de tussenafstand te groot is loopt er geen gang (tussen woonvolumes dus altijd een gang)**

- doodlopende straten geven een ruimte met dubbel gebruik op de kop op gelijkvloers en de gang ertegen (ruimte is uitbreidbaar met de gang)
- op de 1e verdieping loopt de gang aan de kant van de doodlopende straat om een gevel te krijgen

- de verticale circulatie zit aan de rand van een volume
- verticale circulatie in de kopgebouwen en in 1 van de woonvolumes

- ondergrondse parking onder de volledige site voor bewoners en apart deel voor bezoek en personeel
- ook trap aan een doodlopende straat

Het is duidelijk dat deze regels vanuit het specifieke ontwerp van de case komen, maar moeilijk te vertalen zijn naar computertaal. Er wordt hier zelfs nog niet gesproken over de relaties tussen de ruimtes, wat een nog complexer schema zou opleveren. Het is hier ook zichtbaar dat het vertalen in regels van een meer complex ontwerp veel moeilijker is en een gigantische grammar zou opleveren als dit ontwerp vergeleken wordt met de redelijk eenvoudige rechthoekige huizen van Siza in Malagueira (Duarte J. P., Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses, 2001).

1.4. Generatief Ontwerpen

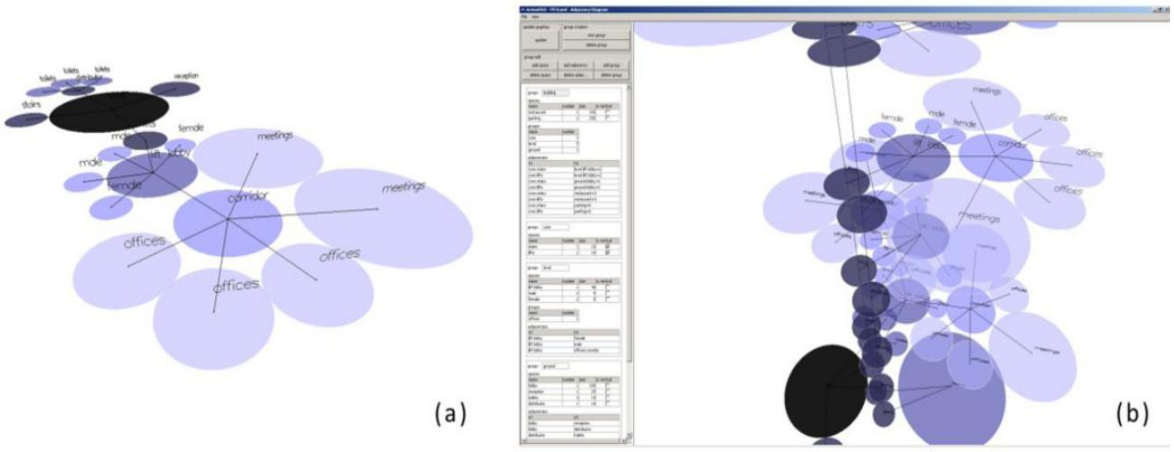
Generatief ontwerpen is een ontwerpmethode waarvan het resultaat gegenereerd wordt door een set van regels of een algoritme, meestal door middel van een computerprogramma. De meeste generatieve ontwerpen zijn gebaseerd op parametrisch modelleren. Dit is een snelle methode om alle mogelijke ontwerpoplossingen te ontdekken en te visualiseren en gebruikt wordt in verschillende velden: naast architectuur, techniek en bouw (AEC) ook in kunst, communicatie en productontwikkeling. (Maeda, 2001)

1.4.1. *Voorbeelden*

In wat volgt worden enkele type voorbeelden aangehaald om de mogelijkheden van generatief modelleren duidelijk te maken in het kader van de thema's van dit onderzoek: namelijk: 'layout-problems', 'floorplanning' en 'packing'. Deze voorbeelden zijn gebaseerd op cases van het bureau AEDAS R&D die naast het ontwerpen van gebouwen ook volop onderzoek voeren in deze vernieuwende onderzoeksvelden. (HELME, DERIX, & GAMLESÆTER, 2012)

Een eerste case is het topologische model: 'automatische bubbel diagram'. Dit model helpt de ontwerper met het visualiseren en het werken met de configurationele structuur van het gebouw al van in een vroeg stadium in het ontwerpproces. Het doel is om de topologische doelstellingen te bereiken, zonder vast te hangen aan geometrische beperkingen.

De ruimtes worden gerepresenteerd door cirkels waarvan de oppervlakte overeenkomt met de oppervlakte van de bedoelde ruimte om al een eerste idee te krijgen van groottes. De relaties worden weergegeven door lijnen die de cirkels verbinden. Er is in deze applicatie een directe manipulatie mogelijk door cirkels te verslepen en anders te verbinden. Ook is er een diepteanalyse gekoppeld aan dit model: namelijk hoe dieper een ruimte zich in het gebouw bevindt, hoe donkerder de cirkel gekleurd zal zijn.

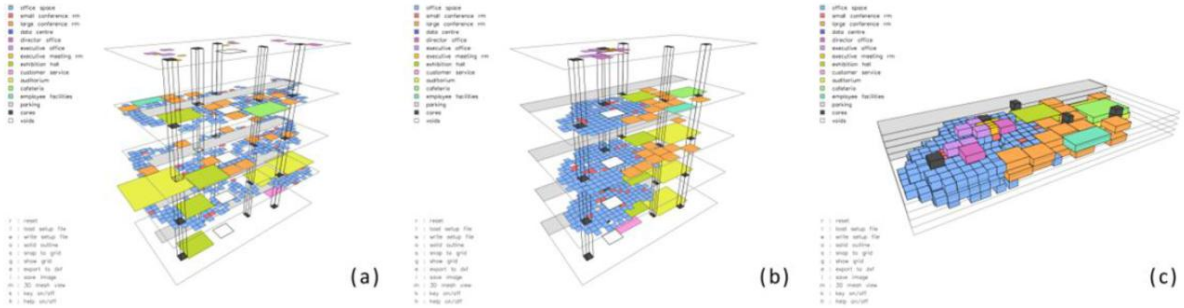


Figuur 14 - Case 1: bubbel diagram

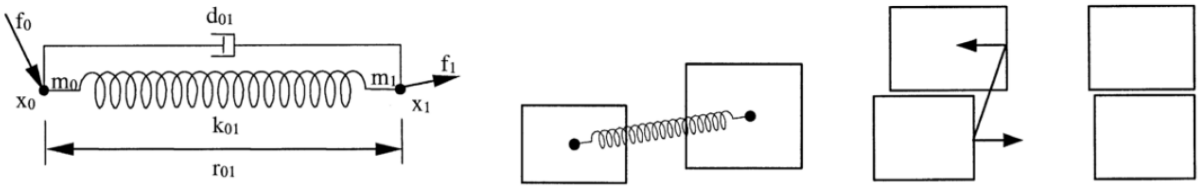
Een tweede case is een model waar de geometrie en de massa van elk volume in het gebouw van bij het begin van het ontwerpproces een rol speelt: ‘massing approximations’. Een grote beperking van de voorgaande case is dat het moeilijk is om te peilen naar de oppervlakte. Een belangrijke dimensionele beperking is de beperking van de vorm en grootte van de vloerplaat en het is al snel nodig om een schatting te kunnen maken van het beschikbare vloerpercentage dat bezet zal worden in de simulatie.

De ontwerper kan de verlangde relaties opleggen door gebruik te maken van een nabijheidmatrix. Dit model representeert deze relaties door middel van een aantrekkingskracht die wordt gegenereerd tussen de 2 ruimtes. De ruimtes dienen ook binnen de randen van de vloerplaat te blijven. De circulatie wordt weergegeven door verticale ruimtes die manueel verplaatst kunnen worden binnen het gebouw. Dit is niet automatisch maar er kunnen wel andere ruimtes ten opzichte van deze verticale kokers aangetrokken worden. Alle andere circulatie wordt voorzien door bij elke ruimte 20% meer ruimte te voorzien om hier van bij het begin niet specifiek rekening mee te hoeven houden.

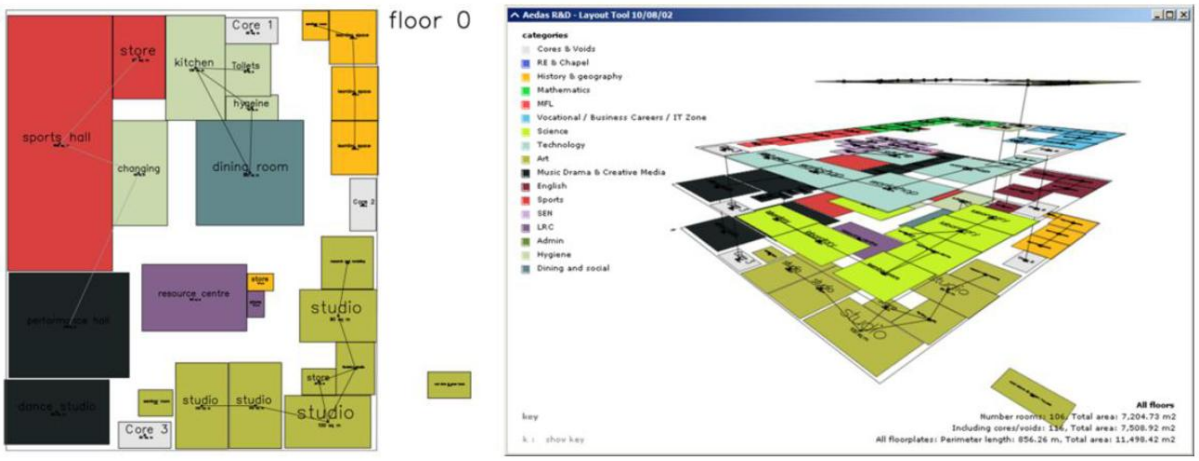
Omdat zowel de dimensies als de relaties worden ingeladen vanuit een ander bestand is de interactie van de gebruiker met dit model indirect. Enkel de plaatsing en manipulatie kan direct in het model gebeuren.



Figuur 15 - Case 2: 'massing approximations'

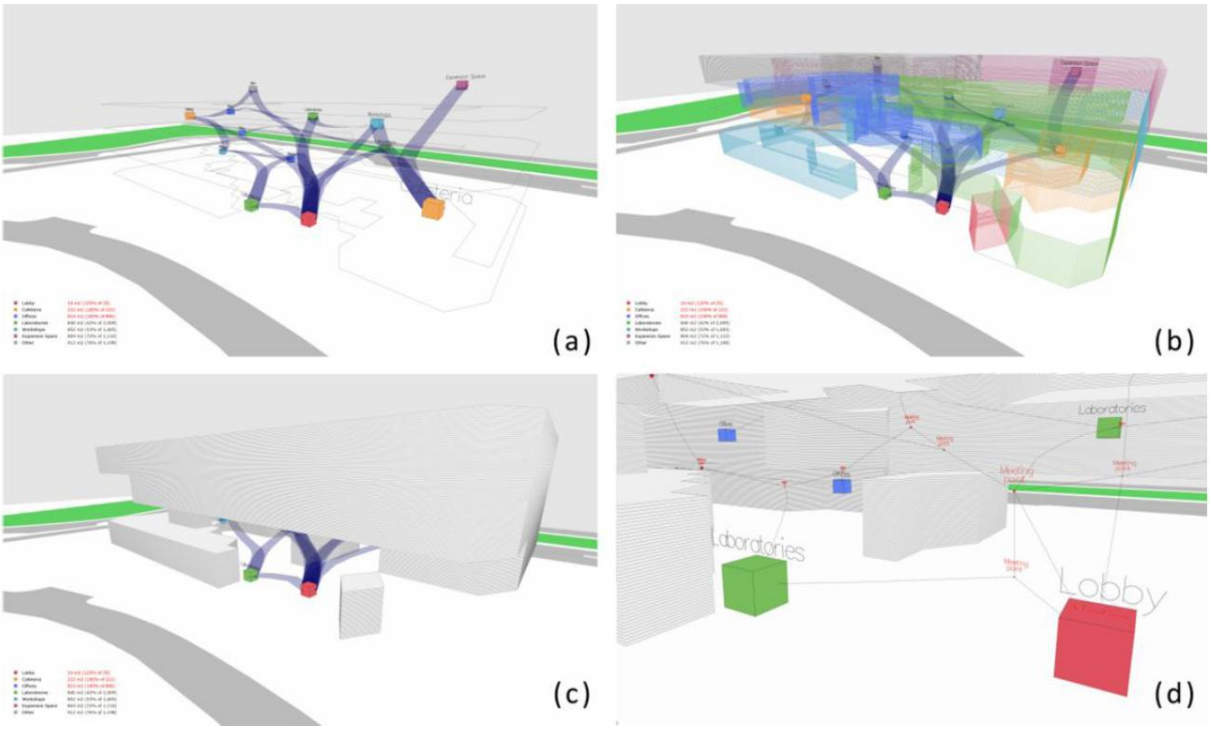


Figuur 16 - Aantrekkingskracht principe



Figuur 17 - Aantrekkingskracht tussen ruimtes

Een derde case is het model dat gestuurd wordt door de beweging doorheen het gebouw. In voorgaande cases werd het principe van circulatie vermeden in het model. In deze case wordt er geëxperimenteerd met hoe de configuratie van ruimtes beïnvloed kan worden door de beweging van mensen. Dit model wordt gestuurd door het plaatsen van knooppunten in elke ruimte Het is aan de ontwerper om manueel patronen te slepen tussen deze knooppunten (in de uiteindelijke applicatie zou het de bedoeling kunnen zijn om via 'virtual reality' de gebruiker te laten rondlopen in het gebouw). Ze worden dus niet automatisch gegenereerd. De routes die daarbij gecreëerd worden, moeten niet gezien worden als letterlijke circulatiewegen maar eerder als uitgesneden volumes waar een circulatie mogelijk is binnen het mogelijk gebouwde volume.



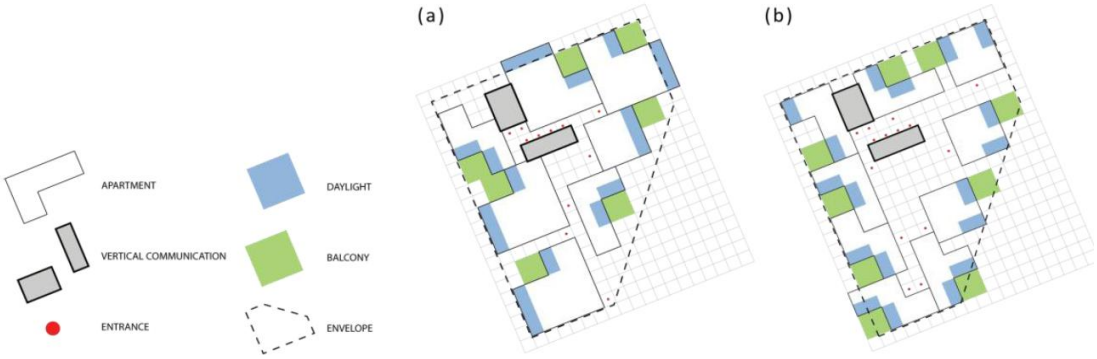
Figuur 18 - Case 3: 'movement driven'

Een vierde case is het ‘packing model’. Er zijn situaties waarop de vorige modellen met aantrekkingskracht niet toepasbaar zijn. Dit is het geval bij een ruimtelijk lay-outprobleem waar het de bedoeling is om volumes of woonunits te stapelen in een toren.

Een eerste voorbeeld hiervan is door appartementen te stapelen met een 3-dimensionele configuratie. Om de verschillende mogelijkheden van appartementen te modelleren is een computermodel nodig omdat het bijna onmogelijk is om alle mogelijke combinaties manueel te genereren. Het concept is om een cilindrische toren op te bouwen bestaande uit de verschillende appartementen die ‘polyominoe’ structuren hebben. Dit zijn aaneenschakelingen van units zoals bijvoorbeeld bij tetris waar alle blokjes uit alle mogelijke 2-dimensionele configuraties van 4 units (tetrominoes) bestaan. Op dezelfde manier als bij dit spel is het de bedoeling om alle appartementen zo dicht mogelijk op elkaar te stapelen. Een evolutionair algoritme wordt gebruikt om een goede oplossing te bekomen (dit is geen optimale oplossing, maar wel een goede).

Figuur 19 - Case 4: 'close packing' - voorbeeld 1

Een tweede voorbeeld is een residentiële hoogbouw, eenvoudiger dan vorig voorbeeld omdat de appartementen allemaal binnen hetzelfde niveau liggen maar moeilijker voor de ontwerpcriteria (en dus ook een ingewikkeldere fitness curve). De input zijn 13 verschillende configuraties van appartementen. Het is de bedoeling om ze allemaal te plaatsen op een bestaand grondplan met als doel het optimaliseren van bereikbaarheid, daglichttoetreding door de ramen en het plaatsen van balkons aan de buitenrand van het gebouw.



Figuur 20 - Case4: 'close pakcing' - voorbeeld 2

Beide voorbeelden hebben een indirecte interactie met de gebruiker aangezien alles op voorhand ingegeven moet worden. Dan wordt er op een knop geduwd en dan wordt een ‘goede’ oplossing weergegeven.

Een conclusie hieruit is dat een applicatie steeds zo eenvoudig mogelijk moet worden opgebouwd om het gemak en de transparantie ervan te bevorderen. Ook is hierbij een directe interactie wenselijk en door het veranderen van een input is het ook aan te raden om alles steeds in ‘real-time’ te laten veranderen zodat duidelijk wordt wat er gebeurt als je welke beperking verandert.

1.4.2. *Grasshopper*

Voor dit meer praktische deel van het onderzoek is ervoor gekozen om in grasshopper3d te werken (plug-in voor Rhino 3-D). Dit laat eenvoudig toe om zowel wiskundig als geometrisch bepaalde voorwaarden aan een model op te leggen. Ook is dit programma zeer gebruiksvriendelijk en de ‘real-time’ verandering wanneer er 1 parameter gewijzigd wordt is wenselijk. Het laat eender wie toe om te experimenteren met generatief ontwerp zonder enige voorkennis van programmeren of ‘scripting’. Het is wel al gebleken dat met enkel de basiscomponenten er een beperking ervaren kan worden en het wel nodig is om een eenvoudige programmeertaal aan te leren om zelf enkele ‘custom’ componenten te schrijven. Hiervoor is gekozen om in de C# omgeving te werken. Dit is een gebruiksvriendelijke taal die heel ‘straightforward’ is in zijn commando’s. Ook werd deze taal al gebruikt in vorig hoofdstuk voor het schrijven van de kleine applicatie om de nabijheidmatrix uit een Revit model af te leiden.

2. Ontwerp

In deze paragraaf worden de verschillende stappen besproken om tot het uiteindelijk ontwerponderzoek te komen dat meer in het bijzonder in hoofdstuk 4 besproken zal worden en waar in hoofdstuk 5 het ontwikkelen van een applicatieprototype op zal volgen. Deze applicatie zal als input enkel de relaties tussen verschillende zelfgekozen ruimtes hebben en als output een mogelijk ontwerp.

Het is belangrijk hier nogmaals te onderstrepen dat een ontwerp niet als een strak afgelijnd probleem kan aanzien worden. Een ‘designtask’ is een ‘Ill Structured Problem’ (ISP) en dus geen ‘Well Structured Problem’ (WSP) wat volgens de literatuur nodig is om de computer te kunnen inzetten. Bij het onderzoek van deze thesis moet dus steeds onthouden worden dat het resultaat van elk model bij de terugkoppeling naar het effectieve ontwerp zeer kritisch moet worden bekeken en gebruikt. Het is niet de bedoeling dat deze modellen voor een letterlijke overname in het ontwerp worden gebruikt.

2.1. Model 1: Contextafhankelijkheid

Het belang van een aanpasbaar ontwerp in een variabele context kan een uitgangspunt zijn om dit vraagstuk te benaderen. In het eerste idee zouden de regels van het detentiehuis zoals ze beschreven werden in vorige paragraaf, met ook de mogelijkheid om bepaalde gebruiksafhankelijke aanpassingen te doen, vertaald worden naar een parametrisch model in grasshopper3d dat zich in functie van context en omgeving zou aanpassen aan de gegeven site.

2.1.1. *Context*

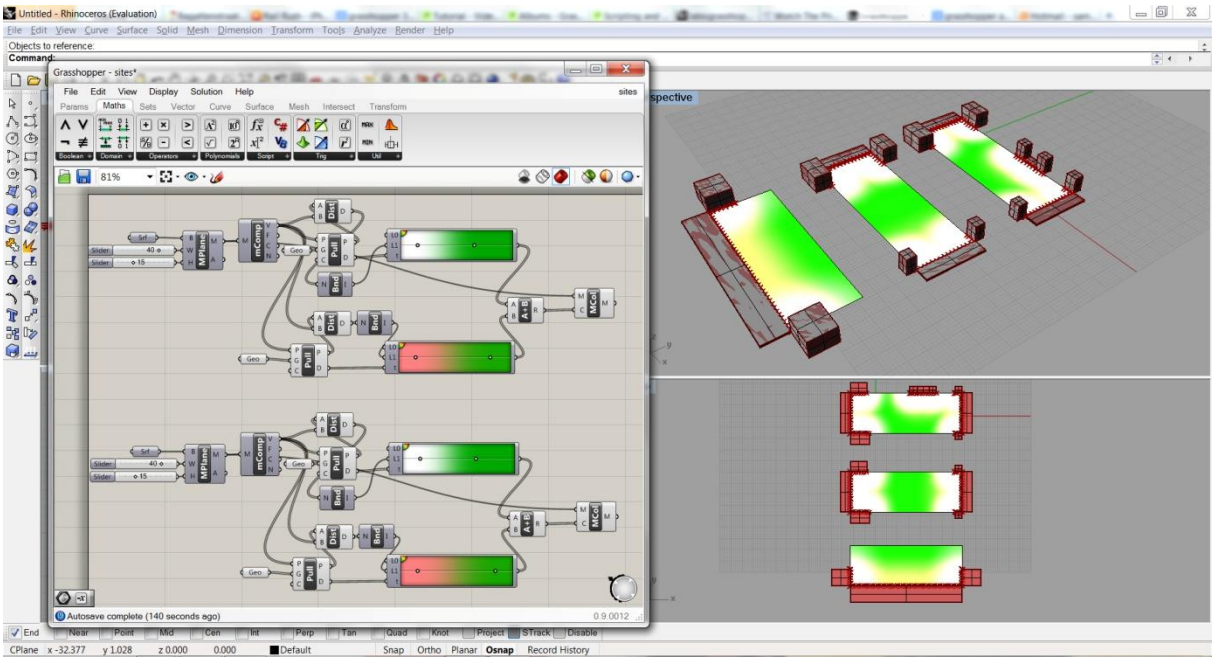
Het doel van het ontwerp van de case van het detentiehuis was om verkrotting in binnengebieden van stedelijke bouwblokken tegen te gaan en deze gebieden te opwaarderen. Als gevolg zal elke site een heel specifieke, deels bebouwde rand hebben met ook bepaalde straten die uitgeven op, of raken aan het preceel. Deze rakende gebouwen en straten zijn dus een eerste input voor het model. Hiervan uitgaand werd een eerste model opgebouwd in grasshopper3d.

Op onderstaande afbeelding kan gezien worden hoe de site verkleurt in functie van de omliggende gebouwen en de rakende straten. De groene zone stelt een gebied voor dat verder weg ligt van straten en gebouwen, waar dus een effectieve groenzone of buitenruimte geplaatst kan worden. Dit om al een eerste indruk te krijgen waar het zwaartepunt van de buitenruimtes zich zal bevinden.

De meer witte bevinden zich naast de aanliggende gebouwen, aangezien daar ook meestal bebouwing zal zijn op de site om het stedelijke kader te vervolledigen.

De gele zones zijn de gebieden die aan straten grenzen. Hier wordt ook ingezet op gebouwen om het stedelijk kader te vervolledigen. In principe zou dit ook net niet kunnen: er zou een opening gecreëerd kunnen worden naar de straat toe, maar voor het geval van de zwaarste detentie zoals in deze case wordt behandeld lijkt dit niet aangewezen. Ook zijn deze zones die raken aan de straat geschikt voor dubbel gebruik (zowel voor gedetineerden als voor omwonenden). Dit zoals besproken in eerder hoofdstuk om het gebouw meer aanvaardbaar te maken voor de omwonenden en om de gedetineerden een soort interactie met de buurt te kunnen laten aangaan.

Het bovenste model is deze van de case, waarop in deze vereenvoudigde, rechthoekige site ook een doodlopende straat uitkomt in het midden van de site.



Figuur 21 - Kleurgradiënt van de sites

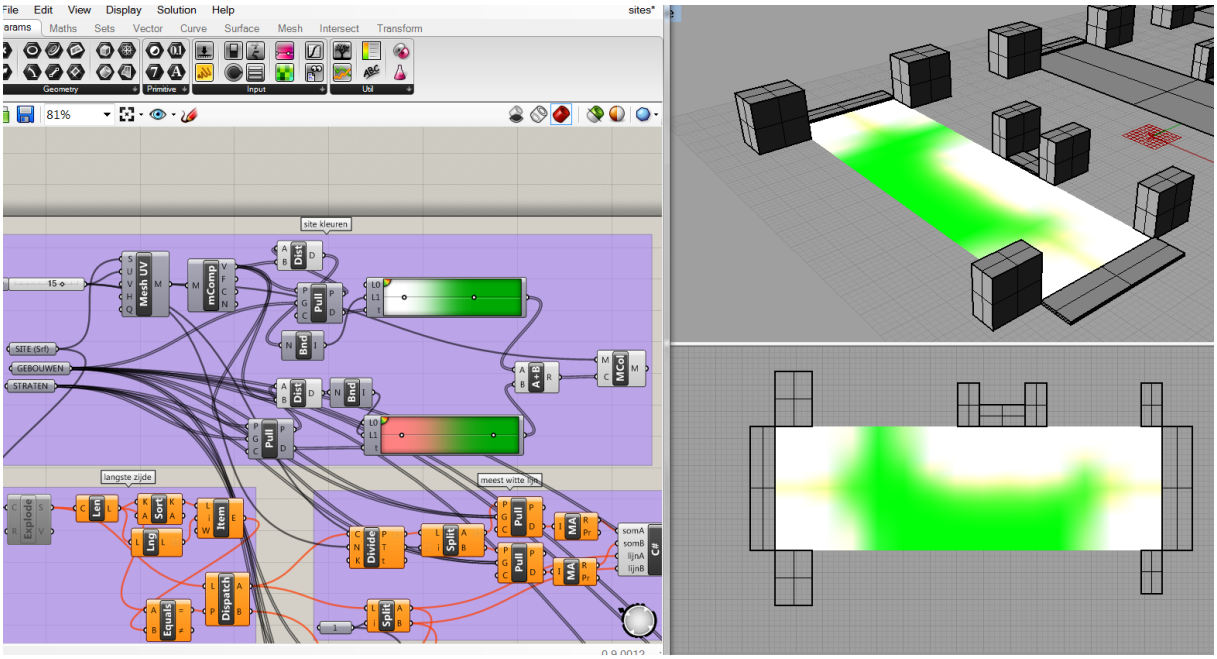
2.1.2. *Model van het gebouw (case)*

Het ontwikkelen van het model van het detentiehuis in grasshopper3d wordt in verschillende stappen opgebouwd. Onderstaande afbeeldingen duiden het modelleerproces om uiteindelijk te komen tot een eerste idee van wat het detentiehuis kan voorstellen met enkele simpele regels en aanpasbaarheden.

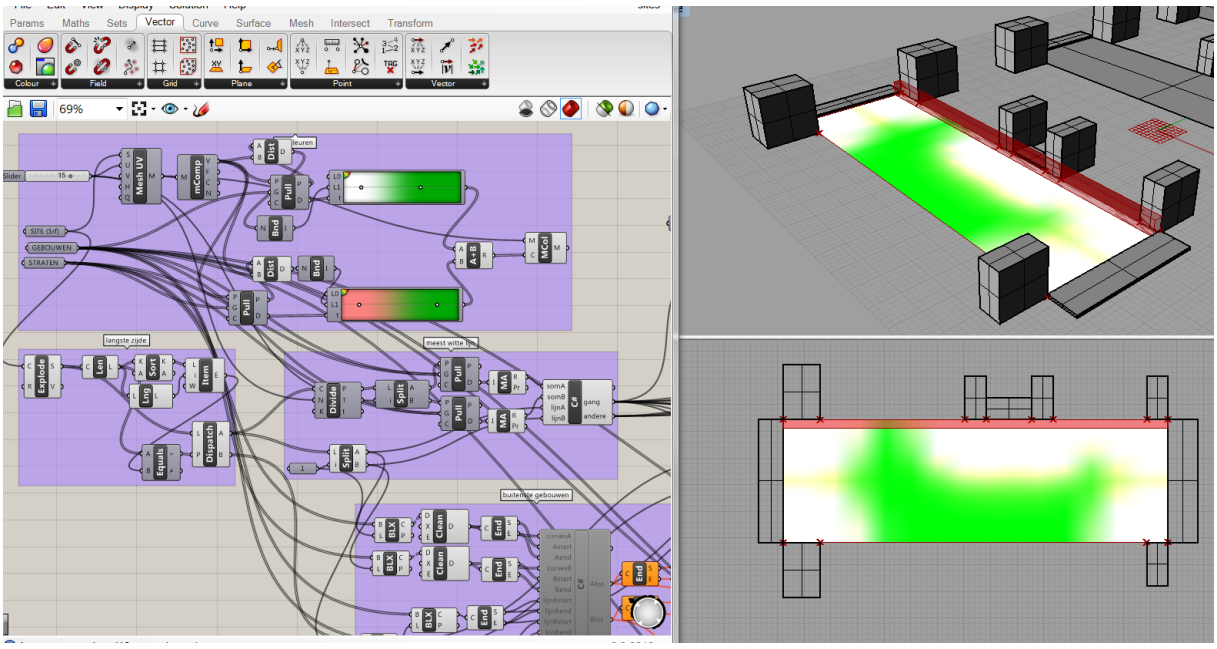
Stap 1: nadat de site reeds geanalyseerd werd en ingekleurd door het model, is het eerste doel om op de site de locatie van de gang te plaatsen. De gang is een volume dat doorloopt over de volledige site. Dit kan als effectieve gang zijn, maar evengoed als doorlopende entiteit in bepaalde andere ruimtes en volumes. Deze gang wordt aan de meest witte kant op de site geplaatst, dit is de zone waar het meeste bebouwing of een straat nabij is. De gang kan door de gebruiker aangepast worden in breedte, met een minimum breedte van 1,6 meter en een maximum breedte van 3 meter.

Stap 2 is het plaatsen van de buitenste gebouwen om het stedelijk kader te vervolledigen. Deze gebouwen volgen in hoogte en breedte de naastliggende gebouwen om zo goed mogelijk in te passen in het weefsel. Hier ligt de aanpasbaarheid in het wijzigen van de diepte van de gebouwen: dieper dan context; maar dit zou later in functie van de nodige ruimtes binnenin kunnen gebeuren.

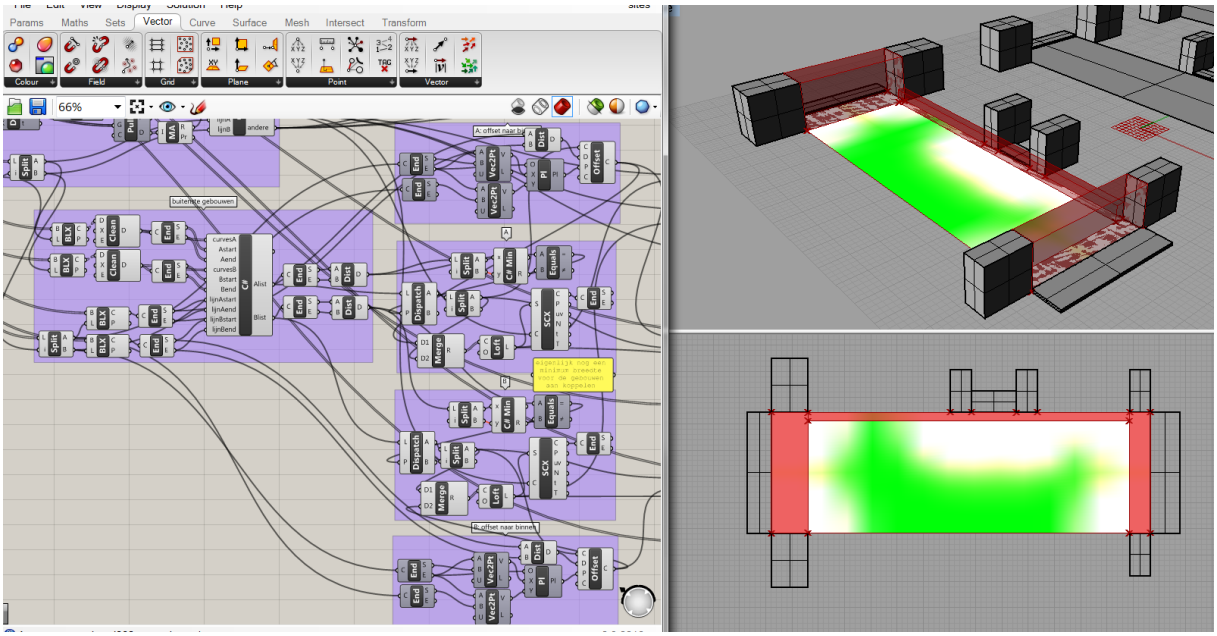
Stap 3 is het plaatsen van de tussenliggende gebouwen. Deze worden geplaatst aansluitend op de gang met een tussenruimte die in grootte aanpasbaar is, maar steeds groot genoeg moet blijven. Deze tussenruimte heeft ook een beperking in grootte, zodat als de tussenruimte te groot wordt het aantal gebouwen zal toenemen. Hier worden ook de gebouwen aanpasbaar in breedte met een minimale waarde zodat deze nuttig kan blijven voor de ruimtes binnenin.



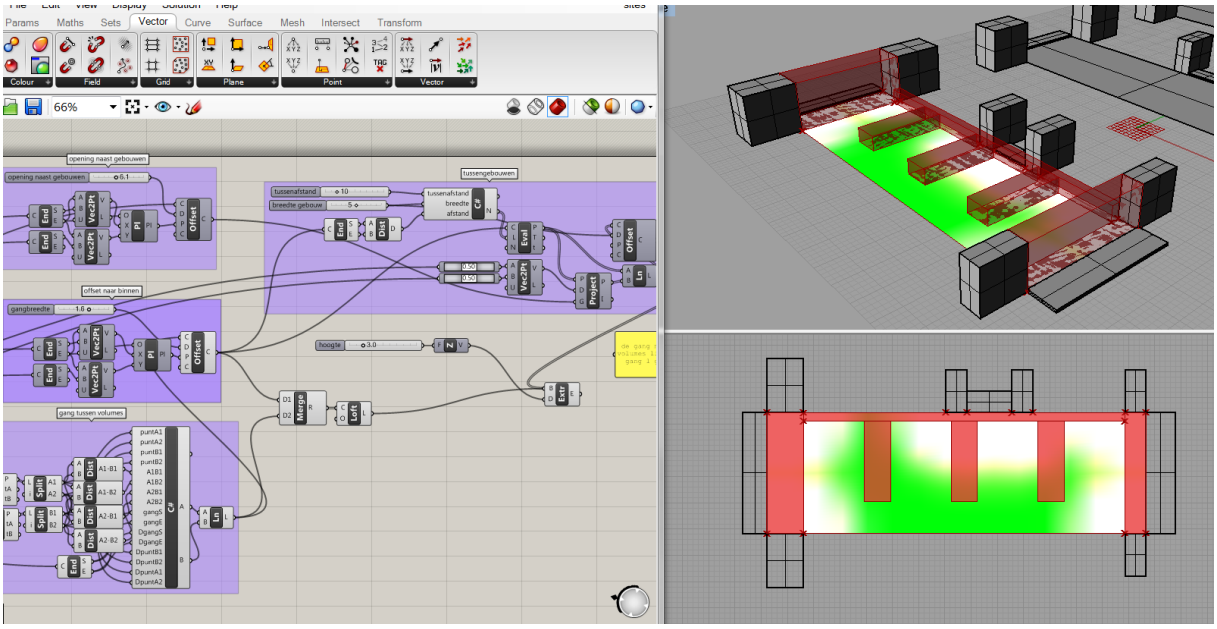
Figuur 22 - Stap 0: site



Figuur 23 - Stap 1: gang



Figuur 24 - Stap 2: randgebouwen

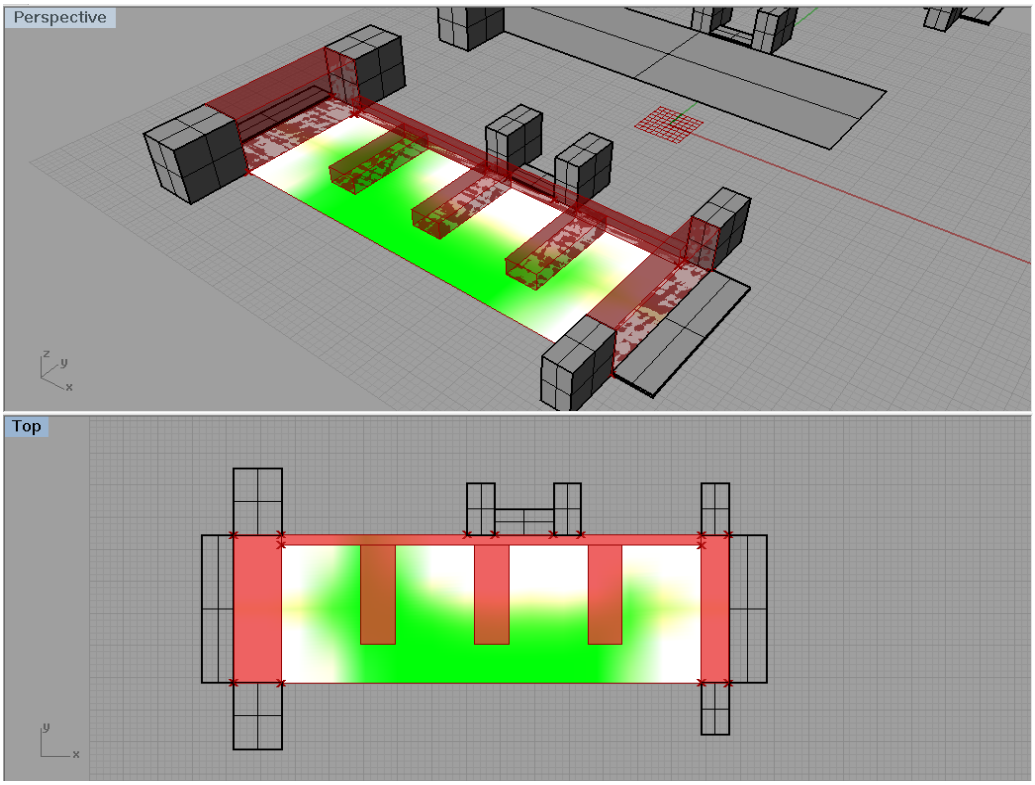


Figuur 25 - Stap 3: tussenliggende gebouwen

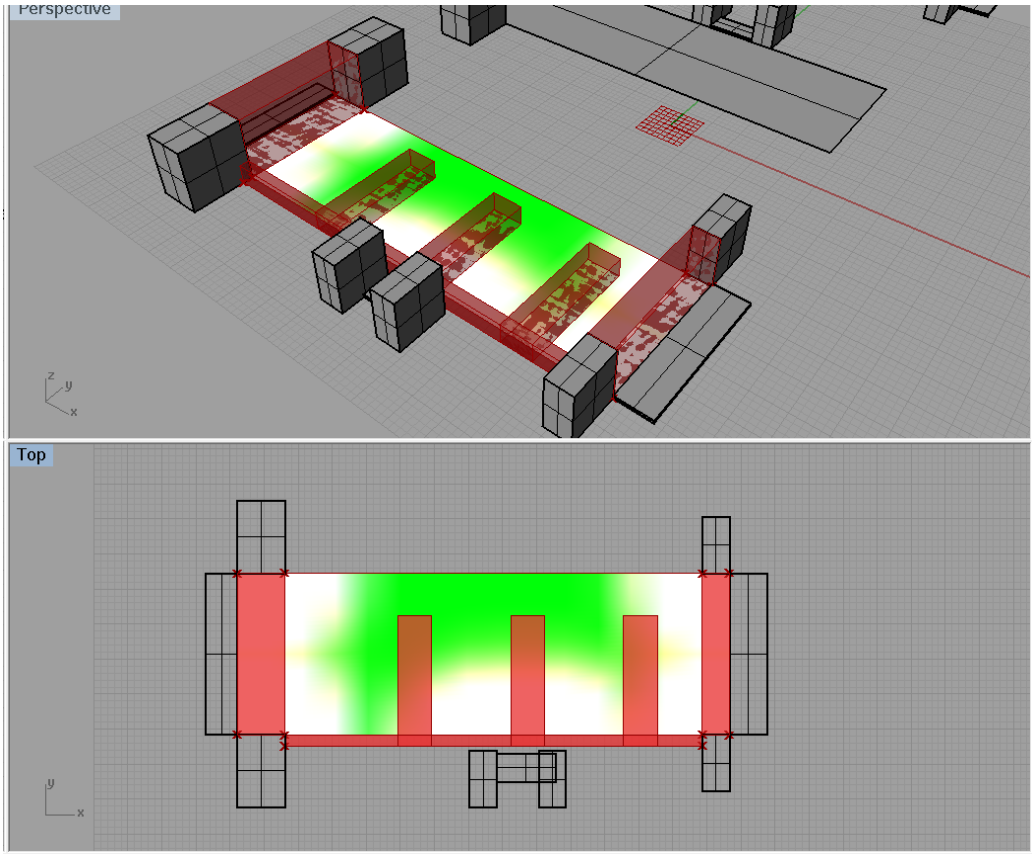
2.1.3. *Model in verschillende sites*

Dit model wordt vervolgens toegepast op verschillende sites om de contextafhankelijkheid weer te geven. Het zal blijken dat dit model werkt voor bepaalde contexten, maar dan voor heel andere contexten niet meer desondanks het al zeer complexe model om slechts deze simpele volumes weer te geven (zie besluit).

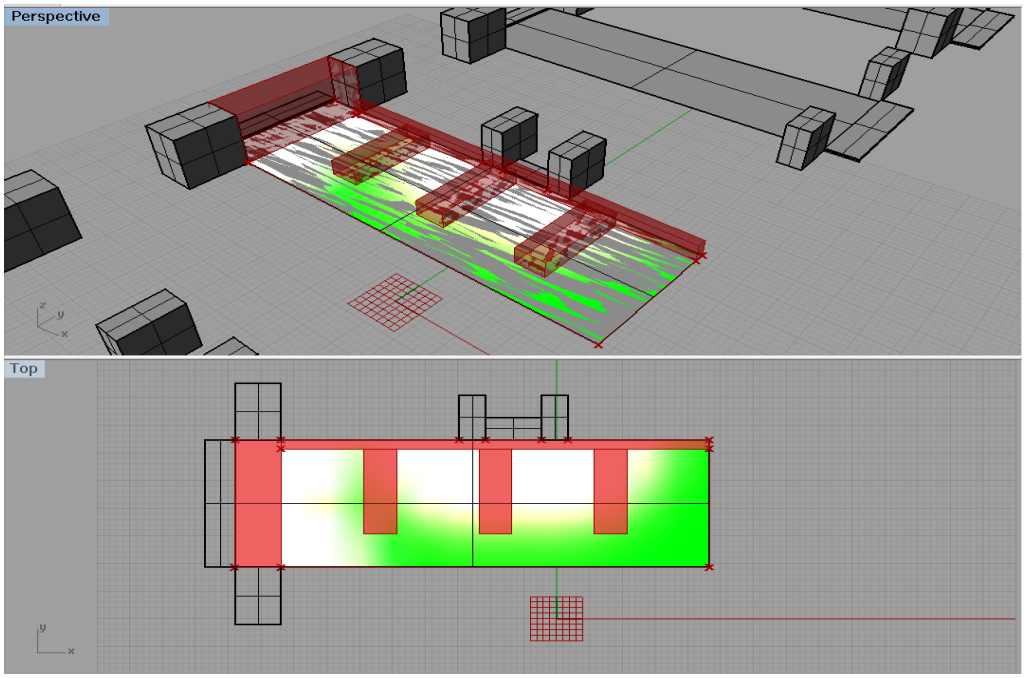
Onderstaande afbeeldingen geven de testen weer van de verschillende contexten. Elke nieuwe afbeelding heeft een andere context, dit is dus andere rakende gebouwen aan het perceel en/of andere rakende straten op andere locaties.



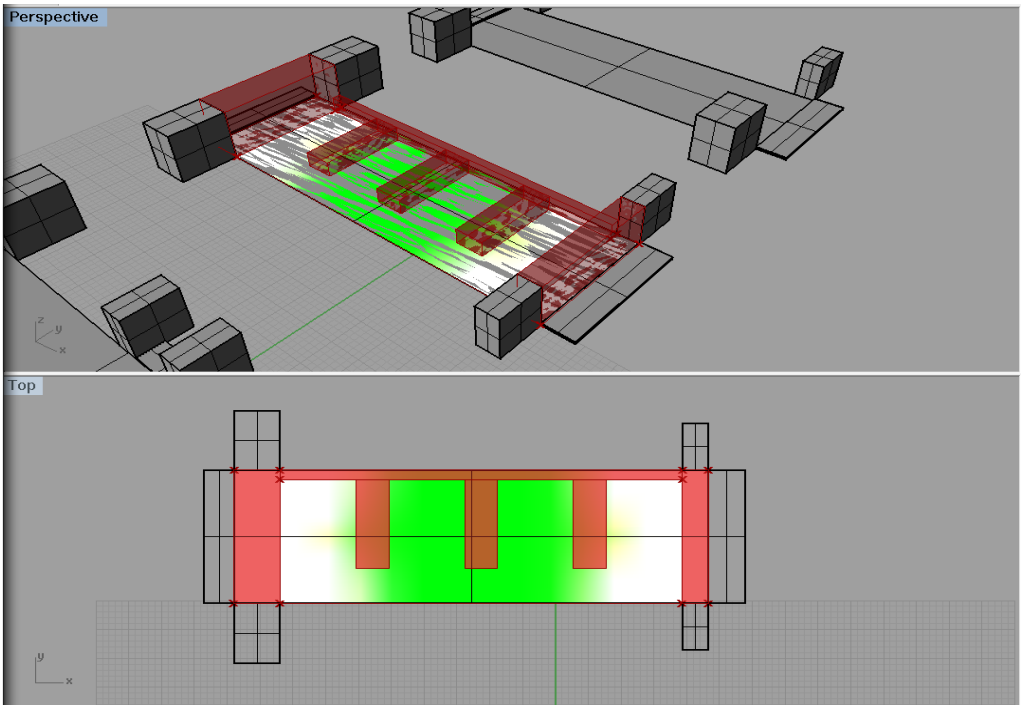
Figuur 26 - Context test 1



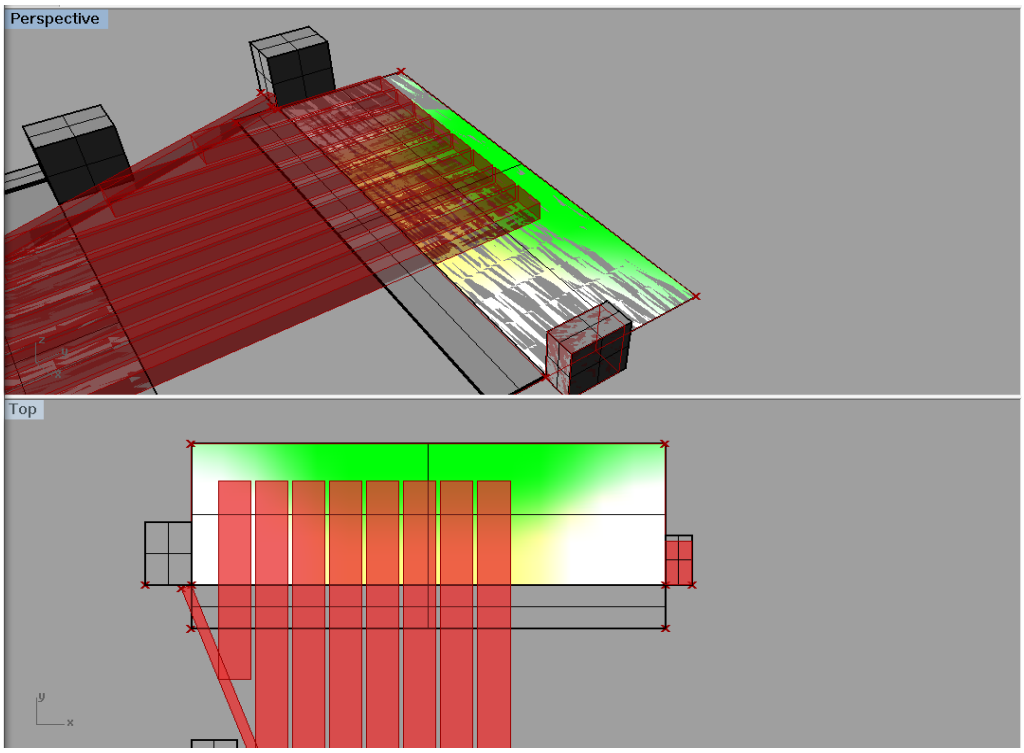
Figuur 27 - Context test 2



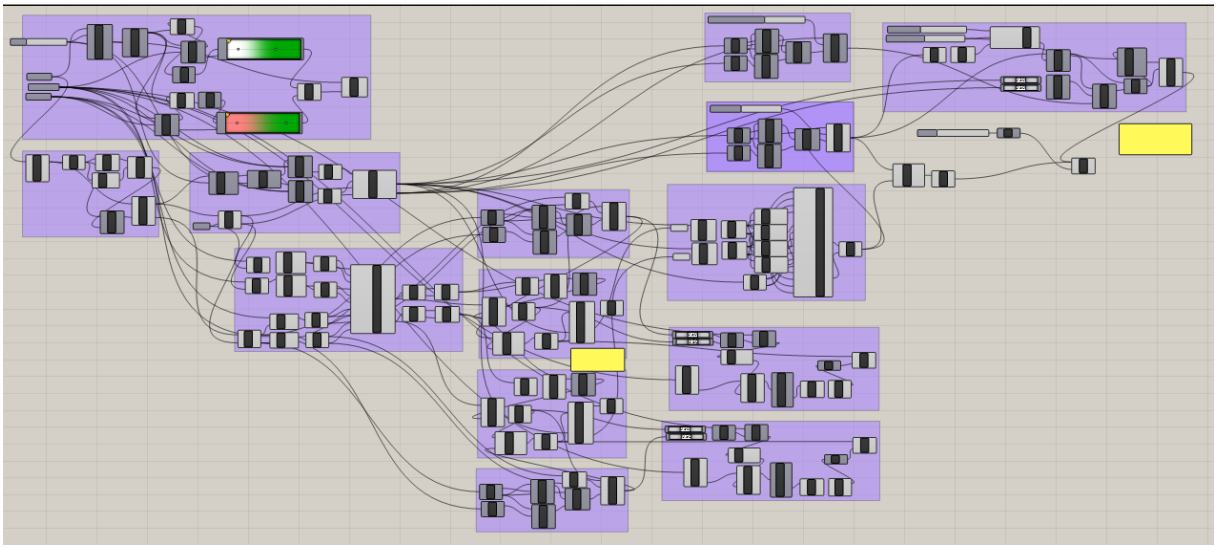
Figuur 28 - Context test 3



Figuur 29 - Context test 4



Figuur 30 - Context test 5



Figuur 31 - Complexiteit van het model

2.1.4. *Besluit*

In de laatste bovenstaande afbeelding is de complexiteit van het model voor 1 context zichtbaar en toch voldoet dit model niet aan de verwachtingen. Het probleem is dat het model enerzijds niet heel aanpasbaar is en dat het gebouw er quasi altijd op dezelfde manier zal uitzien door te vertrekken van de specifieke regels van het case model. Anderzijds is het moeilijk om vanuit de grote volumes te vertrekken en nadien pas de binnenvolumes variabel te maken.

Er zal moeten vertrokken worden van de kleinste korrel, vanuit de ruimtes. Die ruimtes in een bepaalde configuratie, met bepaalde vooropgestelde relaties, vormen dan een volume. Het vervolg van dit onderzoek werkt met het idee om alle mogelijke ruimtelijke configuraties die verschillende volumes genereren te ontwikkelen en dan de volumes af te toetsen volgens de gegeven contexten. Hieruit zullen dan de aanvaardbare configuraties overblijven en zodoende krijg je heel verschillende en verrassende volumes op de gegeven site.

Dit is het uitgangspunt van de rest van dit onderzoek. In wat volgt zal in dit hoofdstuk verder gekeken worden hoe alle verschillende 3-dimensionele ruimtelijke configuraties van een aantal ruimtes gegenereerd kunnen worden door middel van geometrische functies zoals ze in grasshopper3d gegeven zijn.

2.2. Model 2: Eenvoudige kubus

Het vertrekpunt van het verdere onderzoek is de kubus. De eerste vraag is in hoeveel mogelijke manieren je dit volume kan opdelen in 4 verschillende ruimtes, als de 2 snijvlakken vastliggen centraal volgens de X-as en de Y-as.

2.2.1. *Model*

Stap 1 is om een kubus (eigenlijk een balk) te genereren. Dit volume heeft variabele afmetingen die door de gebruiker ingesteld kunnen worden. Het idee is om dit volume als een kleine woonunit met 4 ruimtes voor te stellen. De afmetingen hebben een begrensd minimum en maximum zodat er realistische ruimtes kunnen ontstaan. De hoogte zal in functie van een veelvoud van ongeveer 3 meter zijn. Dit is enkel om het model al op een realistische schaal te kunnen bekijken en niet te vervallen in pure abstractie.

Er wordt nog steeds in grasshopper3d gewerkt om geometrische functies te kunnen gebruiken en ook steeds een directe feedback te krijgen over de representatie in de ruimte.

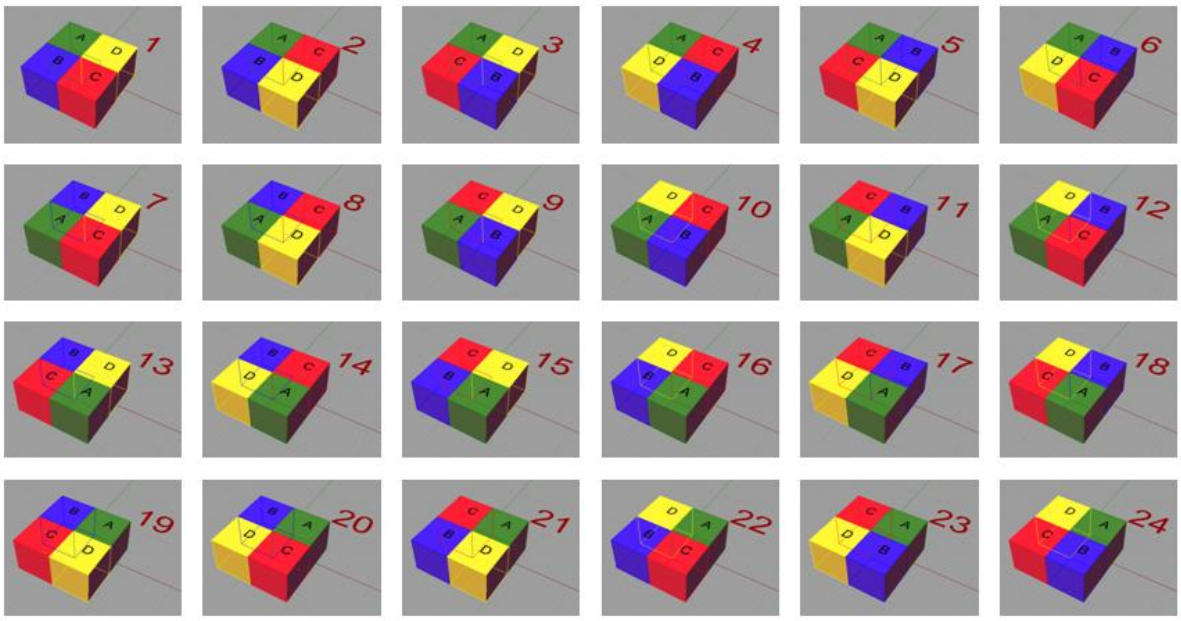
Stap 2 bestaat uit het doorsnijden van dit volume in 4 verschillende volumes. Dit zijn geometrische doorsnijdingen. Op termijn, als het model complexer zal worden zal dit bepaalde vertragingen met zich meebrengen, aangezien steeds de geometrie moet worden herbepaald bij elke stap, waar het sneller zou kunnen om eerst alle bewerkingen te doen en pas achteraf de geometrische representatie op te roepen.

Het resultaat van stap 2 zijn 4 dozen die de 4 ruimtes representeren.

Stap 3 is het laten permuteren van deze dozen (het principe van een permutatie wordt in wat volgt verder uitgelegd). Dit wil zeggen het laten afwisselen van de ruimtes ten opzichte van elkaar en zo verschillende ruimtelijke configuraties van deze 4 ruimtes te bekomen. Dit is om alle mogelijke relaties tussen deze 4 ruimtes te simuleren. Deze permutatie geeft 24

mogelijke oplossingen. Deze 24 oplossingen geven 24 unieke configuraties en dus ook 24 unieke nabijheidmatrices die rekening houden met de positie in de ruimte (oriëntatie).

Onderstaande afbeeldingen geven voor het geval van de simpele kubus in 4 ruimtes op te delen de 24 mogelijke configuraties.



Figuur 32 - 24 permutaties van simpele kubus met 4 ruimtes

2.2.2. *C#-code van de permutatie*

Voor het permuteren in grasshopper3d werd een nieuwe component aangemaakt in C#. Deze code bevindt zich in bijlage C.

2.2.3. *Permutatie*

Een permutatie is een rangschikking van een aantal voorwerpen of getallen, dus een manier om voorwerpen of getallen in een volgorde te plaatsen.

Voorbeeld: kies 4 (r) ruimtes uit een lijst van 4 (n) ruimtes en de **volgorde** is belangrijk:

$$P = \frac{n!}{(n-r)!}$$

$$P = 4! = 24$$

Dit betekent dat het aantal mogelijke oplossingen exponentieel stijgt in functie van het aantal doorsnijdingen, of dus in functie van het aantal ruimtes in het volume, zoals zichtbaar in onderstaande tabel.

n	1	2	3	4	5	6	7	8	9
P	1	2	6	24	120	720	5040	40320	362880

2.2.4. *Besluit*

Alle mogelijkheden van relaties kunnen gerepresenteerd worden, maar het zal een doel worden om rechtstreeks de nabijheidmatrix hieraan te koppelen zodat deze informatie verder gebruikt kan worden met het basisidee van dit onderzoek steeds voorop: om de relaties als input te gebruiken. Het uiteindelijk doel is dan om alle mogelijke kubussen weer te geven en de inputmatrix met de nabijheidmatrix op elkaar af te toetsen om enkel de configuraties over te houden die voldoen aan de vooropgestelde eisen.

Deze kubus was nog zeer eenvoudig, in de volgende paragraaf wordt het plaatsen van de snijvlakken en het kiezen van het aantal snijvlakken en dus het bepalen van de indeling in ruimtes gestuurd door de gebruiker. Het zal dus mogelijk zijn om niet steeds de volledige kubus te moeten doorsnijden, wat nu nog wel het geval was. Ook is er nu de mogelijkheid om een horizontale snede te maken en dus met verschillende niveaus te werken in een al meer 3-dimensionele benadering van het onderzoek.

2.3. Model 3: Geavanceerde kubus

Dit model bouwt verder op model 2 beschreven in vorige paragraaf. Het idee blijft hetzelfde maar het model wordt uitgebreid zodat er veel meer mogelijkheden zijn om snijvlakken te plaatsen. Dit geeft een complexer model met een 3-dimensionele kubus (waar het in vorige paragraaf eigenlijk een 3-dimensionele representatie was van een 2D plan) en ook om de gebruiker een belangrijk aandeel te geven in het creëren van dit volume waar hij zelf kan kiezen waar de snijvlakken worden genomen.

Met het oog op volgend hoofdstuk, waar de geometrie in de bewerkingen zal worden weggelaten, is het nu ook al belangrijk om de nabijheidmatrix en de topologische matrix van elke mogelijke oplossing erbij te houden. Waar dit de matrixrepresentatie van het volume en dus ook de geometrie zal worden.

2.3.1. *Boom*

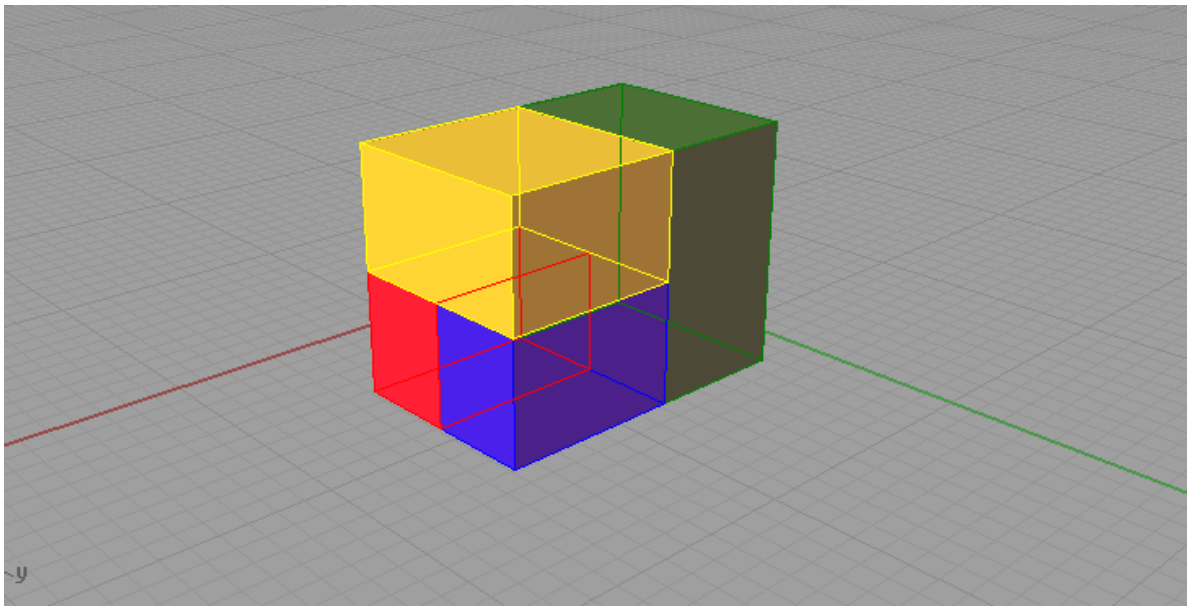
In principe wordt per snijvlak dat de gebruiker plaatst een zijtak van een structurele boom gecreëerd waar alle takken op datzelfde niveau hetzelfde snijvlak zijn maar op een iets andere plaats in het volume. Zodat naast het aantal oplossingen van de permutatie nu ook nog het aantal mogelijkheden van snijvlakken te plaatsen wordt verkregen. Op deze manier

krijg je een groot veelvoud nog eens van dit aantal permutaties. Het aantal is moeilijk te bepalen aangezien je in principe per millimeter een snijvlak kan verplaatsen en zo op een oneindig aantal mogelijkheden komt. Als voorbeeld nemen we voor dit onderzoek dat elk snijvlak per 1 meter kan verplaatst worden en dat elke ruimte minstens 2m breed en 3m hoog is voor een realistische ruimte.

Voorbeeldberekening:

Een volume (hoogte 6m breedte 5m en lengte 8m) wordt doorgesneden met 3 snijvlakken wat dus 4 ruimtes genereert.

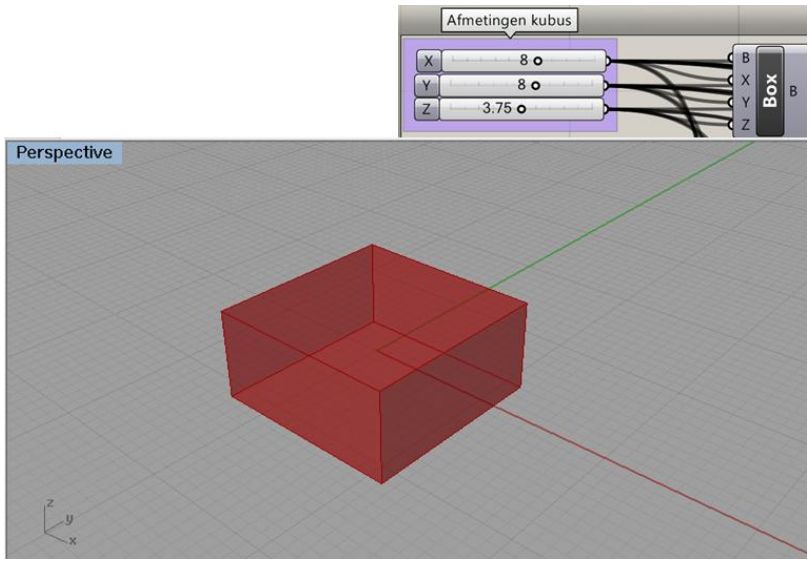
Snede 1: XZ, geeft 5 mogelijkheden om te snijden. *Snede2*: XY, geeft 1 mogelijkheid per ruimte of dus 2 mogelijkheden per deeltak om te snijden. *Snede3*: YZ, geeft 2 mogelijkheden per ruimte, of dus 6 mogelijkheden per deeltak. Dit resulteert in 60 mogelijke ruimtelijke opstellingen. Dit wordt dan nog vermenigvuldigd met de permutatie van 4 wat gelijk is aan 24 om het aantal mogelijk nabijheidmatrices te bekomen. Dit geeft 1440 mogelijkheden. Onderstaande afbeelding geeft 1 mogelijke configuratie weer.



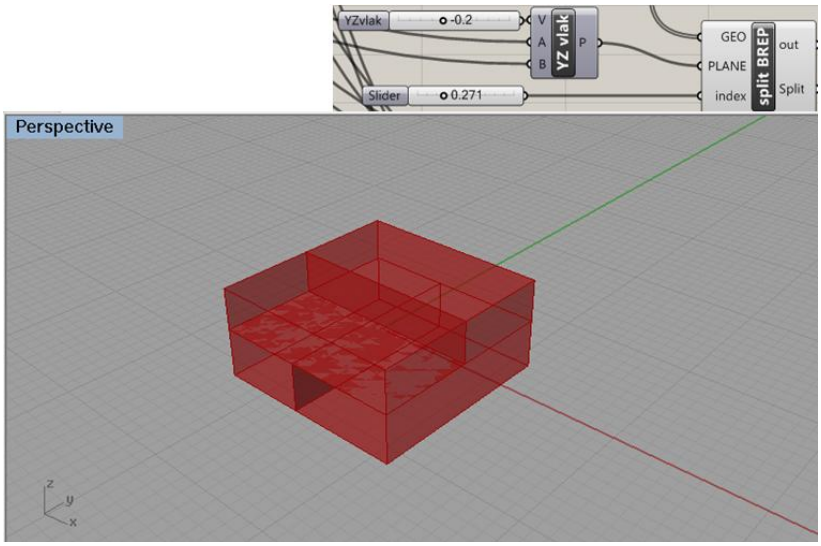
Figuur 33 - 1 van 1440 mogelijke configuraties

2.3.2. *Opbouw model*

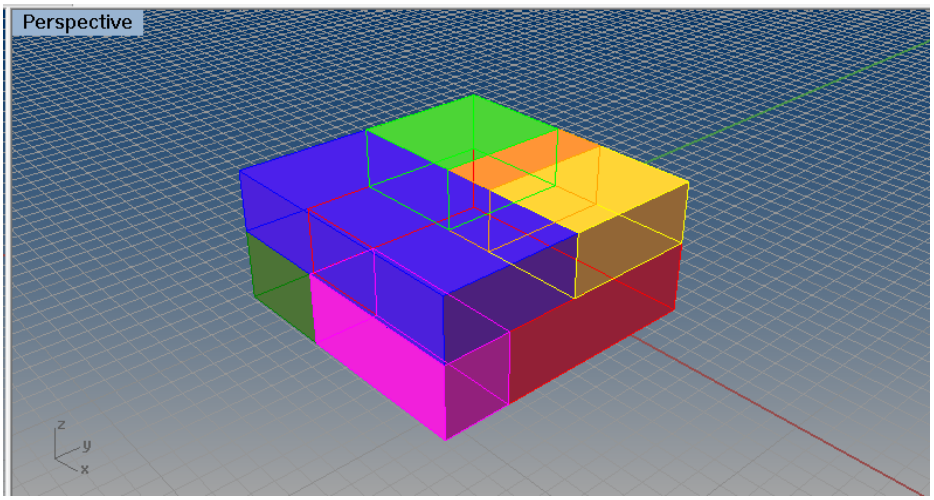
Het model is ontwikkeld in grasshopper3d, dit omdat het voortbouwt op model 2 en omdat geometrie gebruikt wordt om de oplossingenruimte te genereren. Zoals zichtbaar op bovenstaande afbeeldingen is de geometrie 3-dimensioneel en al complexer. Het is al duidelijker dat dit om bijvoorbeeld een woonunit gaat. De werking van het model wordt uitgelegd op basis van een voorbeeldsituatie op onderstaande afbeeldingen die stap per stap de werking zal verduidelijken.



Figuur 34 - Stap 1: beginvolume



Figuur 35 - Volgende stappen: doorsnijdingen

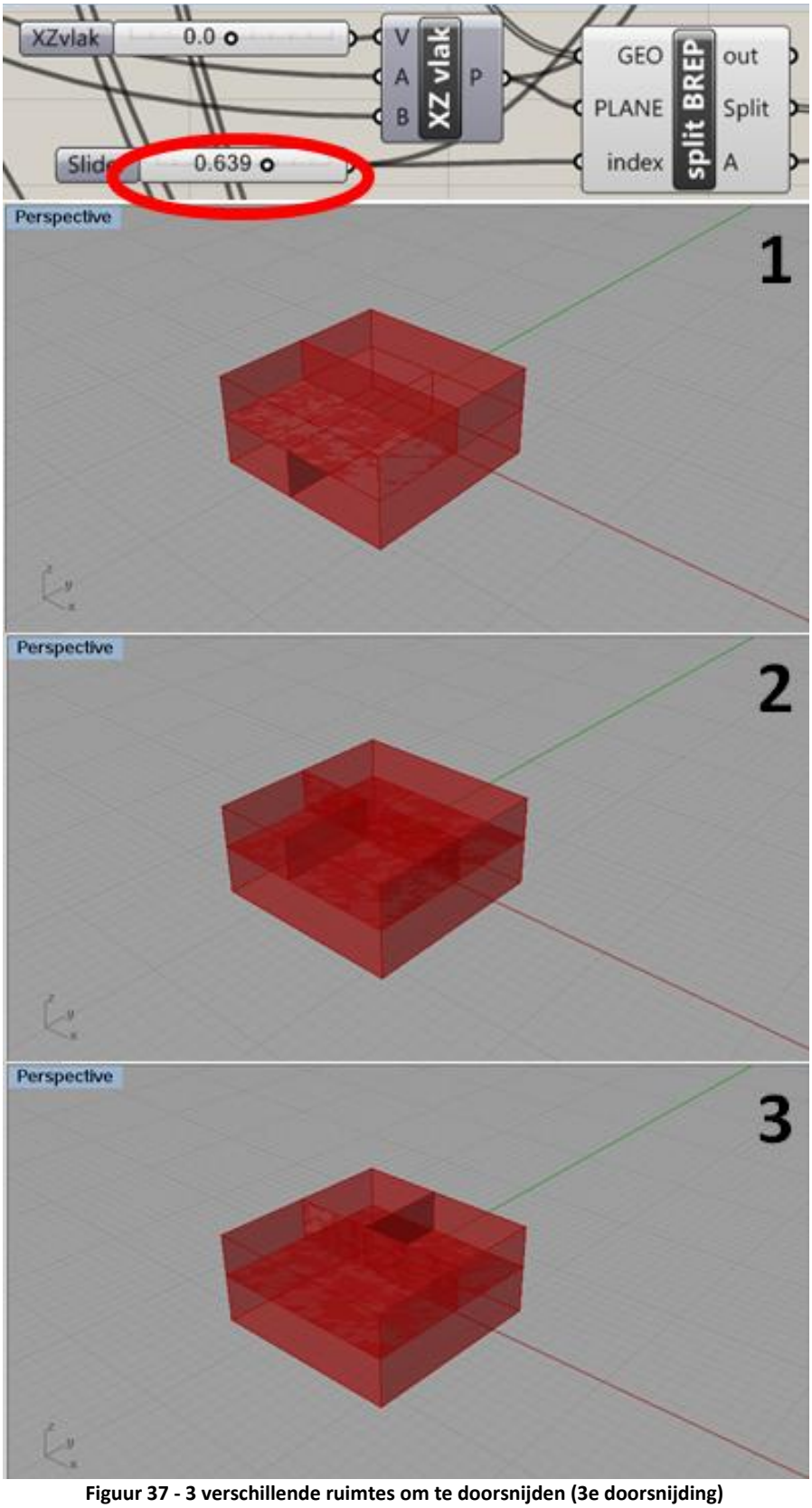


Figuur 36 - Laatste stap: permuteren (vb: 1 mogelijkheid uit 362880, nog zonder beweegbare snijvlakken)

Stap 1 is het aanmaken van begincomponent, zoals eerder steeds ‘de kubus’ genoemd. Dit is de omsluitende vorm van het gebouw. Het volume heeft variabele afmetingen in hoogte, breedte en lengte. Er zijn in dit geval bepaalde maxima en minima ingesteld omdat het volume aanzien wordt als een woonunit. Dit kan natuurlijk ruimer bekeken worden.

De volgende stappen bestaan uit het doorsnijden van het volume naar keuze door de gebruiker. Ook hier zijn er bepaalde minima en maxima opgesteld om realistische en bruikbare ruimtes te bekomen. Doorsnijdingen zijn loodrecht voor dit onderzoek, wat al zeer veel mogelijkheden zal opleveren. Dit is ook de meest gebruikte manier om wanden te plaatsen: typisch kiest men eerder voor rechte wanden dan voor scheef geplaatste wanden. Voor een verdere uitwerking van dit model bestaat er dus nog een mogelijkheid om ook schuine vlakken toe te voegen. Dit kan eenvoudig door een hoekparameter aan het vlak toe te voegen. Voor het doorsnijden werd in grasshopper3d een zelfgecreëerde component in C# aangemaakt. De code hiervan bevindt zich in bijlage D.

Voor elke doorsnijding, behalve de eerste, zijn er meerdere mogelijkheden van welk deelvolume je wilt doorsnijden. Het is aan de gebruiker om te kiezen welk volume hij wil doorsnijden. Dit gebeurt door de ‘sliders’ in het grasshopper-model te verschuiven. Dit wordt duidelijk gemaakt met de voorbeeldsituatie in onderstaande afbeelding. Op de afbeelding worden de 3 mogelijkheden getoond hoe het volume kan worden doorgesneden. Het doorsnijden gebeurt steeds maar door 1 ruimte. De slider met de rood omcirkelde waarde geeft de gebruiker de mogelijkheid om hiertussen te kiezen. Het interval 0 tot 0.33 geeft situatie 1, interval 0.33 tot 0.66 situatie 2 en interval 0.66 tot en met 1 situatie 3. Elke situatie heeft zodoende een even grote kans van voorkomen.



Figuur 37 - 3 verschillende ruimtes om te doorsnijden (3e doorsnijding)

Het is mogelijk om bij verdere ontwikkeling deze component te gebruiken voor optimalisaties met de computer door middel van een algoritme dat dan zelf door de 'sliders' loopt om de beste oplossing te vinden. Hier wordt dit niet gedaan omdat het te lang zou duren door het te grote aantal variabelen en door de geometrische representatie die bij elke positie zou moeten worden herberekend.

De laatste stap bestaat uit het laten permuteren van de gecreëerde ruimtes voor alle mogelijke relaties tussen de ruimtes te modelleren.

2.3.3. *Nabijheidmatrix (en. Adjacency matrix)*

Naast deze stappen loopt parallel het bijhouden van de matrices die overeenstemmen met de grondplannen van elk niveau. Op het einde van de bewerkingen kan vervolgens de nabijheidmatrix opgesteld worden op basis van deze gecreëerde topologische matrices die de geometrie representeren.

Onderstaande afbeelding maakt de opbouw van de nabijheidmatrix duidelijk door middel van de voorbeeldsituatie die in de vorige paragraaf werd opgebouwd. Elke keer er een doorsnijding plaatsvindt, wordt de topologische matrix van het volume bijgehouden. In het begin, voordat er een doorsnijding heeft plaatsgevonden, is de topologische matrix: $[0]$. Bij elke nieuwe doorsnijding wordt deze matrix aangepast. De nieuwe zone wordt steeds zo rechts of zo hoog mogelijk in de nieuwe matrix geschreven om zo een bepaalde volgorde te kunnen respecteren.

Eerst wordt de topologische matrix opgesteld, vertrekkend van de 0 matrix.

Wordt er XY doorgesneden, dan komt er een nieuwe matrix bij om op deze manier het onderscheid tussen de verdiepingen bij te houden:$[0]\ en\ [1]$.

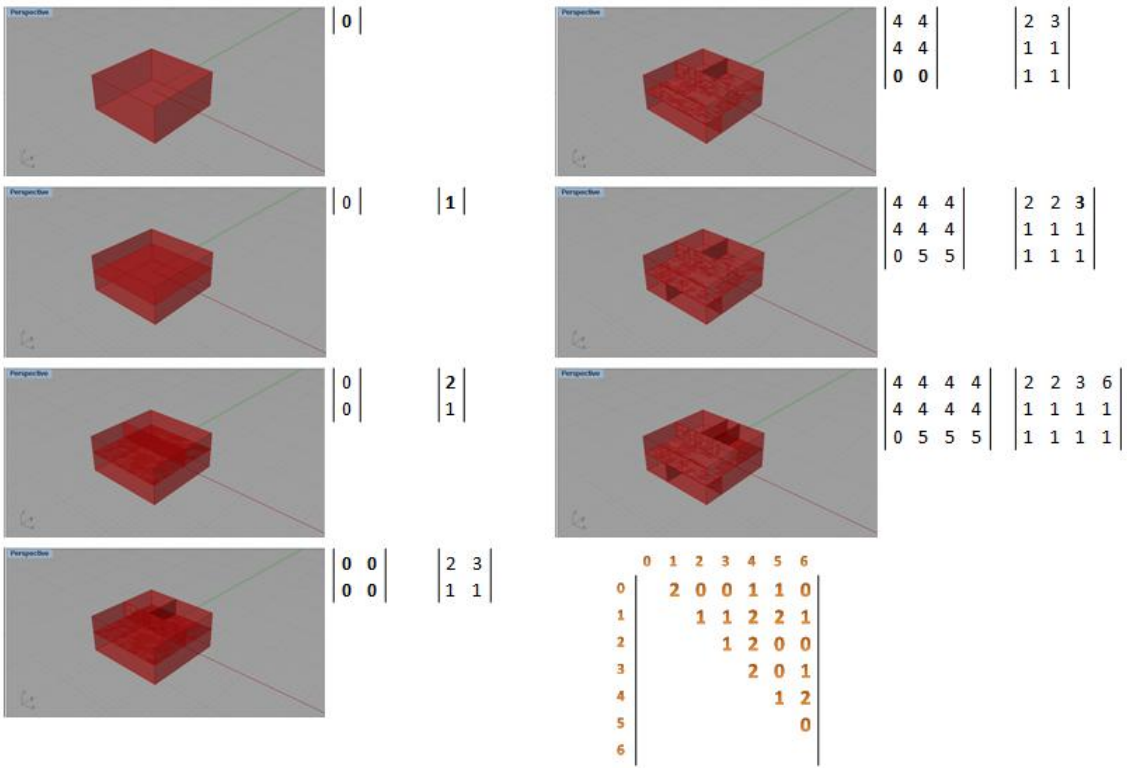
Wordt er XZ doorgesneden, dan krijgt elke matrix een extra rij met de opgedeelde ruimte in 2 gesplitst waar onderaan de bestaande ruimte wordt geschreven en bovenaan de nieuwe ruimte. Ruimte 1 wordt opgesplitst: $\begin{bmatrix}0\\0\end{bmatrix}\ en\ \begin{bmatrix}2\\1\end{bmatrix}$. De opsplitsing gebeurt in beide matrices om zo de relaties te kunnen bijhouden. Zodat ruimtes met dezelfde matrixcoördinaat een boven-onder relatie krijgen.

Wordt er YZ doorgesneden, dan krijgt elke matrix een extra kolom met de opgedeelde ruimte in 2 gesplitst waar links de bestaande ruimte wordt geschreven en rechts de nieuwe ruimte. Ruimte 2 wordt opgesplitst: $\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\ en\ \begin{bmatrix}2 & 3\\1 & 1\end{bmatrix}$. Enzovoort voor alle doorgesneden.

De nabijheidmatrix wordt vervolgens opgesteld op basis van deze topologische matrices. Er wordt begonnen van een lege matrix met overal een nul. De applicatie loopt door de topologische matrices en bekijkt welke getallen naast elkaar liggen. Voor elke 2 ruimtes die raken wordt er dan op de nabijheidmatrix een 1 geschreven. Waar bijvoorbeeld ruimte 1 en 2 raken wordt op coördinaat [1,2] en [2,1] een 1 geschreven aangezien het een vierkante

matrix is. Vervolgens wordt de boven-onder relatie bekeken tussen de verschillende topologische matrices onderling. Voor elke matrixcoördinaat [i,j] wordt in de nabijheidmatrix het getal 2 geschreven op coördinaat $[[i,j]_0,[i,j]_1]$ en $[[i,j]_1,[i,j]_0]$. De matrix bestaat nu uit 2 verschillende informatiedragers, de cijfers 1 voor rakende ruimtes op eenzelfde niveau en de cijfers 2 voor de rakende ruimtes tussen verschillende niveaus. In principe kan je op de diagonaal nog een 3e informatiedrager plaatsen: namelijk de oppervlakte van elke ruimte.

Op het einde wordt deze matrix ook gepermuteerd volgens hetzelfde principe als bij de volumes zodat de matrices nog overeenkomen met de juiste situatie.



Figuur 38 - Opbouw nabijheidmatrix (voorbeeldsituatie)

2.3.4. *Besluit*

Het resultaat is bevredigend omdat je enorm veel mogelijkheden hebt, gemakkelijk input van de gebruiker toelaat en een geometrische representatie hebt van het gemaakte volume. Dit model is wel redelijk zwaar voor deze weinige bewerkingen omdat het steeds alle oplossingen opslaat, maar ook bij elke bewerking moet het alle geometrie opnieuw inladen. Geometrie niet enkel van het weergegeven model maar ook alle geometrische bewerkingen van alle mogelijke modellen. Door deze vrij beslissende factor van het zware model is ervoor gekozen om het aftoetsen van de nabijheidmatrix met de vooropgestelde relaties die door de gebruiker worden ingegeven nog niet uit te werken in dit model. In volgend hoofdstuk wordt dit wel uitgewerkt met het oog op de applicatie in het 2e deel van deze masterproef. Hierbij kan ook gezegd worden dat grasshopper3d op zichzelf ook geen ideale GUI (graphical

user interface) heeft voor een gebruiker die niet noodzakelijk alle achterliggende codes en spaghetti van componenten moet zien.

Een ander punt van kritiek is dat in stap 1 van dit model begonnen wordt van een basisvolume. Het feit dat een volume gekozen wordt om doorsnijdingen in te maken is enerzijds al een beperking omdat dit reeds op voorhand de maximaal omsloten vorm is (ook al wordt er bepaald dat bepaalde ruimtes als buitenruimte aanzien mogen worden). Anderzijds is het aantal mogelijkheden hiervan al zo enorm dat dit een goed onderwerp is van onderzoek om het model te testen. In principe kan je elke mogelijke vorm als input nemen en dan beginnen te doorsnijden. Een andere benadering zou kunnen zijn om een additief model te maken, zodat je steeds ruimtes toevoegt aan je bestaande volume. Maar dit kan onderwerp zijn van een volledig nieuwe masterproef, dus dat zou te ver leiden.

2.4. Contextafhankelijkheid

Het idee van contextafhankelijkheid wordt in het kader van dit onderzoek niet meer toegepast omdat het te ver zou leiden, maar het zou op verschillende manieren benaderd kunnen worden, door bijvoorbeeld daglichtoptimalisaties die alle configuraties gaan testen in de context en maximaliseren naar zonnetoetreding. Tot nu toe is het volume enkel de gesloten kubus, maar er kan van uitgegaan worden dat bepaalde ruimtes buitenruimtes zijn en zo krijg je verschillende volumes. Ook kan deze kubus een representatie zijn van 1 onderdeel van het volledige ontwerp zodat een cluster van kubussen samen gevormd wordt volgens de meest geoptimaliseerde zonnetoetreding ten opzichte van elkaar maar ook ten opzichte van de context (verdere uitleg zie volgende alinea).

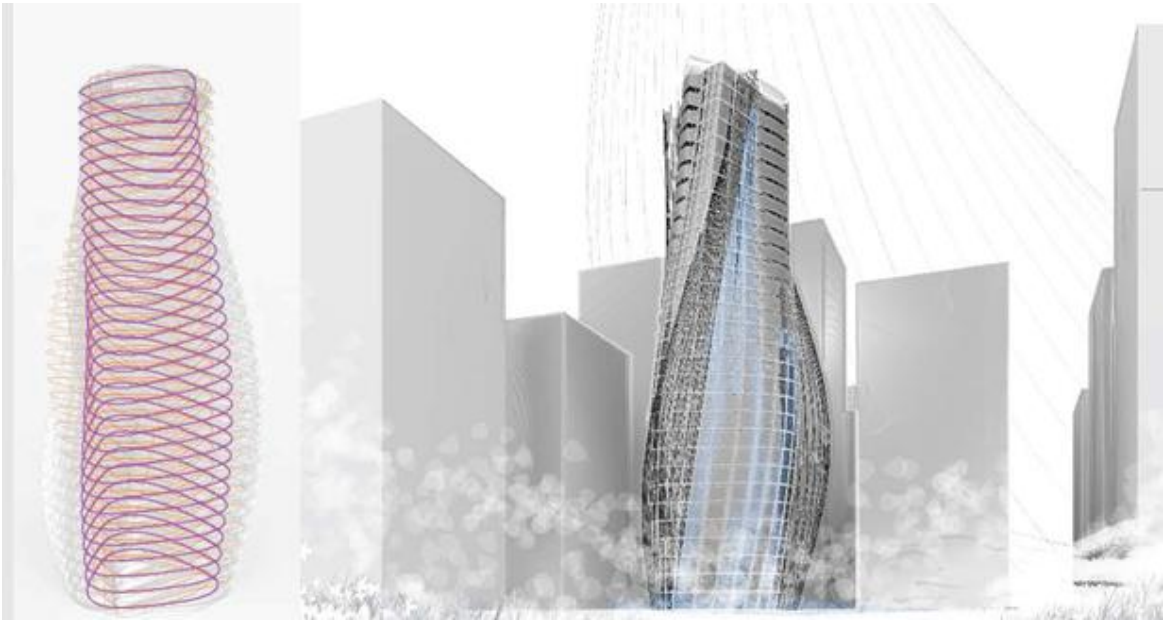
Ook zouden de eerder vooropgestelde regels van het detentiehuis hier kunnen terugkomen zodat er een samenspel bestaat tussen het idee van model 1 en de ideeën van model 2 en 3.

2.4.1. *Zijsprong: zonnetoetreding analyse*

In het kader van de ‘Smart Geometry’ workshop die ik dit jaar kon bijwonen in Londen werden bepaalde interessante toepassingen over zonnetoetreding en schaduwen verder ontdekt die handig zouden gebleken zijn indien dit onderzoek meer naar de contextuele richting werd gevoerd. In de onderzoekscluster werd een volume gemodelleerd dat enkele variabelen had die de vorm van het gebouw deden veranderen. Het was dan de bedoeling om dit model te plaatsen in variërende contexten in het stedelijke weefsel en vervolgens de variabelen van het gebouw in deze variabele context met een iteratieve loop te laten veranderen en vervolgens te kijken wat de gemiddelde meest optimale vorm was.

Gemiddeld omdat er zowel een variabele context als een variabel gebouw is. Deze cluster ging over de probabiliteitsanalyse, om te kijken in een onzekere toekomst wat voor

gebouwen er zouden kunnen bestaan in variabele contexten. Onderstaande afbeelding geeft een resultatenruimte weer van 1 experiment.



Figuur 39 - Resultatenruimte Smart Geometry experiment

3. Besluit

In het kader van dit onderzoek wordt gekozen om dieper in te gaan op de ruimtelijke configuraties van één deelvolume zoals besproken in paragraaf 2.2 en 2.3 van dit hoofdstuk. Dit volume is een representatie van een woonunit of individuele verblijfsruimte zoals deze zou kunnen voorkomen in het detentiehuis. Uit de conclusies van de paragrafen is gebleken dat de geometrische aanpak van deze studie leidt tot een vertraagde werking van de applicatie in grasshopper3d. Dit net omdat al deze volumes bij elke nieuwe bewerking weer opnieuw gegenereerd moeten worden. Verschillende auteurs betwisten ook deze dominante rol van geometrie in ontwerpsoftware. (Kilian, 2006)

Voor het vervolg van dit onderzoek is ervoor gekozen om dieper in te gaan op een tekstuele representatie van een ontwerp. Met enkel 1 tekstregeltje kan een volledig grondplan beschreven worden. Het voordeel hiervan is dat alle bewerkingen op basis van deze tekst gebeuren en dat pas achteraf de geometrische representatie gegenereerd kan worden. Het gevolg hiervan is dat de geometrie slechts 1 keer gegenereerd moet worden wat een veel snellere werking van de applicatie zal teweegbrengen.

4. TEKSTREPRESENTATIE

In dit laatste hoofdstuk van het onderzoek worden de principes, die vastgesteld werden tijdens de vorige hoofdstukken, verder ontwikkeld. Het uitgangspunt is de nabijheidmatrix, uit hoofdstuk 2: evaluatie, die als input zal dienen voor het vastleggen van de relaties die men al dan niet in het ontwerp wil. Deze set van regels wordt door de gebruiker op voorhand vastgelegd en zal gebruikt worden om de grondplannen, die gegenereerd worden op basis van de inputwaarde van het aantal ruimtes, mee af te toetsen. De manier waarop dit gebeurt bouwt verder op hoofdstuk 3: ontwerp, waar verschillende modellen ontwikkeld zijn die een hele reeks mogelijke configuraties opbouwen die al dan niet voldoen aan deze vooropgestelde regels. Uit dit vorige hoofdstuk bleek echter wel dat het gebruik van grasshopper3d met zijn geometrische bewerkingen een grote beperking biedt enerzijds voor de snelheid van de applicatie maar anderzijds is het ook interessanter om nog niet onmiddellijk dit principe van ruimtes genereren met een onderlinge relatie aan een geometrie te koppelen. Op deze manier wordt er nog enorm veel vrijheid behouden en wordt er enkel een uitspraak gedaan over de relaties, waar het eigenlijk om gaat als er wordt gekeken naar de input, zijnde de nabijheidmatrix. Natuurlijk zal het ook wel mogelijk zijn om deze meer abstracte informatie van grondplannen te representeren in een mogelijke geometrie. Dit is louter representatie en wordt dus ook enkel aan het einde van de applicatie gegenereerd, wat een snellere werking zal bevorderen.

1. Kader

Dit hoofdstuk baseert zich op het principe van 'floorplanning'. Dit idee wordt ook gebruikt voor het automatische uittekenen van geïntegreerde elektronische circuits om de meest optimale plaatsing van de onderdelen te verkrijgen. Het is een essentieel ontwerponderdeel voor een hiërarchische opbouw van bouwmodules. Floorplanning kan snel een feedback geven om architecturale beslissingen of geschatte elektronische chip oppervlakken te evalueren. De technologie gaat vooruit en zo worden ook deze circuits steeds groter. Dit zorgt ervoor dat floorplanning steeds belangrijker zal worden voor de kwaliteit van very large scale integration (VLSI) designs. (Wang, Chang, & Cheng, 2009)

Het ontwerpvraagstuk zoals het in dit onderzoek behandeld wordt, kan ook op deze manier benaderd worden aangezien het om hetzelfde idee van schikken van ruimtes gaat. Het zal de bedoeling zijn om alle mogelijke oplossingen te genereren en dan hiermee zelf aan de slag te gaan, waar het bij elektronische circuits de bedoeling is om naar 1 optimale oplossing te gaan via algoritmes.

Er zijn verschillende benaderingen mogelijk voor het floorplanning probleem. In het kader van dit onderzoek wordt de methode onderzocht die gebruikt maakt van de 'Polish expression'. Deze manier van representeren is efficiënt, flexibel en zeer effectief voor het modelleren van geometrische relaties (A links van B, A rechts van C , B boven D enz.) Het idee bij dit onderzoek is om op basis hiervan alle mogelijke oplossingen te genereren en dan op een eigen manier de "goede" oplossingen te zoeken door middel van de nabijheidmatrix die de input is van de applicatie.

Normaal wordt bij 'floorplanning' gebruik gemaakt van bijvoorbeeld een 'simulated annealing' algoritme (Sait, 1999). Dit is een generiek probabilistisch heuristiek optimalisatiealgoritme gebruikt om een benadering van het globale optimum van een gegeven functie in een grote zoekruimte te vinden. In het kader van dit onderzoek is dit niet gewenst, aangezien het de bedoeling is om alle oplossingen te genereren die voldoen aan de vooropgestelde relationele regels en niet 1 optimum te bekomen.

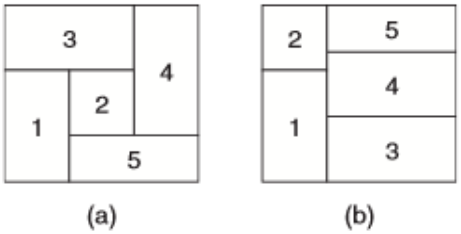
2. Floorplanning

Bij elektronische chips is het gewoonlijk de bedoeling om de oppervlakte van de chip te minimaliseren. Door deze methode te gebruiken zal het resultaat van dit onderzoek elke ruimte zo klein mogelijk simuleren. Het is ook maar een representatie van wat het zou kunnen zijn, dit net omdat er afgestapt wordt van de geometrie. Het zal dus de bedoeling zijn om achteraf het resultaat van de applicatie correct te interpreteren en het niet zomaar letterlijk over te nemen.

2.1. Model

Er zijn verschillende manieren om een model op te bouwen. Er kan gewerkt worden op een additieve manier, waar steeds ruimtes worden toegevoegd op basis van standaardgroottes en relaties. Maar het principe waarop dit onderzoek zal voortbouwen is het idee van het rechthoekige volume dat doorgesneden wordt, zoals ook in het vorig hoofdstuk behandeld werd.

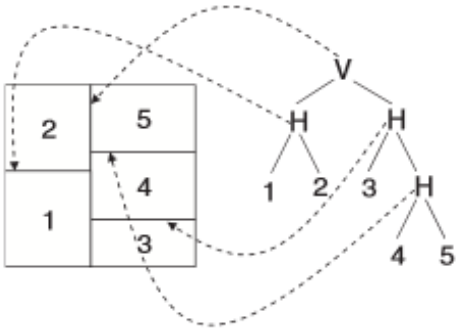
Bij het 'floorplanning' concept worden 2 soorten modellen gedefinieerd die vertrekken van een rechthoekig volume: 'slicing' en 'non-slicing floorplans'. De 'slicing' modellen kunnen verkregen worden door steeds opnieuw een volume horizontaal of verticaal te doorsnijden, waar de 'non-slicing' modellen dit niet kunnen. Op onderstaande afbeelding wordt het onderscheid hiertussen getoond op basis van een voorbeeld.



Figuur 40 - (a) non-slicing - (b) slicing

2.2. Boom

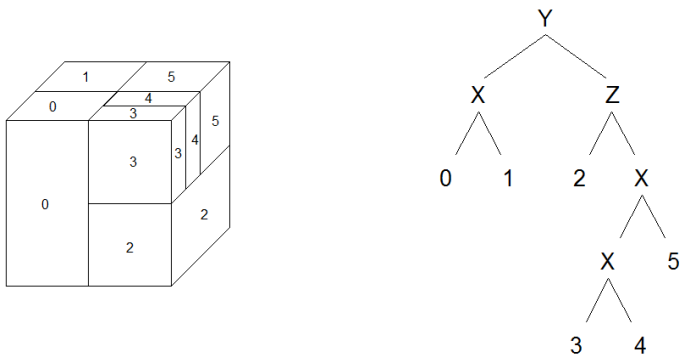
De 'slicing' grondplannen kunnen worden voorgesteld in een boom die zeer handig zal blijken in het representeren van een grondplan onder de vorm van tekst. De boom is opgebouwd uit takken welke in de kruispunten bestaan uit operatoren of doorsnijdingen en aan de uiteinden bestaan uit de ruimtes. Er zijn in de 2-dimensionele representatie 2 operatoren of doorsnijdingstypes: H en V. H doorsnijdt het volume horizontaal en V doorsnijdt verticaal. Na een doorsnijding is er een opsplitsing in de boom, links staat het cijfer van het oorspronkelijke (deel-)volume dat doorgesneden werd en rechts het nieuwe. Zo ook zal in de representatie van het grondplan na een horizontale snede de oorspronkelijke ruimte onder de nieuwe ruimte staan en bij een verticale snede respectievelijk links en rechts. Onderstaande afbeelding geeft een boom weer van het 'slicing' voorbeeld uit voorgaande afbeelding.



Figuur 41 - Boom van een slicing grondplan

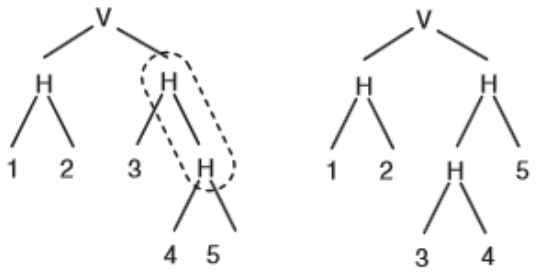
Voor het toepassen van het onderzoek in de applicatie zal deze 2-dimensionele techniek gebruikt worden. Dit vereenvoudigt het werk van het schrijven van de codes enorm maar geeft op dezelfde manier de resultaten weer en zal dus de werking duidelijk maken. Voor de volledigheid van het onderzoek zal steeds wel naast deze 2-dimensionele representatie ook een 3-dimensionele representatie uitgelegd worden om de uitbreidingen en mogelijkheden te duiden.

Om de derde dimensie toe te voegen wordt er gewerkt met de letters X, Y en Z. X zal dezelfde functie hebben als H, het opdelen in de richting parallel aan de x-as (in 2d dus onder en boven opsplitsing). Y heeft dezelfde functie als V in het 2-dimensionele model. De bijkomende operator is Z, die een ruimte kan opsplitsen 3-dimensioneel in een beneden en een boven. Op deze manier zal het ook mogelijk zijn om deze 3 hoofdoperatoren uit te breiden met een gewicht op welke plaats ze het volume doorsnijden, aangezien nu na een doorsnijding steeds 2 even grote ruimtes gemaakt worden. Dit principe wordt in het kader van de applicatie niet verder uitgewerkt om de complexiteit ervan niet te veel te laten toenemen, maar het is zeker de moeite om dit te vermelden, omdat het de mogelijkheden van deze techniek weergeeft. In hoofdstuk 3 was dit bijvoorbeeld wel mogelijk om op verschillende plaatsen te doorsnijden. Er worden dus bepaalde functionaliteiten achterwege gelaten voor de applicatie om de werkbaarheid te bevorderen, maar zoals eerder vermeld is de locatie van het doorsnijden van minder belang aangezien er toch steeds een terugkoppeling moet gemaakt worden van de applicatie naar het werkelijke ontwerp waarbij de ontwerper vervolgens zelf de groottes van de ruimtes kan aanpassen binnen het gegeven relatieschema. Onderstaande afbeelding geeft een voorbeeldsituatie weer van een 3-dimensionele situatie met zijn bijbehorende boom.



Figuur 42 - 3-dimensioneel voorbeeld van het 'slicing floorplan' principe

Merk op dat bij elk grondplan meerdere bomen kunnen voorkomen omdat de volgorde van snijden kan verschillen. Deze manier van representeren levert een enorme oplossingenruimte die voor een groot deel bestaat uit dezelfde grondplannen, wat een vertragende werking zal hebben op de zoektocht naar de grondplannen die voldoen aan de vooropgestelde relaties. Daarom is het aangewezen om deze representaties die hetzelfde grondplan weergeven eruit te halen. Om dit te doen wordt in de literatuur het principe van de 'skewed slicing tree' voorgesteld (Wang, Chang, & Cheng, 2009). Deze boom heeft na elk kruispunt op de rechtertak nooit dezelfde operator als op het kruispunt zelf. Als aan dit principe wordt voldaan zullen er geen dubbele grondplannen gegenereerd worden. Onderstaande afbeelding geeft 2 verschillende bomen weer die beiden het grondplan van het vorige voorbeeld van een 2-dimensioneel grondplan weergeeft.



Figuur 43 - (links): 'skewed' - (rechts): 'non-skewed'

2.3. Normalized Polish Expression

Bij 'floorplanning' kan het grondplan op verschillende manieren worden weergegeven met het oog op een algoritme om het meest optimale te vinden. Populaire representaties bij de 'simulated annealing' benadering zijn de 'B*-tree', de 'Sequence Pair' en de 'Normalized

Polish Expression'. Deze laatste zal voorwerp worden van het verdere onderzoek en ook gebruikt worden voor de toepassing van de applicatie.

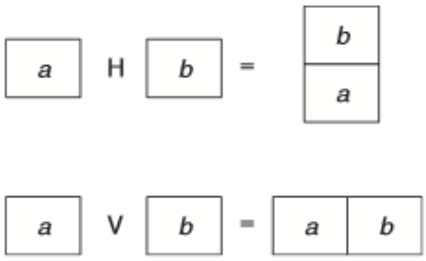
2.3.1. *De uitdrukking*

In vorige paragraaf werd de boom geïntroduceerd om een grondplan te representeren. Op eenzelfde manier zal ook deze 'Polish expression' het grondplan representeren vertrekkend vanuit de boom. Elke 'Polish expression' bestaat uit 1 tekstregel bestaande uit de letters van de operatoren en de cijfers van de ruimtes. De lengte van de tekstregel is 2 keer het aantal ruimtes min 1, aangezien er steeds 1 operator minder is dan dat er ruimtes zijn. De volgorde van de tekens geeft de structuur van de boom weer. Zoals eerder vermeld in de alinea van de boom, zijn er meerdere representaties die hetzelfde grondplan voorstellen. Ook bij deze 'Polish expression' is dat het geval. Om deze problemen weg te werken wordt de 'normalized Polish expression' in de literatuur gedefinieerd overeenkomstig de 'skewed slicing tree' (Wong, 1986). Zoals de boom op 2 opeenvolgende kruispunten geen zelfde operator mag hebben, geldt ook hetzelfde hier dat de uitdrukking geen direct opeenvolgende gelijkaardige operatoren mag bevatten (VV en HH in de 2-dimensionele representatie). Dit resulteert in een 1 op 1 overeenkomstigheid tussen de 'skewed slicing trees' en de 'normalized Polish expressions'.

2.3.2. *Uitdrukking naar grondplan*

Om deze 'normalized Polish expression' nu om te zetten naar zijn overeenkomstig grondplan wordt gebruik gemaakt van een 'bottom-up' methode op basis van deeluitdrukkingen die in de volledige uitdrukking staan. Namelijk elke 'Polish expression' is opgebouwd uit verschillende deeluitdrukkingen die elk een operator bevatten en zo kan de volledige uitdrukking begrepen worden en getransformeerd worden naar het grondplan.

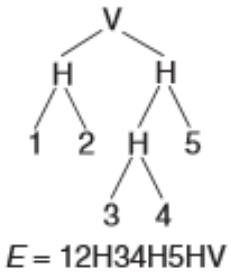
In het geval van de 2-dimensionele representatie zijn er zoals eerder vermeld 2 operatoren: V en H. De uitdrukking wordt als volgt opgebouwd: als a en b 2 ruimtes voorstellen of deel-grondplannen, dan betekent de uitdrukking abH dat ruimte a onder ruimte b staat en betekent de uitdrukking abV dat de ruimte a links van ruimte b staat zoals weergegeven wordt op onderstaande afbeelding.



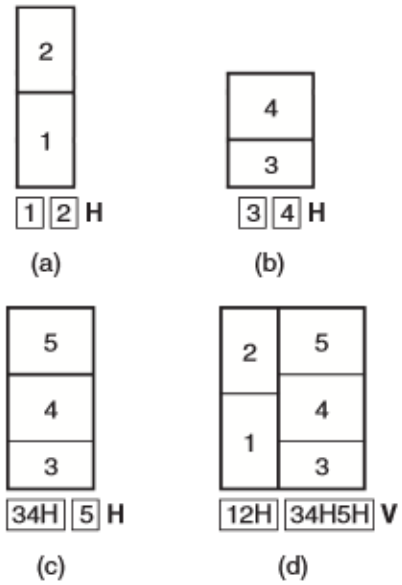
Figuur 44 - Uitleg 'Polish expression'

Hetzelfde principe geldt ook voor de 3-dimensionele methode maar dan met operatoren X, Y en Z zoals eerder uitgelegd werd bij de boom. Bijvoorbeeld: uitdrukking 12H betekent dat ruimte 1 onder ruimte 2 wordt geplaatst, de uitdrukking 34V betekent dat ruimte 3 links van ruimte 4 wordt geplaatst. Voor de uitdrukking 123H… kan gesteld worden dat ruimte 2 onder ruimte 3 geplaatst wordt aangezien de operatoren steeds gelden voor de 2 voorstaande ruimtes of modules van ruimtes.

Het lezen van de uitdrukking gebeurt van links naar rechts als volgt: er wordt gecontroleerd of het een operator of een ruimte is. Is het een ruimte dan wordt het opgeslagen in een virtuele lijst. Kom je een operator tegen dan worden de vorige opgeslagen ruimtes uit deze virtuele lijst gehaald en in relatie ten opzichte van elkaar gezet volgens de functie van de operator. Dit geheel geldt dan als sub-grondplan wat dezelfde functie heeft als 1 ruimte en dus kan opgeslagen worden in een virtuele lijst om nadien door een operator gebruikt te kunnen worden. Dit moet worden blijven gedaan tot elke ruimte en elke operator gebruikt is en het finale grondplan weergegeven wordt. Het voorbeeld uit de vorige paragrafen wordt in wat volgt opgebouwd op basis van zijn 'normalized Polish expression'.

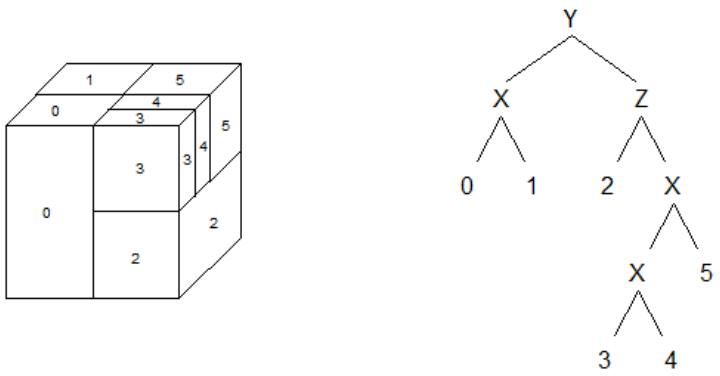


Figuur 45 – Voorbeeld 2D: 'skewed slicing tree' naar 'normalized Polish expression'



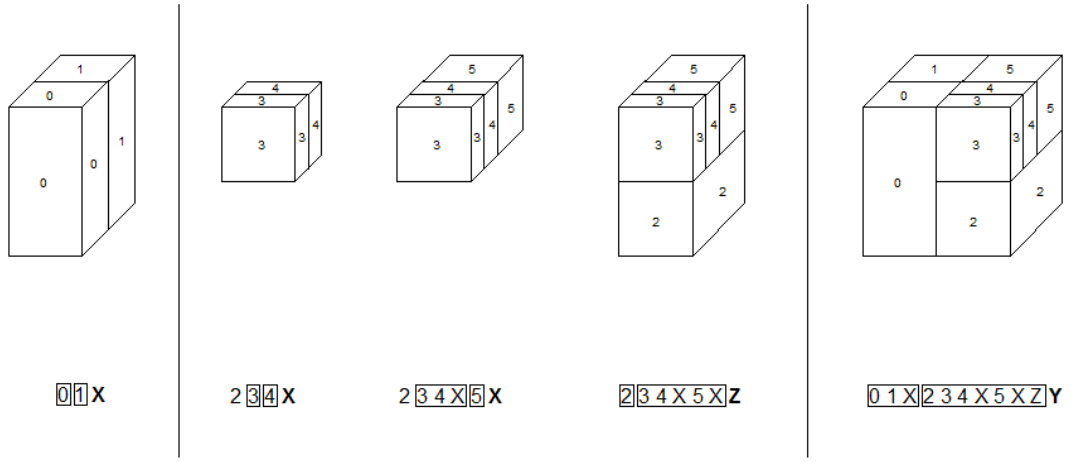
Figuur 46 – Voorbeeld 2D: 'normalized Polish expression' naar grondplan (4 stappen)

Hetzelfde kan gedaan worden voor de 3-dimensionele representatie aan de hand van het voorbeeld uit vorige paragrafen. Onderstaande afbeelding geeft deze uitwerking weer.



E = 01X234X5XZY

Figuur 47 – Voorbeeld 3D: 'skewed slicing tree' naar 'normalized Polish expression'



Figuur 48 – Voorbeeld 3D: 'normalized Polish expression' naar grondplan (5 stappen)

2.3.3. *Oplossingenruimte*

De serie van alle 'normalized Polish expressions' vormt de oplossingenruimte die gebruikt zal worden bij de applicatie om na te gaan van welke configuratie de nabijheidmatrix voldoet aan de vooropgestelde inputmatrix van relaties. Voor de grootte van de oplossingenruimte wordt verwezen naar de resultaten van de applicatie in volgend hoofdstuk waar dit een onderdeel zal zijn van de testen die met de applicatie zullen gebeuren.

3. Toepassing

De tekstuele methode om grondplannen te beschrijven wordt in de huidige paragraaf gebruikt om, refererend naar de vorige hoofdstukken, een toepassing te ontwikkelen om vertrekkend van een input van de gebruiker een aantal mogelijke oplossingen te bieden van grondplannen die voldoen. De input van de gebruiker omvat het aantal ruimtes en de relaties tussen deze ruimtes.

In wat volgt zullen alle C# codes die in bijlage zitten aan de hand van voorbeelden uitgelegd worden. Dit om te verklaren hoe enerzijds al deze theoretische opbouw van de 'normalized Polish expression' vertaald kan worden naar een toepasbare applicatie en anderzijds hoe van deze tekstregel een topologische matrix en een nabijheidmatrix kan worden gemaakt.

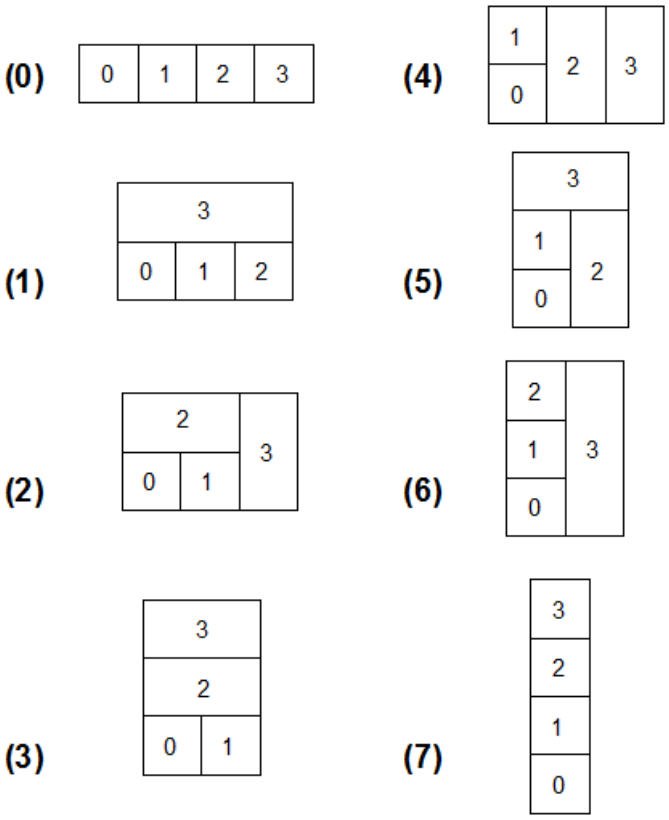
Er wordt gekozen voor een aanpak in een tekstgebaseerde console-omgeving van C# omdat na enkele experimenten in grasshopper3d, waar het ook mogelijk is om bomen te representeren aan de hand van datastructuren, gebleken is dat de tekstuele aanpak van representatie zoals beschreven in dit hoofdstuk sneller en eenvoudiger is. Met het oog op de visualisatie zou grasshopper3d wel meer mogelijkheden toelaten maar dan weer trager zijn. In hoofdstuk 5, waar de applicatie beschreven zal worden, wordt er meer ingegaan op de

visualisatie en de ‘graphical user interface’ (GUI) van de applicatie en hoe deze zal worden opgebouwd door gebruik te maken van het gratis programma processing. Dit zal toelaten om de tekst op een visuele manier te representeren.

3.1. Voorbeeld

Ter inleiding van dit gedeelte wordt verklaard wat de uiteindelijke bedoeling is van dit hoofdstuk: namelijk om alle mogelijke oplossingen te genereren die overeenkomen met het aantal ruimtes dat de gebruiker voor ogen heeft. Daarbij hoort (1) de ‘normalized Polish expression’, (2a) de topologische matrix en dus ook (2b) een representatie van een mogelijk grondplan en (3) de nabijheidmatrix. Het relateren van de nabijheidmatrix aan de voorwaarden die de gebruiker heeft vooropgesteld wordt behandeld in hoofdstuk 5 bij de effectieve uitwerking van de applicatie.

Het voorbeeld omvat alle mogelijke oplossingen die gegenereerd kunnen worden met als inputwaarde 4 verschillende ruimtes. Onderstaande afbeeldingen geven het resultaat weer: 8 mogelijke oplossingen om de ruimtes te schakelen. Oriëntatie speelt ook een rol want zoals zichtbaar, zijn sommige oplossingen dezelfde maar 90 graden geroteerd. Voor elke mogelijke oplossing moeten we per configuratieoplossing de ruimtes ook nog laten permuteren. Per deeloplossing zijn er dus 24 mogelijke permutaties, wat het totaal aantal mogelijke combinaties, met elk zijn nabijheidmatrix, op 192 mogelijkheden brengt.



Figuur 49 - Voorbeeld: de 8 mogelijke configuraties voor 4 verschillende ruimtes

In wat volgt wordt de achterliggende werking uit de doeken gedaan. Dit is een redelijk specifieke compliceerde uitleg. Er wordt hier vooraf verwezen naar het eind van dit hoofdstuk waar een overzicht staat van de volledige werking. Ook wordt er verwezen naar volgend hoofdstuk waar de applicatie zal worden uitgelegd. Dit biedt, eventueel vooraf, een duidelijker kader voor de werking uitgelegd in de volgende paragrafen.

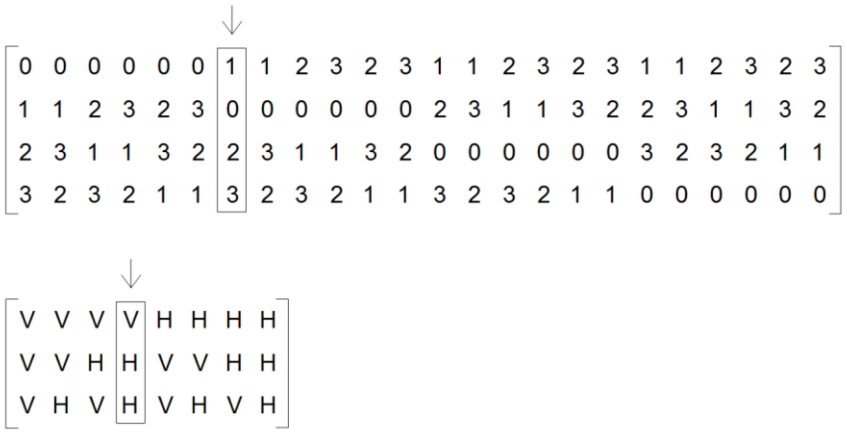
3.2. Normalized Polish expression

3.2.1. *Werking*

De toepassing in grasshopper3d begint met de enige input van het aantal zones. Dit is de sturende factor om alle mogelijke uitdrukkingen te genereren. Dit is het beginpunt van alle bewerkingen die daarop toepasbaar zijn. In wat volgt zullen alle voorbeelden uitgelegd worden aan de hand van het geval waarbij er 4 zones zijn.

Om de 'normalized Polish expression' op te bouwen wordt er vertrokken van 2 matrices. De ene matrix bevat alle mogelijke combinaties van operatoren of slices waaruit dus steeds 1 kolom gebruikt zal worden per ontwikkeling. Zo zullen alle mogelijke combinaties van operatoren behandeld worden. De andere matrix bevat alle mogelijke combinaties van de

ruimtenummers. Deze nummers kunnen gezien worden als type-ruimtes zoals bijvoorbeeld: slaapkamer, leefruimte, gang enz. zoals het in dit onderzoek verwijst naar een woonunit van het detentiehuis. Onderstaande afbeelding geeft het voorbeeld voor 4 zones van deze matrices. Zoals zichtbaar op de afbeelding bestaat de matrix met ruimtenummers uit 4 rijen en de operatormatrix uit 3 rijen. Zoals eerder besproken in dit hoofdstuk omdat bij het creëren van 4 ruimtes er slechts 3 operators of doorsnijdingen nodig zijn.



Figuur 50 - Boven: cijfermatrix / onder: operatormatrix

Deze input wordt gebruikt om vervolgens de 'normalized Polish expression' te vormen. Deze uitdrukking wordt gevormd door zich te houden aan een aantal regels zoals eerder besproken. De praktische omzetting van regels naar code wordt in wat volgt besproken. Eerst moet duidelijk gesteld worden dat in wat volgt er steeds per 1 kolom van de cijfermatrix en per 1 kolom van de operatormatrix gewerkt wordt. Nadien zal de code over alle kolommen lopen zodat elke combinatie gebruikt en opgeslagen wordt.

Vervolgens worden er eerst een aantal constanten en variabelen gecreëerd die ervoor gaan zorgen dat de code weet welke regel toegepast moet worden. De constanten zijn de integers: *numbercount*, *slicescount* en *polishcount*, welke respectievelijk gelijk zijn aan: het aantal zones, het aantal operators en het totaal van de 2 voorgaande. In het geval van het voorbeeld met 4 ruimtes zijn deze constanten respectievelijk gelijk aan: 4, 3 en 7.

De variabelen zijn de matrices: *sliceact* en *numberact*, welke als beginwaarde respectievelijk [1] en [2] hebben. Dit zijn respectievelijk het aantal operatoren en het aantal ruimtenummers die tot dan toe al zijn ingevuld in de uitdrukking. Deze starten op 1 en 2 omdat de eerste regel reeds enkele tekens zal invullen zoals uitgelegd wordt in volgende alinea.

Het eerste wat bij elke uitdrukking geldt is dat de eerste 2 tekens van de uitdrukking gelijk zijn aan de eerste 2 waarden van de cijfermatrix en dat het 3^{e} teken gelijk is aan het 1^{e} teken van de operatormatrix. Het omkaderde voorbeeld uit bovenstaande afbeelding zou dan als tussenresultaat de volgende uitdrukking hebben: '1 0 V _ _ _ _'.

Voor alle volgende tekens in de uitdrukking, behalve voor het laatste, worden 3 regels opgesteld, met bepaalde voorwaarden, die steeds moeten doorlopen worden:

De eerste regel stelt indien '*sliceact* = *numberact* – 1', dan moet er een ruimtenummer geplaatst worden in de code. Dit is omdat het aantal cijfers altijd groter moet zijn dan het aantal operatoren.

De tweede regel stelt indien '*numberact* < *numbercount* – 1 EN *sliceact* < *slicescount* – 1', dan moeten beide mogelijkheden opgeslagen worden, namelijk het volgende teken is een ruimtenummer en het volgende teken is een operator, waardoor het aantal mogelijkheden opsplitst naar 2.

De derde regel stelt indien '*sliceact* = *slicescount* – 1', dan moet er een ruimtenummer geplaatst worden en indien '*numberact* = *numbercount* – 1', dan moet er een operator geplaatst worden.

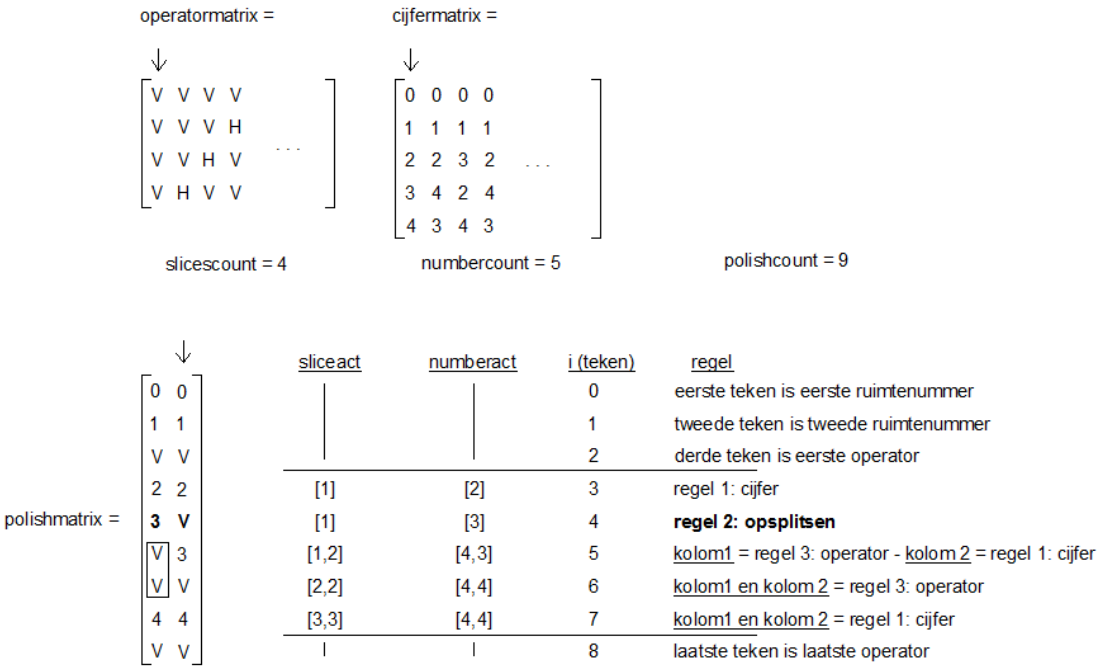
Wanneer bij deze voorgaande 3 regels een operator geplaatst wordt, gaat de *sliceact* omhoog met 1 en wanneer er een ruimtenummer geplaatst wordt gaat de *numberact* met 1 omhoog.

Vervolgens geldt voor het laatste teken dat het een operator is, omdat de 'normalized Polish expression' zoals eerder besproken van de vorm '01V' is waardoor achteraan steeds een slice of operator moet staan. Als laatste regel moeten de uitdrukkingen die gevormd werden waarin VV of HH voorkomt geschrapt worden om de genormaliseerde uitdrukking van de 'skewed slicing tree' te bekomen.

Het vervolg van het voorbeeld met 4 zones wordt als volgt opgelost:

- Tot hier toe was de deeloplossing: '1 0 V _ _ _ _'.
- Teken 4: Regel 1: *sliceact* [1] = *numberact* [2] – 1: dus het volgende ruimtenummer wordt geplaatst: '1 0 V 2 _ _ _', nadien zijn *sliceact* = [1] en *numberact* = [3].
- Teken 5: Regel 3: *numberact* [3] = *numbercount* (4) – 1: dus de volgende operator wordt geplaatst: '1 0 V 2 H _ _', nadien zijn *sliceact* = [2] en *numberact* = [3].
- Teken 6: Regel 1: *sliceact* [2] = *numberact* [3] – 1: dus het volgende ruimtenummer wordt geplaatst: '1 0 V 2 H 3 _'.
- Het laatste teken is de laatste operator: '1 0 V 2 H 3 H'.

Regel 2 is pas nodig van vanaf 5 ruimtes en meer. Deze regel is ook de reden waarom de *sliceact* en de *numberact* een matrix zijn. Zodat deze uitgebreid kan worden met meer kolommen zodat elke kolom in deze *numberact* en *sliceact* overeenkomen met een kolom in de oplossingenmatrix. Om dit duidelijk te maken wordt de eerste kolom van 5 zones in onderstaande afbeelding geduid.



Figuur 51 - Voorbeeld met 5 zones - duiding regel 2

De laatste stap is om de *polishmatrix* met alle oplossingen om te zetten naar tekstregels.

3.2.2. *C#-code*

Bijlage E.

3.3. Topologische matrix

3.3.1. *Werking*

Voor het opbouwen van de nabijheidmatrix en voor de uiteindelijke visualisatie van het grondplan wordt gebruik gemaakt van de topologische matrix. Deze bevat de meest eenvoudige planconfiguratie van de 'normalized Polish expression'. De ontwikkeling van de topologische matrix wordt in wat volgt beschreven.

Voor elke uitdrukking wordt er een lijst van matrices gemaakt met elk teken van de uitdrukking vervangen door een matrix met dat cijfer erin. Voor doorsnijdingen wordt V als '-1' en H als '-2' aangenomen om de eenvoudige reden dat alles dan een integer getal is.

Op de manier zoals besproken in eerder hoofdstuk worden steeds, indien er een operator is met daarvoor 2 cijfers, deze 2 matrices met ruimtenummer samengenomen op de manier die verondersteld wordt door de operatormatrix erna. Voor een verticale doorsnijding worden de 2 matrices met dimensies [1,1] vervangen door één matrix met een extra kolom

en dus met dimensie [1,2]. Voor het geval van een horizontale doorsnijding worden de 2 matrices met het ruimtenummer vervangen door één matrix met een extra rij en dus met dimensie [2,1]. De matrix met de operator verdwijnt. In het geval dat de 2 matrices van de ruimtenummers een verschillende dimensie hebben worden deze vergroot tot het kleinste gemene veelvoud. Zodat bijvoorbeeld bij een horizontale splitsing van matrices met dimensie [1,2] en [2,3] omgebouwd tot één matrix met dimensie [3,6]. Een voorbeeld van deze dimensies hieronder. Ter herinnering: matrix [-2] is de horizontale doorsnijding en bij een horizontale doorsnijding wordt de eerste matrix onderaan geplaatst en de 2^{e} bovenaan. Deze matrices moeten gezien worden als deelgrondplannen zoals ook besproken in eerder hoofdstuk.

$$[0 \quad 1]\begin{bmatrix}2 & 3 & 3\\ 2 & 4 & 4\end{bmatrix}[-2] == \begin{bmatrix}2 & 2 & 3 & 3 & 3 & 3\\ 2 & 2 & 4 & 4 & 4 & 4\\ 0 & 0 & 0 & 1 & 1 & 1\end{bmatrix}$$

Bij een verticale doorsnijding moeten dus de kolomdimensies opgeteld en de rijdimensies vereenzelvigd worden. Bij een horizontale doorsnijding moeten de rijdimensies opgeteld en de kolomdimensies vereenzelvigd worden.

3.3.2. *C#-code*

Bijlage F.

3.4. Nabijheidmatrix

3.4.1. *Werking*

De topologische matrix wordt gebruikt als uitgangspunt om de nabijheidmatrix te bepalen. Het is een eenvoudig algoritme dat doorheen de topologische matrix loopt en bij elk getal kijkt welke getallen errond staan en op deze manier wordt dan steeds het cijfer 1 geplaatst in de nabijheidmatrix die vertrekt van een nulmatrix.

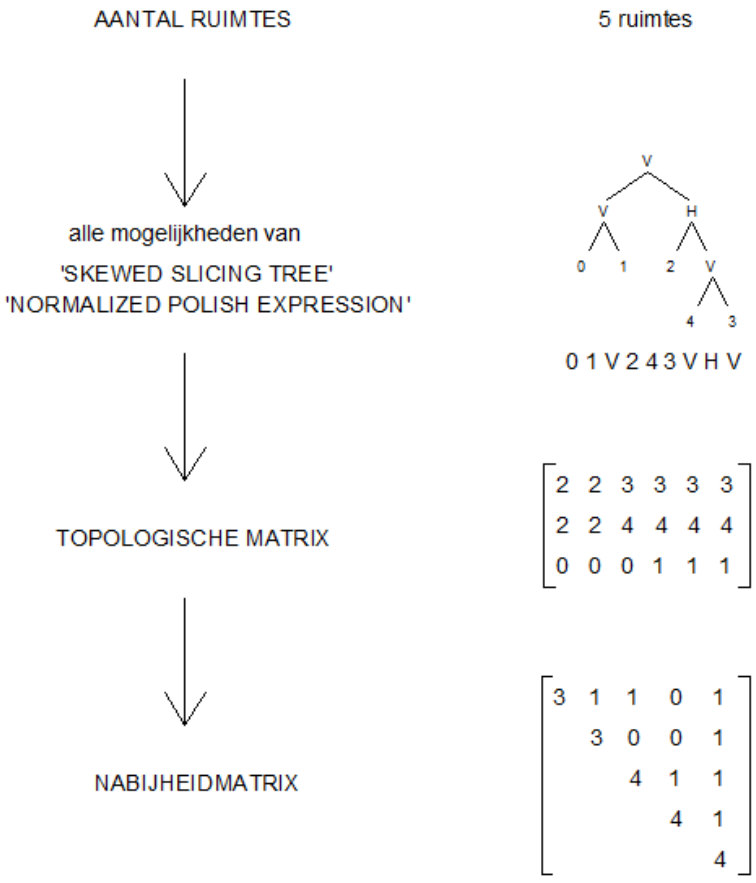
Op de diagonaal wordt de grootte van de ruimte opgeslagen in deze meest eenvoudige configuratie. Het is dus eerder een grootte ten opzichte van de andere ruimtes dan een absoluut getal van grootte. Deze waarde kan dus in verdere toepassing bij de applicatie gebruikt worden om een extra voorwaarde op te leggen, bijvoorbeeld dat men wil dat de slaapkamer groter is dan de badkamer. Op deze manier zullen ook veel mogelijkheden wegvallen om tot alle oplossingen te komen die voldoen aan de voorwaarden opgelegd door de gebruiker.

3.4.2. *C#-code*

Bijlage G.

3.5. Volledige werking

Onderstaande afbeelding maakt de volledige werking duidelijk en toont aan welke gegevens opgeslagen worden om verder van gebruik te kunnen maken bij de applicatie. Naast de tekst wordt een voorbeeld getoond van 1 mogelijke configuratie bij 5 zones.



Figuur 52 - Alle verzamelde gegevens

4. Besluit

Alle nodige informatie om een werkende applicatie te maken is hieruit besloten. Met enkel op tekst en matrix gebaseerde informatie zou de werking van de applicatie in volgend hoofdstuk vlot moeten verlopen in tegenstelling tot de geometrische werking met grasshopper3d.

Om de relaties in te geven zal in volgend hoofdstuk een kleine toepassing geschreven worden die de inputmatrix vergelijkt met de nabijheidmatrix. Dit werd in dit hoofdstuk nog niet behandeld omdat dit hoofdstuk de opbouw is naar de applicatie en deze code wordt geschreven met het oog op een resultatenbespreking, die ook zal volgen in volgend hoofdstuk.

Algemeen kan gesteld worden dat deze tekstrepresentatie voordelig is omdat ze vooraf geen geometrie koppelt aan relaties wat op onderzoeksvlak interessanter is. Dit omdat er dan nog veel breder in mogelijkheden kan gedacht worden. Een beperking van deze aanpak is dat er steeds binnen het rechthoekige kader wordt gewerkt, maar zoals eerder besproken zijn er wel mogelijkheden met het benoemen van buitenruimtes.

Een laatste punt is het 2-dimensionele van deze uitwerking van de applicatie. Dit is slechts voor de vereenvoudiging van het ontwikkelen van de applicatie. Zoals gezien in het onderzoek is het steeds mogelijk om elke uitdrukking ook om te zetten naar een 3-dimensionele representatie.

DEEL 2: APPLICATIE

5. PROTOTYPE

Dit hoofdstuk beschrijft de ontwikkeling van de applicatie, gebruikmakend van de kennis die vergaard werd in vorige hoofdstukken van het onderzoeksgedeelte. Het uiteindelijke doel zal zijn om, vertrekkend vanuit alle mogelijke rechthoekige configuraties van een aantal ruimtes, net die configuraties over te houden die voldoen aan een vooropgesteld aantal voorwaarden. De voorwaarden zijn wel- en niet-relaties alsook een groter dan of kleiner dan voorwaarde tussen 2 ruimtes. Uit de resultaten van de testen van de applicatie zal blijken hoeveel voorwaarden er gesteld moeten worden om de grote oplossingenruimte te reduceren tot enkele oplossingen. Op het einde van dit hoofdstuk zal ook teruggegrepen worden naar de case van het detentiehuis om als testcase voor de applicatie een woonunit te ontwikkelen.

1. Opbouw

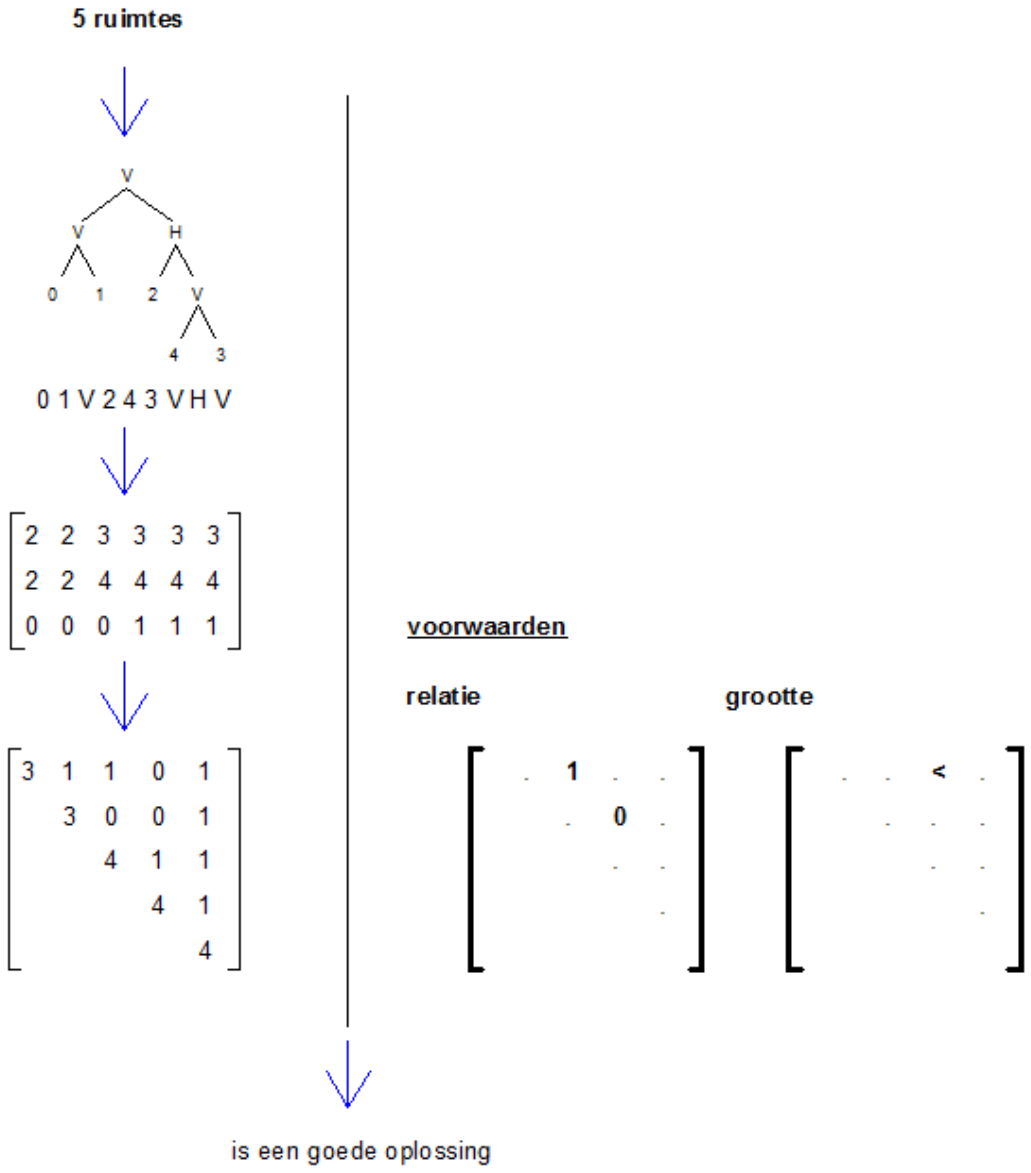
De applicatie werd ontwikkeld in het gratis programma processing dat veel vrijheid toelaat om binnen de programmeeromgeving van Java een applicatie te schrijven die niet afhankelijk is van geometrie zoals dat bij grasshopper3d wel de bedoeling is. Er wordt hiernaar overgeschakeld omdat grasshopper3d in het kader van dit onderzoek minder

mogelijkheden biedt om een goede GUI (graphical user interface) op te bouwen. Aangezien het wel een vooropgesteld doel is van de applicatie om enerzijds al gewoon mooi en anderzijds ook gebruiksvriendelijk te zijn is processing een beter alternatief. Hierbij wordt de achterliggende opbouw verborgen, maar wel is het belangrijk om de achterliggende werking bij elke verandering van input te begrijpen. Dit zal kunnen door de rechtstreekse aanpassing van de oplossingenruimte bij elke nieuwe input.

1.1. Input

Het doel van de applicatie is om slechts met enkele inputwaarden alle mogelijke oplossingen te genereren en ook deze te reduceren tot de enkelen die voldoen aan de vooropgestelde voorwaarden. De hoofdzakelijke input is het aantal ruimtes. Deze waarde bepaalt de oplossingenruimte waaruit 'goede' oplossingen gekozen kunnen worden. Zoals beschreven in vorig hoofdstuk bestaat de oplossingenruimte uit oplossingen die elk een 'normalized Polish expression' met daarbijhorende 'skewed slicing tree' hebben, alsook een topologische matrix om het grondplan te kunnen visualiseren, alsook een nabijheidmatrix die de voorwaarden zal toetsen en zo zal bepalen of het een 'goede' oplossing is.

Deze eigenschappen van elke oplossing worden uitvoerig beschreven in hoofdstuk 4 naar opbouw en werking. Onderstaande afbeelding toont een voorbeeld van de werking voor 1 oplossing. De inputwaarden zijn in het vet aangeduid. Het aantal ruimtes is 5 en de 3 voorwaarden zijn: (1) ruimte 0 en 2 hebben een relatie, (2) ruimte 1 en 3 hebben geen relatie en (3) ruimte 0 is kleiner dan ruimte 3. De eerste twee voorwaarden kunnen getoetst worden op basis van de niet-diagonaal waarden die de relaties tussen ruimtes duiden. De derde voorwaarde verwijst naar de getallen op de diagonaal die een referentiegrootte geven van de ruimtes zoals bepaald door de topologische matrix. De topologische matrix is een soort minimum grondplan in een beperkt rechthoekig kader.



Figuur 53 - Voorbeeld van de werking van de applicatie (5 ruimtes - 3 voorwaarden)

De ruimtes zijn nu steeds als getal voorgesteld maar het is gemakkelijk zich voor te stellen dat deze getallen effectieve ruimtes weergeven. Dit zal nog duidelijker worden bij het uitwerken van de testcase van één woonunit van het detentiehuis.

1.2. Output

Het resultaat van de applicatie, nadat het aantal ruimtes en de vooropgestelde voorwaarden zijn ingevoerd, is een volledige oplossingenruimte waarbij alle oplossingen die voldoen naar voor komen. Het is aan de gebruiker om met de applicatie te spelen en eerst de belangrijkste

voorwaarden in te geven en nadien pas stapsgewijs meer voorwaarden in te geven tot de oplossingenruimte gereduceerd wordt tot een paar goede oplossingen die dan teruggekoppeld kunnen worden naar het ontwerpen.

De voorwaarde van het al dan niet hebben van een relatie kan heel breed geïnterpreteerd worden. Er kan verondersteld worden dat er ofwel een gemene muur is, ofwel dat er een visueel contact is ofwel dat er een fysiek contact is. Het is aan de gebruiker van de applicatie om voor zichzelf te bepalen hoe er met de output van de applicatie omgesprongen wordt. Het is niet de bedoeling dat dit als letterlijk grondplan wordt beschouwd, maar dat er een terugkoppeling gemaakt wordt op het eigenlijke ontwerp met de eigen ontwerpintenties. Op die manier wordt het een zoektocht naar hoe deze applicatie een grondplan kan genereren dat inpast in het idee van de ontwerper.

Voor de eenvoud van het onderzoek is dit alles voorgesteld in een 2-dimensionele representatie omdat de derde dimensie extra complexiteit zou meebrengen bij het ontwikkelen alsook bij de visualisatie. Het is wel gemakkelijk voor te stellen dat de derde dimensie meer mogelijkheden zal meebrengen, zoals bijvoorbeeld een extra voorwaardenmatrix waarop aangeduid kan worden op welke verdieping welke ruimte zich kan bevinden, alsook een voorwaarde dat een ruimte bijvoorbeeld 2 verdiepingen hoog moet zijn.

1.3. Java-code in processing

Bijlage H.

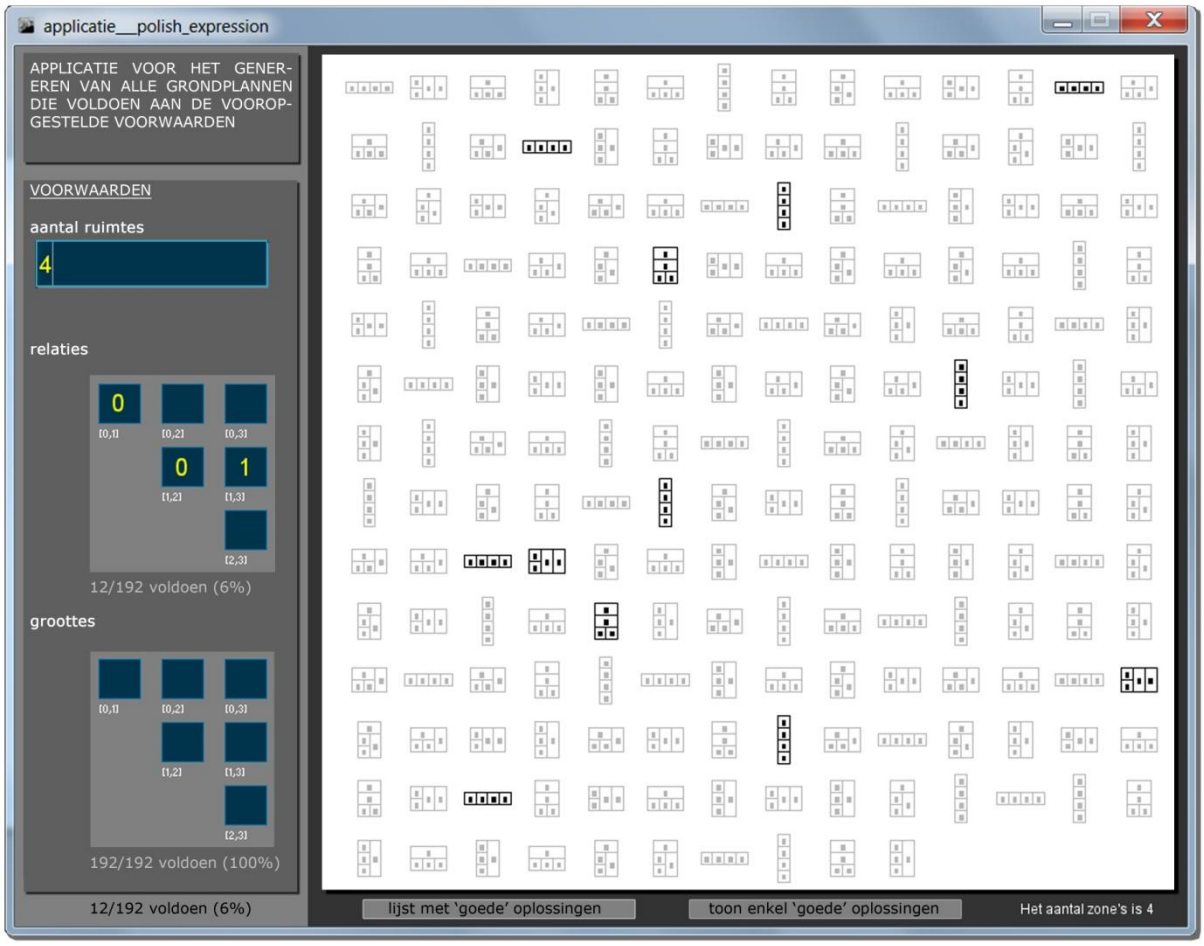
2. Visualisatie (GUI)

Op het moment van dit schrijven is de GUI (graphical user interface) van de applicatie nog in ontwikkeling. Op de bijbehorende presentatie zal de definitieve versie voorgesteld worden. Voor de compleetheid van het onderzoek wordt in deze paragraaf de applicatie voorgesteld met een interface zoals ze eruit zal moeten zien. Dit kan beschouwd worden als de effectieve werking met een vooropgesteld doel van interface.

2.1. Interface

Het doel van de interface is een visualisatie van de achterliggende werking van de applicatie. Zoals op onderstaande afbeelding weergegeven wordt bestaat het scherm uit 2 hoofddelen: links het deel waar de gebruiker zijn verlangde input kan ingeven en rechts de resultaten. In

dit voorbeeld is het aantal ruimtes 4, waardoor de configuraties nog ietwat zichtbaar blijven. Maar bij hogere waarden zal het aantal configuraties exponentieel toenemen waardoor dit scherm zich zal beperken tot een puntenwolk om een indruk te geven van de oplossingenruimte.



Figuur 54 - Het uiteindelijke doel van de applicatie

Het zal aan de gebruiker zijn om de nodige inputwaarden in te geven, zoals het aantal ruimtes en de nodige voorwaarden om het aantal 'goede' oplossingen te reduceren. Het ingeven en veranderen van deze waarden zal de werking van de applicatie duidelijk maken. Deze afbeelding is het beoogde eindresultaat van de applicatie, waar zichtbaar is welke configuraties voldoen aan de vooropgestelde voorwaarden. De 'goede' oplossingen staan zwart gedrukt tussen de grijze andere mogelijke configuraties. Door rechtstreekse interactie kan de gebruiker eenvoudig de voorwaarde aanpassen met als gevolg dat het resultatenscherm ook zal wijzigen.

In volgend subhoofdstuk wordt een voorbeeldsituatie uitgelegd om de werking en het gebruik te duiden.

2.2. Gebruik

Onderstaande opeenvolging van afbeeldingen duidt het gebruik van de applicatie:

(1) De eerste afbeelding is het beginscherm als de applicatie geopend wordt. Hier wordt enkel een inputwaarde voor het aantal ruimtes gevraagd. Hiermee begint de volledige achterliggende werking. Het getal dient groter te zijn dan 1 aangezien voor één ruimte er geen verschillende mogelijke configuraties zijn.

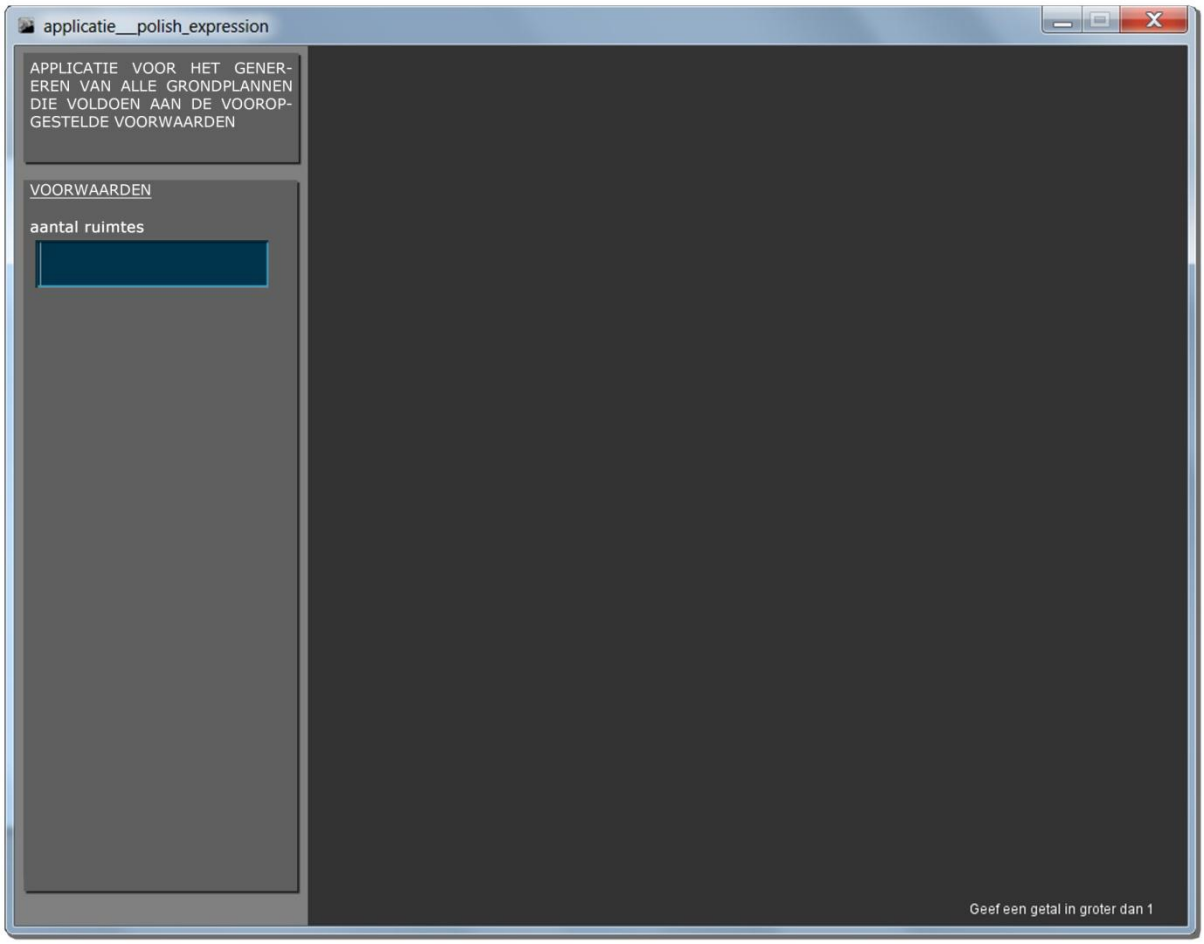
(2) Na het ingeven van het aantal ruimtes zullen links bijkomende mogelijkheden komen om inputwaardes te geven. Bij dit onderzoek is voornamelijk de nadruk gelegd op de nabijheidmatrix om de relaties in te geven. Hierin wordt door de gebruiker de waarde 0 (voor geen relatie) of de waarde 1 (voor wel een relatie) ingevuld waar men een relatie wilt duiden. Maar er is daaronder ook een matrix voorzien om groottes in te geven, dit als voorbeeld om de mogelijkheden van deze tool verder bloot te leggen. Hier geeft de gebruiker de tekens '<' of '>' in om een verhouding in grootte tussen twee ruimtes te duiden. Onder elk vakje van de matrix staat om welke 2 ruimtes het gaat. Bijvoorbeeld: [1,2] = 1 zal een relatie bepalen tussen ruimte 1 en 2. Een ander voorbeeld: [1,2] = < zal bepalen dat ruimte 1 kleiner moet zijn dan ruimte 2, hierbij is de volgorde van de cijfers belangrijk.

(3) Bij het invullen van elke waarde zal steeds de resultatenruimte aangepast worden in het rechtse scherm met de 'goede oplossingen' in het zwart. Hier heeft de gebruiker de mogelijkheid om onderaan in het scherm te kiezen uit twee knoppen.

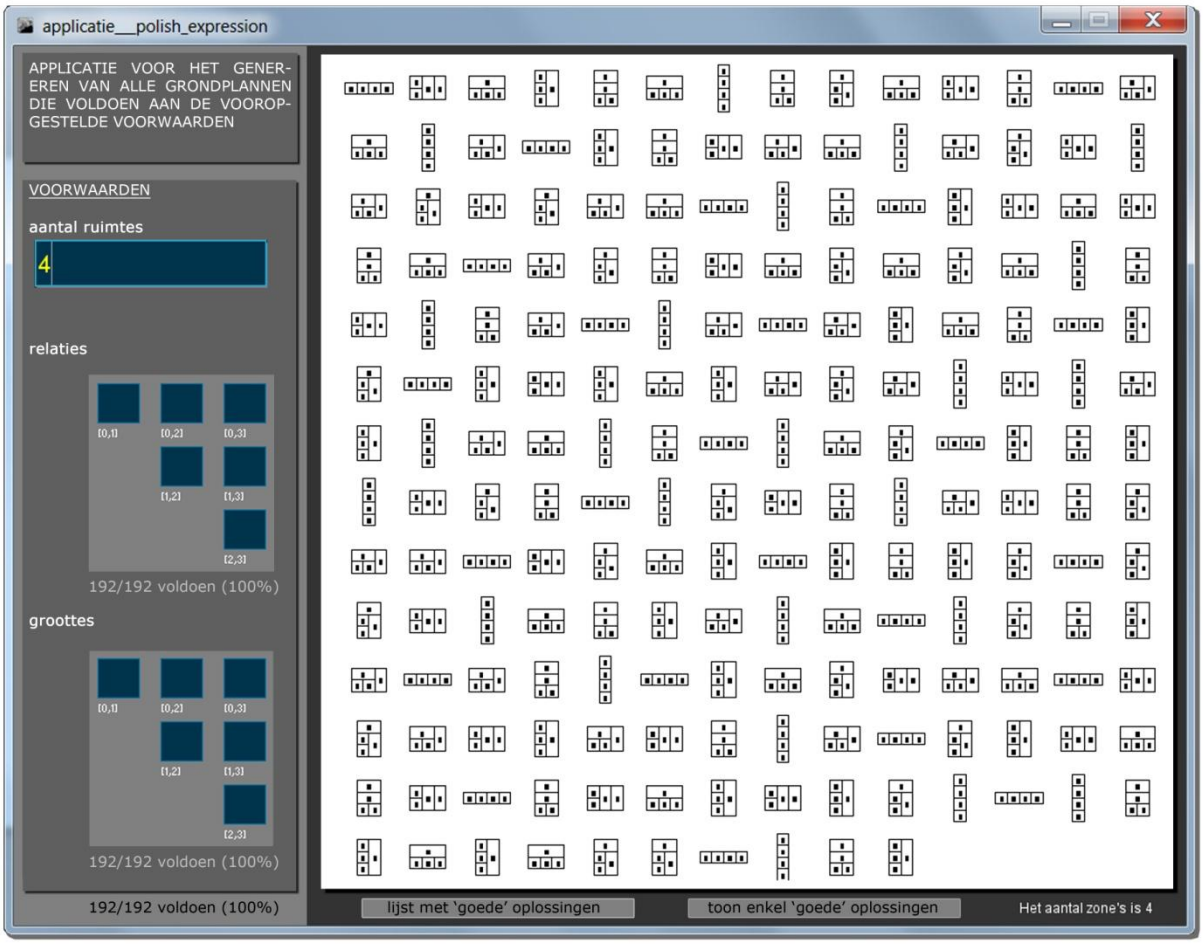
(4) De knop: 'toon enkel goede oplossingen' zal de gebruiker naar een scherm brengen waar enkel de oplossingen getoond worden die voldoen aan de ingegeven voorwaarden. Hier blijft de mogelijkheid bestaan om de voorwaarden aan te passen en zo zal rechtstreeks zichtbaar worden welke oplossingen nog steeds voldoen.

(5) Een tweede optie was om de knop: 'lijst met goede oplossingen' te kiezen. De knop brengt een lijst van 'normalized Polish expressions' naar boven waar alle configuraties in staan die voldoen aan de gestelde voorwaarden, overeenkomstig het scherm van de vorige afbeelding met alle goede oplossingen. Hier kan de gebruiker een keuze maken om een van deze oplossingen te selecteren.

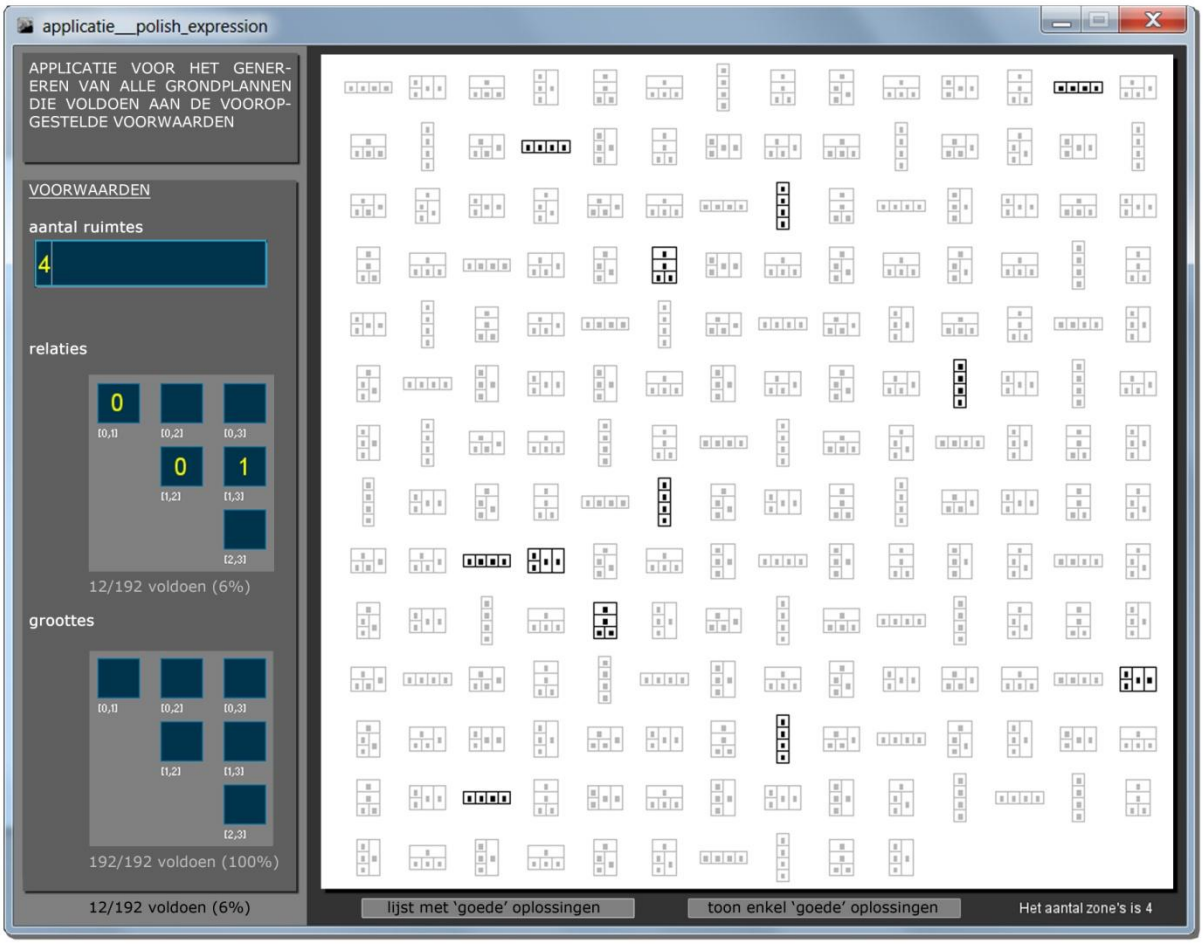
(6) Bij selectie van een 'goede' oplossing uit deze lijst komt de gebruiker in een scherm waar alle informatie die de applicatie over deze oplossing heeft opgeslagen weergegeven wordt. Dit scherm heeft als doel om de achterliggende werking ietwat te duiden om te verklaren waarom dit een goede oplossing is.



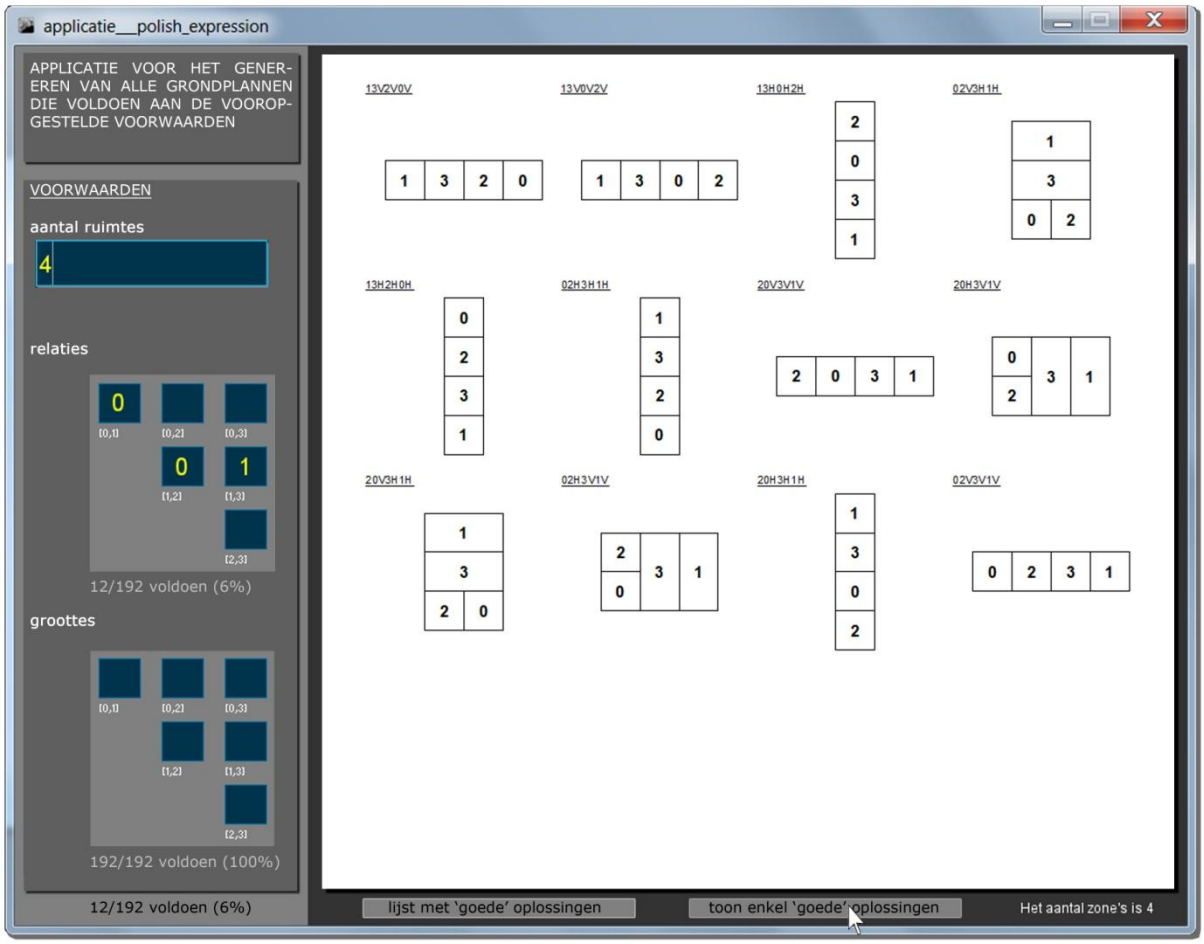
Figuur 55 - GUI(1): hoofdscherm



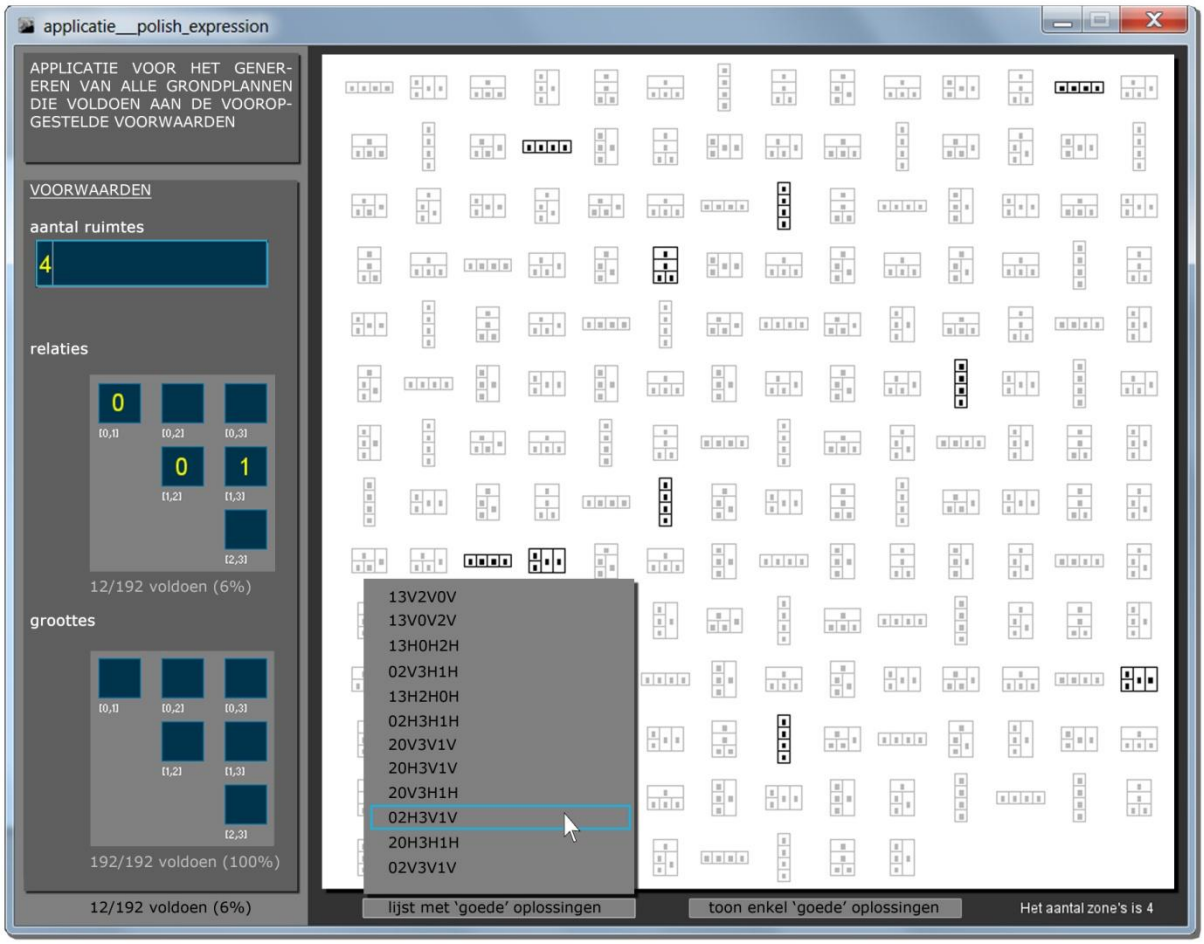
Figuur 56 - GUI(2): alle mogelijke configuraties



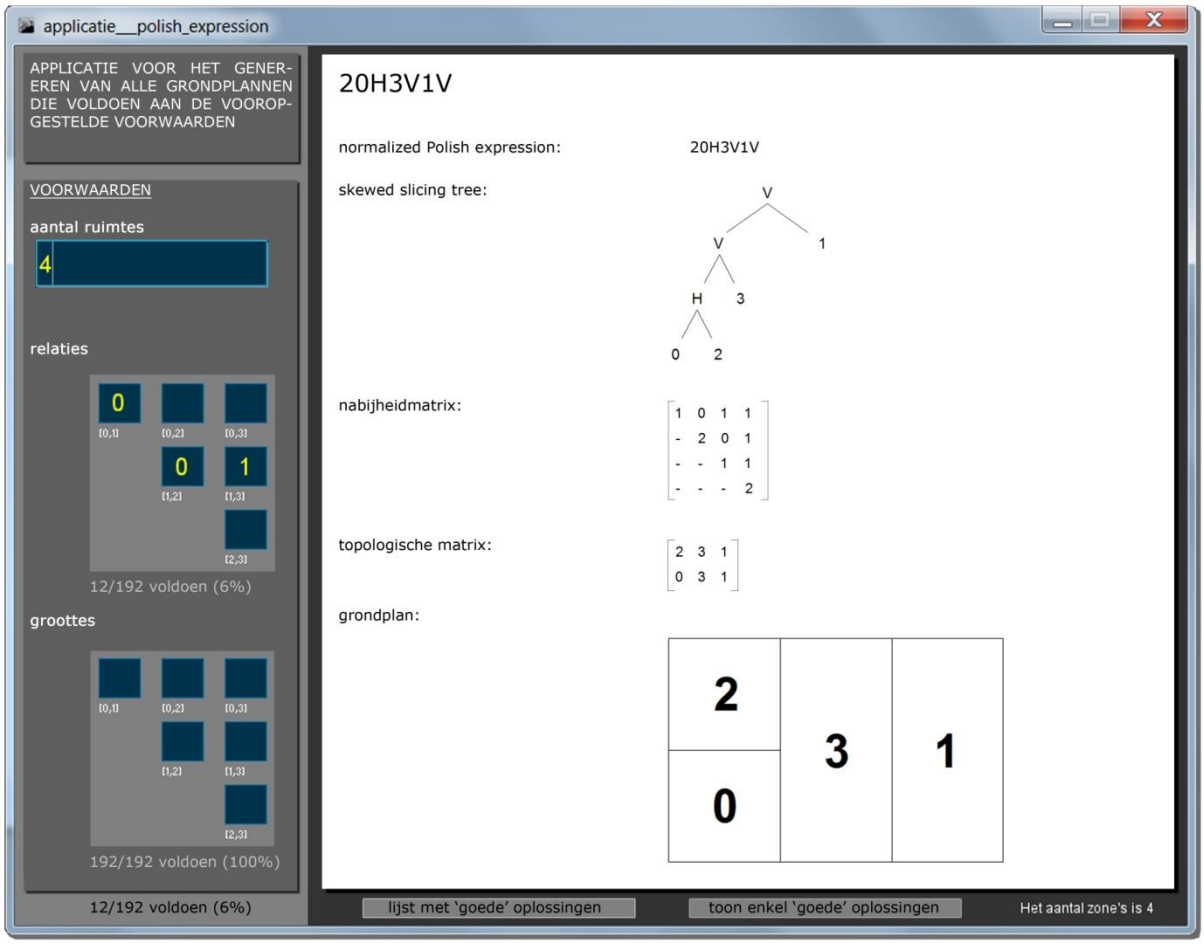
Figuur 57 - GUI(3): configuraties die voldoen



Figuur 58 - GUI(4): toon enkel 'goede' oplossingen



Figuur 59 - GUI(5): lijst met 'goede' oplossingen



Figuur 60 - GUI(6): formulier van één oplossing die voldoet

3. Resultaten

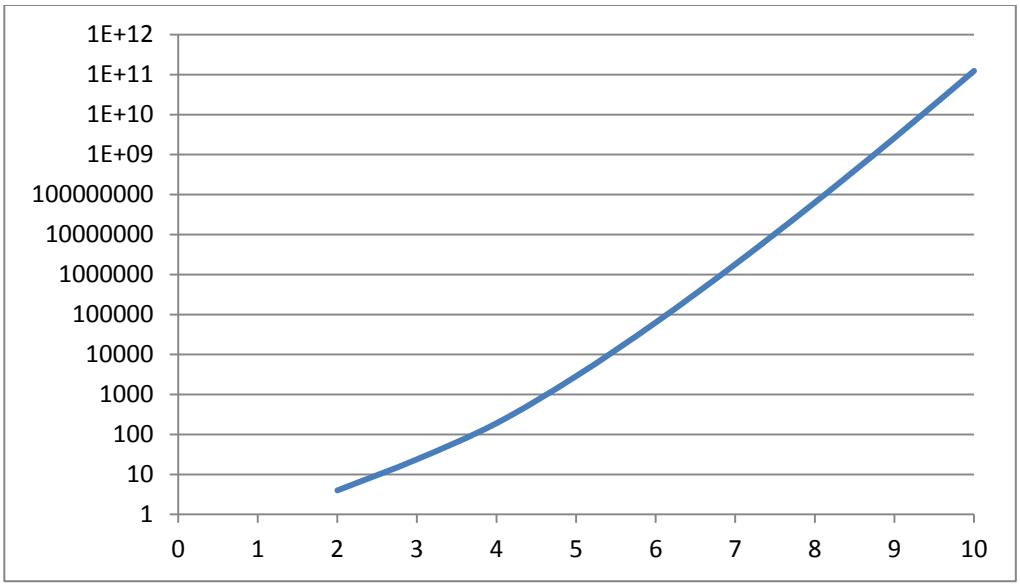
Huidige paragraaf test de applicatie in zijn mogelijkheden. Er wordt gekeken naar hoe groot de oplossingenruimte is en wat de invloed is van de input voorwaarden. De applicatie is geschreven zodat het mogelijk is om dit voor oneindig veel ruimtes te doen, maar uit wat volgt zal blijken dat al vanaf een klein aantal ruimtes het aantal mogelijkheden enorm groot is wat meebrengt dat de applicatie steeds meer tijd nodig zal hebben om de goede oplossingen te vinden en uiteindelijk geen resultaat meer zal geven.

3.1. Alle mogelijkheden

De basis oplossingenruimte, zonder vooropgestelde voorwaarden, wordt weergegeven in onderstaande tabel en grafiek. De weergave loopt tot zo ver het mogelijk is om deze te bepalen. Het was aangewezen om voor deze grafiek enkel dit stuk van de code te gebruiken om op deze manier zo ver mogelijk te kunnen bepalen wat de oplossingenruimte is.

Onderstaande tabel geeft het totaal aantal mogelijkheden weer voor een gegeven aantal ruimtes. Dit totaal wordt bepaald door het aantal mogelijke ruimtelijke configuraties te vermenigvuldigen met het aantal permutaties (P) zodat elke vooropgestelde ruimte kan voorkomen op elke plaats in de configuratie. Na 10 ruimtes was de applicatie voor deze berekening reeds zonder geheugen gevallen. Wat dus voor het verdere gebruik van de applicatie zal betekenen dat het enkel gebruikt zal kunnen worden indien er niet meer dan 10 ruimtes zijn, wat wel voldoende is in het kader van dit onderzoek.

ruimtes	P	configuraties	totaal
1	1	0	0
2	2	2	4
3	6	4	24
4	24	8	192
5	120	24	2880
6	720	88	63360
7	5040	360	1814400
8	40320	1576	63544320
9	362880	7224	2,62E+09
10	3628800	34232	1,24E+11



Grafiek 1 - Alle mogelijke configuraties voor een gegeven aantal ruimtes

3.2. Voorwaarden

Dit gedeelte onderzoekt het gebruik van de applicatie. Hoe de gebruiker invloed kan hebben op de applicatie door ermee te spelen en wat dus de invloed zal zijn van het invoeren en veranderen van de inputvoorwaarden. Er zijn twee keer twee soorten voorwaarden: twee ruimtes hebben een relatie, ze hebben geen relatie, de ene ruimte is groter dan de andere en de ene ruimte is kleiner dan de andere.

3.2.1. *2 ruimtes hebben een relatie*

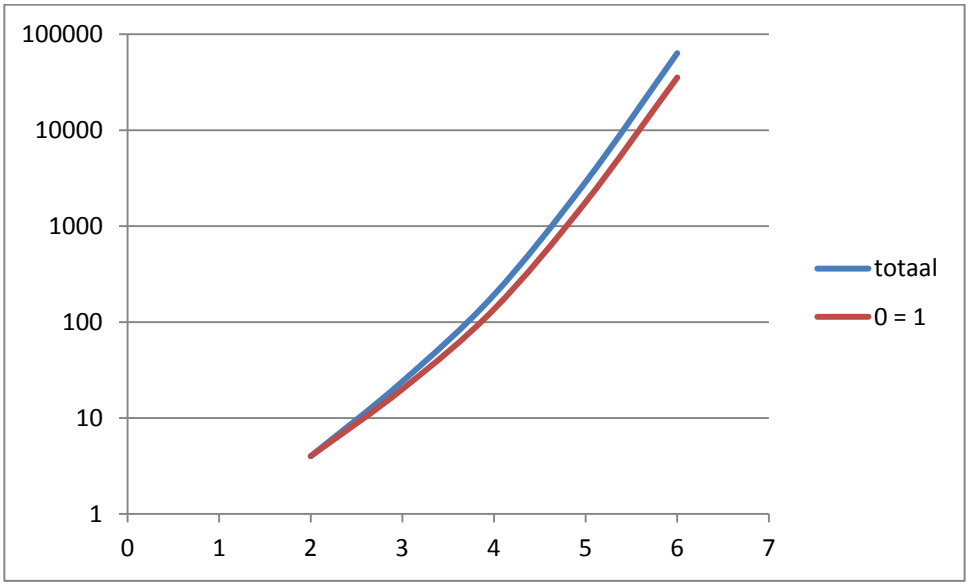
Deze voorwaarde kan ingevoerd worden indien men een relatie wilt tussen 2 ruimtes. Een relatie kan breed geïnterpreteerd worden: het kan gezien worden als een fysieke overgang tussen de 2 ruimtes enerzijds door een deur of anderzijds volledig in elkaar overlopen.

Een andere mogelijkheid is dat de twee ruimtes aan elkaar raken: dit kan een wens zijn indien je 2 ruimtes dicht bij elkaar wilt hebben. In deze applicatie is het nog niet mogelijk om het 'dicht bij elkaar liggen' te kiezen. Maar dit zou makkelijk ook een uitbreiding kunnen zijn, door te zeggen dat twee ruimtes dicht bij elkaar liggen als ze op een afstand van bijvoorbeeld 2 units van elkaar liggen. De slimme gebruiker kan dit in huidige applicatie invoeren als 2 ruimtes die gescheiden worden door 1 gemeenschappelijke ruimte (dus 2 WEL-relaties met een gemeenschappelijke ruimte).

Nog een andere mogelijkheid is het visueel raken van twee ruimtes. Dit is bijvoorbeeld te gebruiken indien één van beide ruimtes een buitenruimte is. Dan geef je eigenlijk onrechtstreeks de ruimtes in die van veel lichtinval kunnen genieten (dit geldt natuurlijk voor alle ruimtes aan de buitenschil indien het totale volume volledig buiten staat). Er kan

hier ook nog een uitbreiding bij bedacht worden, waarbij er kan aangeduid worden welke ruimte zeker aan de buitenomgeving moet raken. Om bijvoorbeeld licht en zicht in de leefruimte te hebben, maar ook voor verluchting in de badkamer bijvoorbeeld.

Op onderstaande grafiek kan gezien worden dat het aantal oplossingen in kleine mate afneemt bij één maal deze voorwaarde.



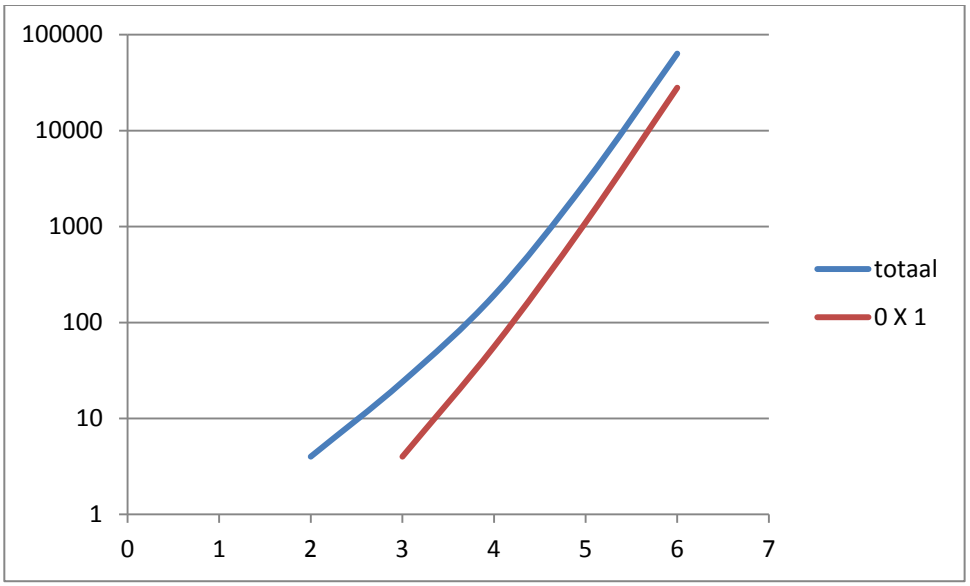
Grafiek 2 - 'Wel'-relatie

3.2.2. *2 ruimtes hebben geen relatie*

Deze voorwaarde kan worden ingevoerd indien men geen relatie wilt tussen 2 ruimtes. Dit kan verschillende redenen hebben. Een voorbeeld is het niet hebben van een relatie tussen de leefruimte en de wc. Dit om akoestische redenen maar ook om geurhinder te vermijden. Anderzijds mogen ze ook niet te ver uit elkaar liggen. Hier ligt ook weer een mogelijkheid om de applicatie uit te breiden: naast een niet-relatie kan er ook een 'dicht bij elkaar'-relatie worden ingeven.

Ook kan de NIET-relatie tussen 2 ruimtes ingevoerd worden om privacyredenen, zoals bijvoorbeeld tussen de leefruimte en de slaapkamer aangezien de leefruimte een ietwat publieke ruimte is, waar de slaapkamer privaat bedoeld wordt.

Onderstaande grafiek geeft de invloed van één maal deze voorwaarde weer. Als deze met bovenstaande voorwaarde vergeleken wordt, kan gezien worden dat deze voorwaarde strenger is.



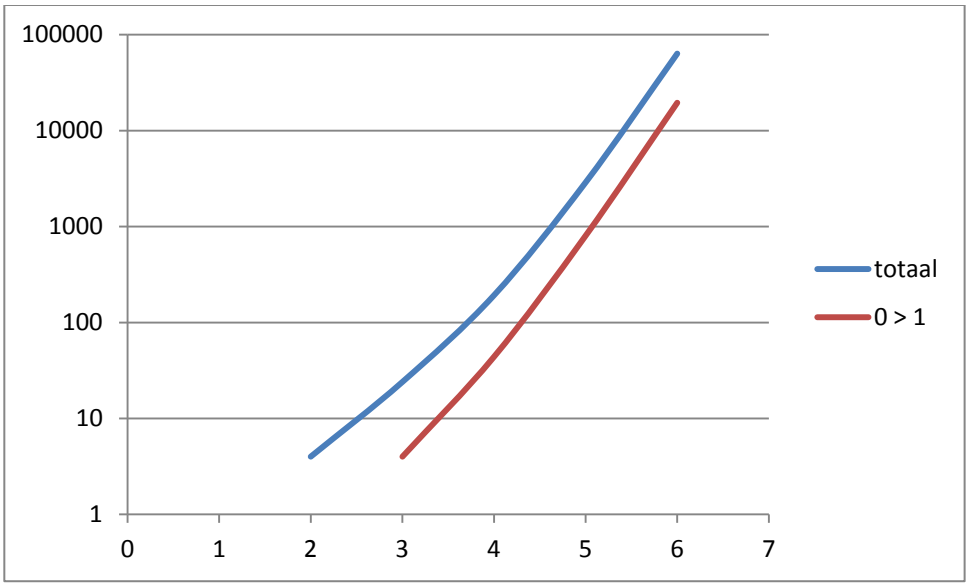
Grafiek 3 - 'Niet'-relatie

3.2.3. *1 ruimte is groter dan een andere*

Het willen groter of kleiner hebben van een ruimte dan een andere heeft vrij evidente redenen. Enerzijds puur praktisch omdat men meer meubels plaatst in een leefruimte dan in een wc, maar anderzijds ook om licht, lucht en ruimte te hebben in ruimtes die veel gebruikt worden. Mogelijke voorbeelden zijn om de slaapkamer groter te maken dan de badkamer, of om de leefruimte groter te maken dan de keuken.

Onderstaande grafiek geeft de invloed van één maal deze voorwaarde weer. Het is duidelijk dat deze voorwaarde strenger is dan bovenstaande relationele voorwaarden. Dit kan verklaard worden doordat de applicatie minimum oppervlaktes genereert waardoor in de meest basische vorm elke ruimte dezelfde grootte heeft. Dit heeft als oorzaak dat de applicatie opgebouwd is zonder de invloed van geometrie en dus heeft een geometrische voorwaarde een drastisch effect op de oplossingenruimte. Ook is het de bedoeling dat na de applicatie er een slimme interpretatie van het gegenereerde grondplan gedaan wordt. Hier is het makkelijk om groottes van ruimtes nog aan te passen zonder de relaties te veranderen.

Een uitbreiding op deze applicatie zou kunnen zijn dat het strenge ‘groter dan’ vervangen wordt door het minder specifieke ‘groter dan of gelijk aan’. Dit heeft als gevolg dat de oplossingenruimte vergroot wordt tot het complement van de oplossingenruimte met de strengere ‘groter dan’ voorwaarde.



Grafiek 4 - 'Groter dan' voorwaarde

3.2.4. *1 ruimte is kleiner dan de andere*

Deze voorwaarde genereert dezelfde oplossingenruimte als de 'groter dan' voorwaarde, aangezien elke ruimte door de permutatie eens kan voorkomen op elke plaats.

3.2.5. *Combinatie van één keer elk van de 4 voorwaarden*

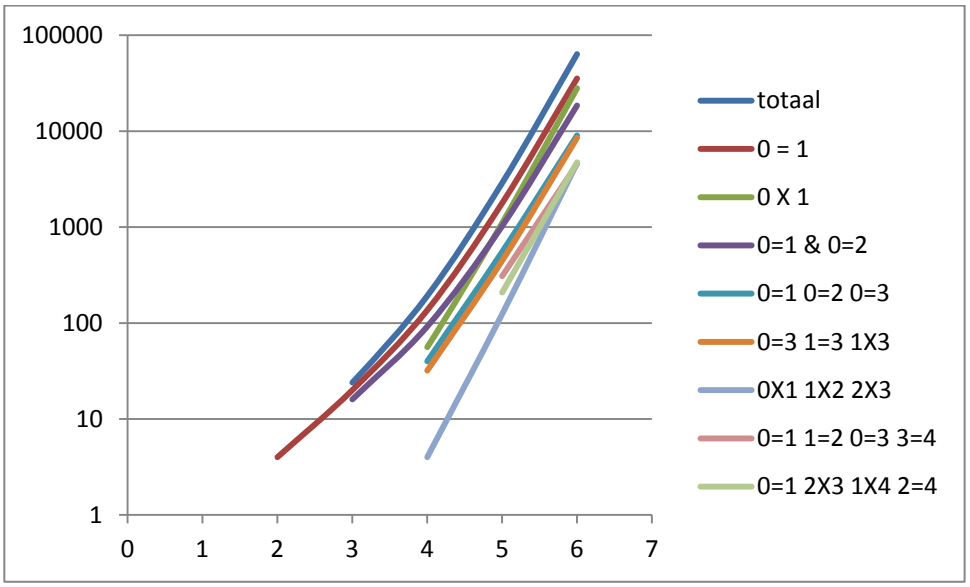
Onderstaande tabel toont aan dat bij invoer van de 4 verschillende voorwaarden er pas vanaf 6 ruimtes nog oplossingen overblijven. Dit komt omdat er bij minder dan 6 ruimtes te weinig ruimtes zijn om zoveel verschillende soorten voorwaarden te hebben. Hieruit kan besloten worden dat er voorzichtig moet worden omgesprongen met het invoeren van voorwaarden. Een stapsgewijze opbouw van eerst de meest belangrijke voorwaarden is aangeraden.

N	0=1 0X2	0<1 0>2	0=1 0X2 0<1 0>2
1	/	/	/
2	/	/	/
3	4	0	0
4	44	0	0
5	752	60	0
6	16872	1680	264

3.2.6. *Meer voorbeelden*

Onderstaande tabel met aantal mogelijke oplossingen geeft een reeks voorwaarden combinaties weer en het bijbehorende overgebleven aantal oplossingen. Een WEL relatie wordt aangeduid door een '='-teken en een NIET relatie door een X.

N	0 = 1	0 X 1	0=1 & 0=2	0=1 & 0X2	0=1 0=2 0=3	0=3 1=3 1X3	0X1 1X2 2X3	0=1 1=2 0=3 3=4	0=1 2X3 1X4 2=4
1	/	/	/	/	/	/	/	/	/
2	4	0	/	/	/	/	/	/	/
3	20	4	16	4	/	/	/	/	/
4	136	56	92	44	40	32	4	/	/
5	1776	1104	1024	752	552	448	124	308	208
6	35328	28032	18456	16872	9024	8520	4640	4552	4736



Grafiek 5 - Combinatie van meer voorbeelden

Er kan besloten worden dat een NIET-relatie meer dwingend is dan een WEL-relatie. Het is daarentegen wel zichtbaar op de grafiek en in de tabel dat voor een kleiner aantal ruimtes deze NIET-relatie nog meer dwingend is: voor 4 ruimtes zijn er nog 56 oplossingen voor één NIET-relatie en 92 voor twee WEL-relaties, waar bij 6 ruimtes het eerste getal groter zal zijn dan het tweede. Hetzelfde kan gezien worden bij het voorbeeld van 3 NIET-relaties ten opzichte van 4 WEL-relaties, maar dan bij een hoger aantal ruimtes. Hieruit kan dus besloten worden dat een kleiner aantal NIET-relaties steeds minder oplossingen zal geven dan een hoger aantal WEL-relaties, tot deze aantallen beduidend lager worden dan het aantal ruimtes want dan geldt de eenvoudige regel: hoe meer voorwaarden hoe minder oplossingen. De gebruiker zal hier dus met meer voorwaarden een preciezere oplossing bekomen. Het is wel aan te raden om niet onmiddellijk te veel voorwaarden te geven, op deze manier kan je de applicatie zien werken en wordt de achterliggende werking begrepen.

3.3. Besluit

Het belangrijkste besluit is dat de relationele voorwaarden voor een betere werking van de applicatie zullen zorgen, aangezien ook dit de primaire doelstelling was bij het ontwikkelen van de applicatie. De grootte van de ruimtes is een bijkomende factor geworden na het opstellen van de topologische matrix, om als voorbeeld te stellen hoe het mogelijk is om op de nabijheidmatrix nog meer informatie te plaatsen. Hierbij kan verwezen worden naar hoofdstuk 2 waar de nabijheidmatrix van het BIM model werd afgeleid. Hier werd het voorbeeld van deuropeningen aangehaald die ook als extra informatie op de nabijheidmatrix geplaatst zou kunnen worden om het verschil tussen een fysieke en een andere relatie te duiden.

Uit de resultaten blijkt ook dat de applicatie slechts snel oplossingen kan genereren tot 6 ruimtes. Hierbij duurt het slechts enkele seconden. Dit aantal kan al een goede weergave bieden voor een realistische configuratie. De applicatie kan effectief werken tot oneindig veel ruimtes maar per stap van 1 ruimte extra zal de wachttijd exponentieel oplopen. In de toekomst, buiten het kader van dit onderzoek, kan deze applicatie nog geoptimaliseerd worden voor een snellere werking door eventueel gebruik te maken van een sneller zoekalgoritme. Als in de toekomst ook de 3-dimensionele representatie in de applicatie ingewerkt zou worden zou dat ook voor vertraging zorgen waardoor optimalisatie nodig zal zijn voor meer mogelijkheden.

Ten slotte moet nogmaals benadrukt worden dat deze resultaten slechts een suggestie zijn van een grondplan met tot nu toe maar enkele soorten parameters. Het is dus nog steeds aan de ontwerper om hiermee iets nuttig te doen en de terugkoppeling te maken naar de eerste ontwerpintenties. Het nut van deze applicatie is een suggestie te doen om de ontwerper bij te staan in een vroeg stadium van het ontwerpproces.

4. Testcase: Detentiehuis

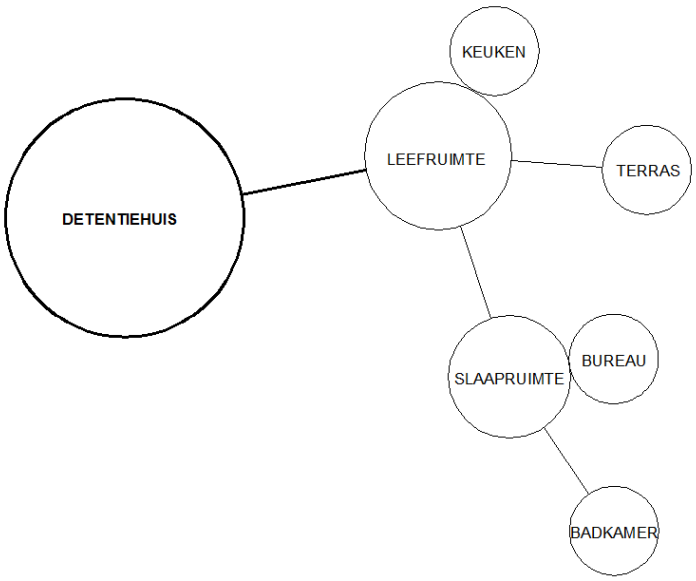
In deze paragraaf wordt een terugkoppeling gemaakt naar de case van het detentiehuis. In het bijzonder wordt, zoals vooropgesteld in de vorige hoofdstukken, een klein deel van het detentiehuis onderzocht als representatief deel voor het volledige ontwerp. Dit gedeelte is één woonunit. Dit is een individuele verblijfsruimte van een gedetineerde met alle private voorzieningen die nodig zijn om te kunnen leven, aangezien het op bepaalde momenten, zoals ‘s avonds, de bedoeling is dat de gedetineerden enkel in hun individuele ruimte verblijven zodat de controle gemakkelijker is.

4.1. Individuele verblijfsruimte

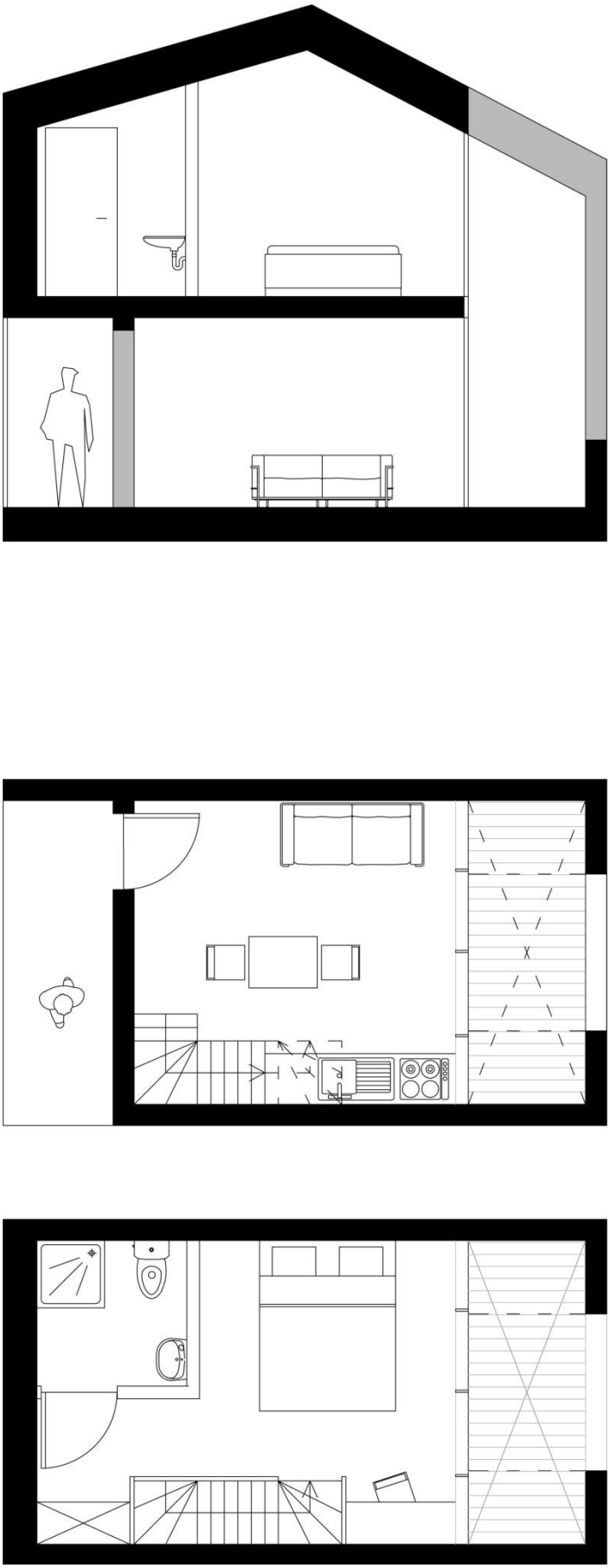
De individuele verblijfsruimte zoals ontworpen in de case bestaat uit een leefruimte met kitchenette waaraan ook een terras is, een slaapruimte met bureau en een badkamer. Onderstaande tabel geeft de oppervlaktes weer; dit zou als secundaire inputvoorwaarde voor de applicatie kunnen dienen. Secundair omdat de groottes zoals eerder besproken in de resultaten een minder goede invloed hebben op de applicatie en snel veel mogelijkheden schrappen.

Individuele verblijfsruimte:	leefruimte + keuken	15	m²
	terras	5,3	m²
	badkamer	3,6	m²
	slaapkamer + bureau	15,8	m²
	totaal	39,7	m²

Om de relaties te duiden die ontworpen werden in de case wordt verwezen naar onderstaande afbeeldingen. Hier kan gezien worden hoe de ruimtelijke configuratie van de individuele verblijfsruimte in grondplannen en doorsnede is. Het is, zoals eerder besproken, op dit moment, met deze applicatie niet mogelijk om in twee verdiepingen te werken. Daarom wordt voor deze kleinere testcase een gelijkaardig schema gebruikt, maar wel op één verdieping. Hiervoor werd een graaf opgesteld die de relaties voorstelt. Deze kan vervolgens ook vertaald worden naar een eigen nabijheidmatrix die voor de input in de applicatie gebruikt zal kunnen worden.



Figuur 61 - Case detentiehuis: graaf van individuele verblijfsruimte



Figuur 62 – Case detentiehuis: plannen individuele verblijfsruimte

4.2. Testen van de applicatie

4.2.1. *Input*

De voornaamste input is het aantal ruimtes, in dit geval is dat 6 ruimtes: leefruimte, keuken, terras, slaapruimte, bureau en badkamer.

Vervolgens wordt op basis van de graaf een nabijheidmatrix opgesteld die alle relaties weergeeft. Het is belangrijk om in de eerste fase van het gebruik van de applicatie eerst enkel de belangrijkste relaties in te voeren om zo geleidelijk aan steeds meer te verfijnen naar die oplossingen die het meest voldoen aan de gewenste voorwaarden. Daarom wordt in onderstaande afbeelding een volledige nabijheidmatrix opgesteld en in vet aangeduid welke relaties het belangrijkst zijn. In bovenstaande graaf is het terras enkel verbonden met de leefruimte, dit is omdat deze graaf enkel de fysieke verbindingen legt. Maar zoals eerder besproken, kan men in de nabijheidmatrix verschillende soorten relaties leggen. Een voorbeeld hiervan is de visuele relatie tussen het terras en de slaapruimte, wat zeker een even belangrijke relatie is. Maar voor het testen van de applicatie zal eerst de nadruk gelegd worden op de fysieke relaties, om dan achteraf te kijken of het nog mogelijk is om deze andere relatie ook te leggen.

0: leefruimte	-	**1**	**1**	**1**	0	**0**
1: keuken		-	0	0	0	0
2: terras			-	**(1)**	(1)	0
3: slaapruimte				-	1	**1**
4: bureau					-	0
5: badkamer						-

Figuur 63 - Testcase: nabijheidmatrix

Zoals zichtbaar is op de afbeelding is het belang van de relaties gelegd op het verbinden van de leefruimte met de keuken, het terras en de slaapruimte. Een belangrijke NIET-relatie is gelegd tussen de leefruimte en de badkamer. Het getal ‘1’ vetgedrukt, tussen haakjes is die visuele relatie zoals hierboven besproken, die toch ook wel belangrijk is. Vervolgens is het ook nog belangrijk dat de slaapkamer en de badkamer wel met elkaar verbonden zijn.

In principe is er ook nood aan een relatie tussen de leefruimte en de rest van het detentiehuis. In deze applicatie is zo een voorwaarde nog niet mogelijk, maar het is makkelijk te bedenken dat er een voorwaarde zou kunnen zijn waar er bij een ruimte een vakje wordt aangevinkt waardoor deze ruimte zeker aan de rand van het grondplan komt te

liggen waardoor verbinding met een buitenstaande ruimte mogelijk is. Dit kan makkelijk getest worden op basis van de topologische matrix.

De rest van de voorwaarden die in deze nabijheidmatrix zijn opgenomen zijn niet bindend voor het ontwerp en worden voorlopig ook niet ingevuld om een ruime oplossingenruimte te verkrijgen die nog veel vrijheid toelaat. Als blijkt dat deze ruimte toch te groot is kunnen we nog zo een relatievoorwaarde toevoegen of overgaan op een groottevoorwaarde die wel heel streng zal optreden zoals bijvoorbeeld: de badkamer moet kleiner zijn dan de slaapkamer.

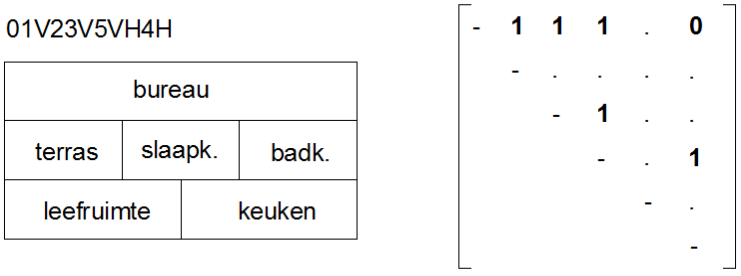
De nabijheidmatrix toont hier ook hoe je moet omgaan met de applicatie, aangezien alle ruimtes steeds als nummer beschreven worden. Het is de bedoeling om vooraf aan elk nummer een ruimte toe te wijzen om zo achteraf de grondplannen correct te kunnen lezen.

4.2.2. *Output*

Na het invoeren van de vetgedrukte voorwaarden wordt de oplossingenruimte van 63360 mogelijke configuraties bij 6 ruimtes gereduceerd tot 1328 mogelijke configuraties die voldoen aan deze voorwaarden. Een voorbeeld van één mogelijkheid wordt weergegeven in onderstaande afbeelding. Hier kan gezien worden hoe aan alle voorwaarde voldaan werd. Het is ook onmiddellijk zichtbaar dat dit geen werkend grondplan is dat letterlijk overgenomen kan worden, maar het is wel een tool om mee aan de slag te gaan. De ontwerper krijgt onmiddellijk zin om hierop transformaties door te voeren om een werkend grondplan te creëren dat past in het volledige ontwerp.

Het is interessant te zien hoe dit grondplan, dat puur bepaald is door de relaties tussen ruimtes, toch een bepaalde diepgang heeft. Zo wordt, door het invoeren van een terras als een van de inputruimtes, de ingesloten slaapruimte toch aan de buitenruimte verbonden en krijgt ze op deze manier licht, lucht en zicht.

Er is ook te zien hoe op toevallige basis de badkamer en de keuken verbonden zijn, wat een secundaire voorwaarde zou kunnen geweest zijn om leidingen eenvoudig weg te werken. Zo worden ook nog relaties ontdekt waaraan de ontwerper in eerste instantie niet gedacht zou hebben.

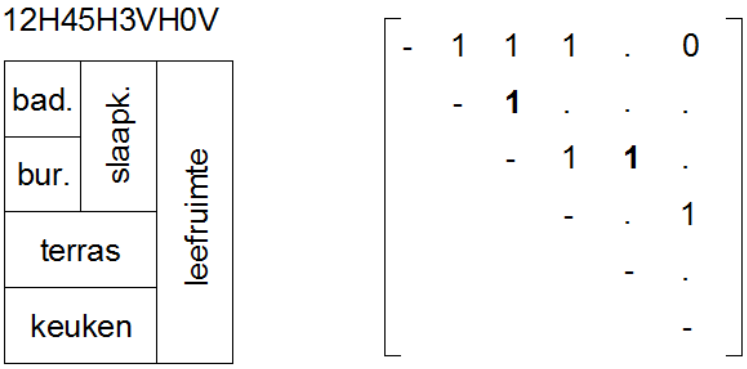


Figuur 64 - Testcase: test 1

Dit duidt het interessante van het bladeren doorheen alle mogelijke ontwerpen die nu al voldoen. Toch is deze 1328 nog een redelijk hoog aantal om al goed te kunnen selecteren. Er worden nog meer relaties ingevoerd om een kleiner aantal te bekomen.

Wanneer ook nog de visuele relatie tussen het terras en de bureauruimte erbij genomen wordt, wat ook wel gewenst is, wordt de oplossingenruimte gereduceerd tot 544 mogelijke configuraties. Bovenstaande afbeelding voldeed hier op toevallige basis ook al aan. Dit komt omdat de relaties die niet ingegeven worden als vrij beschouwd worden met als gevolg dat alle gevallen worden weergegeven.

Na het invoeren van een nieuwe voorwaarde die stelt dat de keuken ook een verbinding heeft met het terras blijven nog 148 mogelijkheden over. Onderstaande afbeelding geeft 1 van de 148 mogelijke oplossingen weer. Hier is zichtbaar dat de badkamer en de keuken nu niet meer naast elkaar liggen, dit was bij de vorige test een toevalligheid. Maar wanneer nu die voorwaarde zou worden ingevoerd blijkt dat dit niet meer mogelijk is; er blijven 0 oplossingen over, wat betekent dat de combinatie van al die voorwaarden een onmogelijke oplossing is.



Figuur 65 - Testcase: test 2

Als test met de groottes van de ruimte werd volgende voorwaarde ingegeven: de leefruimte moet groter zijn dan de badkamer. Hieruit volgt dat het aantal oplossingen gereduceerd wordt van 63360 naar 19536. Als dit gecombineerd wordt met de 148 mogelijkheden van de relatievoorwaarden wordt het totale resultaat gereduceerd naar 46 oplossingen, waarvan bovenstaande afbeelding ook een oplossing is.

4.2.3. *Invloed gedetineerden*

Voor het ontwikkelen van het detentiehuis zou deze applicatie gebruikt kunnen worden in het kader van een onderzoek. Men zou elke gedetineerde kunnen laten ‘spelen’ met de applicatie om zo tot een reeks resultaten te komen waarbij al deze mogelijkheden geanalyseerd kunnen worden om zo de meest optimale en wenselijke individuele

verblijfsruimte te creëren. Op deze manier kunnen de gedetineerden ook een impact hebben op de vorm van detentiehuizen in de toekomst, wat één van de uitgangspunten is van Hans Claus.

4.3. Besluit

Uit deze testen blijkt dat voor een optimaal gebruik van de applicatie de voorwaarden op voorhand goed gedefinieerd moeten worden. Het is dan ook aangewezen om een onderscheid te maken tussen primaire en secundaire voorwaarden om zo stapsgewijs de applicatie te kunnen gebruiken en geleidelijk aan nieuwe voorwaarden in te voeren.

Zoals eerder bleek kunnen de tussenresultaten ook nieuwe inzichten brengen waaraan op voorhand niet gedacht werd. Hierin zit ook net de sterkte van deze applicatie.

Ten slotte is het belangrijk dat de ontwerper een slimme terugkoppeling maakt van de output van de applicatie naar de realiteit van het ontwerp.

6. CONCLUSIE

Een eerste algemene conclusie na het lezen van verschillende teksten en het bijwonen van de ‘smart geometry workshop’ is dat er nog steeds meer vernieuwende bijdragen gebeuren aan het onderzoek rond ‘generative design’ en parametrisch ontwerpen. Dit is hoopvol voor de toekomst dat er steeds meer mogelijkheden zullen komen.

Vervolgens kan besloten worden uit het gevoerde onderzoek dat het gebruik van ‘building information models’ (BIM) uitgebreide mogelijkheden biedt om een ontwerp te analyseren en om binnen dit model te ontwerpen. Het is een geïntegreerd model waar alle partijen van het bouwproces hun invloed op kunnen hebben.

Een ontwerp zal nooit volledig kunnen gerationaliseerd worden zodat een computer het kan genereren. Uit het onderzoek is ook gebleken dat vertrekken van een bestaand ontwerp ook zoveel moeilijker is omdat zo een ontwerp doordrongen is van keuzes en beslissingen die niet altijd even transparant zijn. Het principe van ‘shape grammar’ is hier een goede manier om zoiets te bereiken, maar zoals gezien in het onderzoek is zelfs van een eenvoudig rechthoekig ontwerp de ‘shape grammar’ al zeer uitgebreid.

Steeds zal elke ontwerper bij een nieuw ontwerpopdracht vertrekken van een graaf waar de relaties tussen de ruimtes centraal staan. Dit uitgangspunt was onderwerp van het vervolg van het onderzoek dat belangrijk was voor het uitwerken van de applicatie. Deze applicatie

laat de gebruiker toe om eigen relatievoorwaarden in te voeren en zo tot alle mogelijke rechthoekige configuraties te komen die voldoen aan die voorwaarden. Deze configuraties kunnen gezien worden als een deelgrondplan van een volledig ontwerp en moeten vervolgens slim teruggekoppeld worden naar het effectieve ontwerp.

Het onderzoek naar de applicatie gebeurde in verschillende programma's en op verschillende manieren. Het is aangewezen om steeds te streven naar de meest optimale manier zodat een applicatie snel en gebruiksvriendelijk kan werken. Processing was hiervoor een goed alternatief voor grasshopper3d omdat het eenvoudig een 'graphical user interface' opbouwt. Ook werd naar processing overgeschakeld omdat de geometrische bewerking, die grasshopper3d rijk maakt, niet meer nodig was om enerzijds een snelle werking te bekomen, maar anderzijds ook om zo weinig mogelijk vast te hangen aan geometrische voorwaarden.

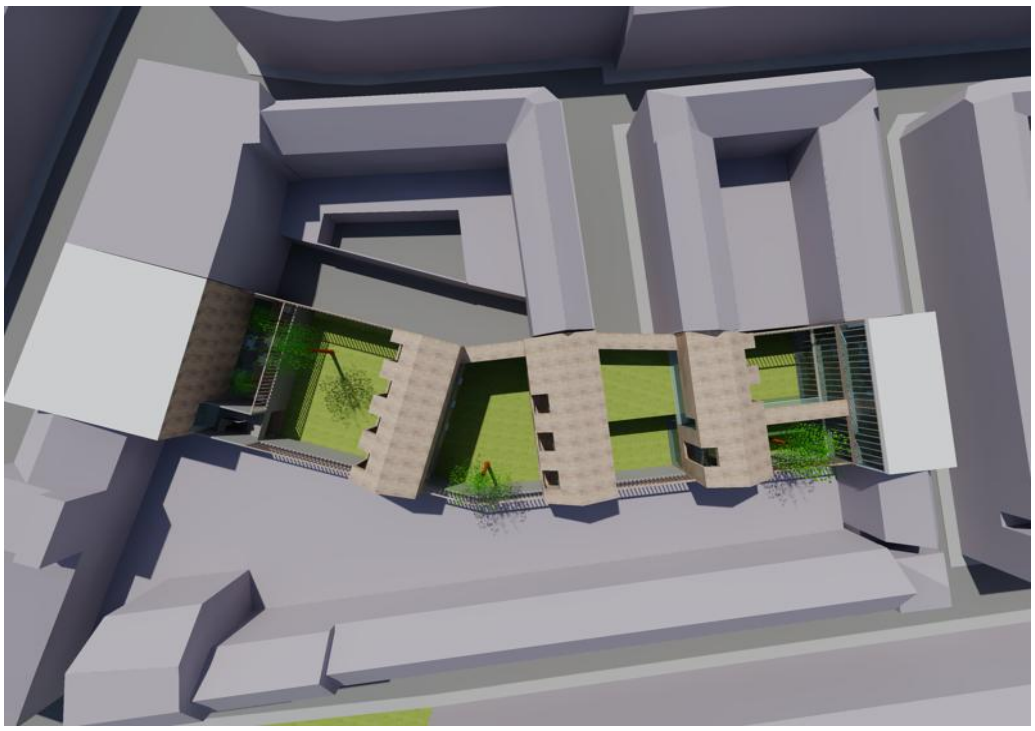
Tot slot kan besloten worden dat de applicatie een goede basis is voor een rijke applicatie die steeds meer mogelijkheden kan integreren. Zelfs nu al is deze basis een interessante tool waarmee elke ontwerper graag zou spelen om alternatieve grondplannen te genereren. Zoals zichtbaar bij de testcase geven de resultaten alvast een bevredigend resultaat wat het bij elke ontwerper zou moeten doen kriebelen om hiermee verder aan de slag te gaan.

De nadruk moet hierbij nogmaals gelegd worden op het feit dat de applicatie een suggestie biedt voor een grondplan en het aan de gebruiker is om die suggestie intelligent te vertalen naar het effectieve ontwerp. Vaak ligt hier het grote punt van kritiek op deze generatieve ontwerpen: dat het de creatieve geest in het gedrang brengt. Maar zolang de computer niet kan denken zoals de mens is steeds een intelligente terugkoppeling naar de realiteit vereist.

7. BIJLAGEN

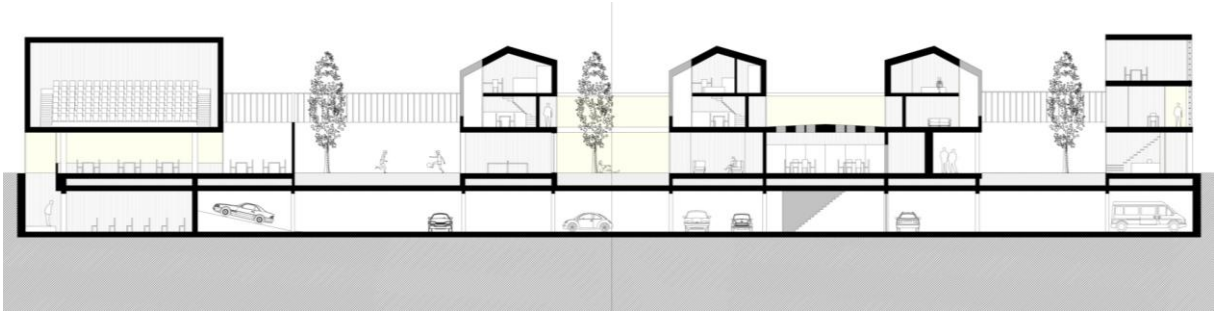
BIJLAGE A: Detentiehuis plannen en beelden (H1-4.2.3)

Totaalbeeld:



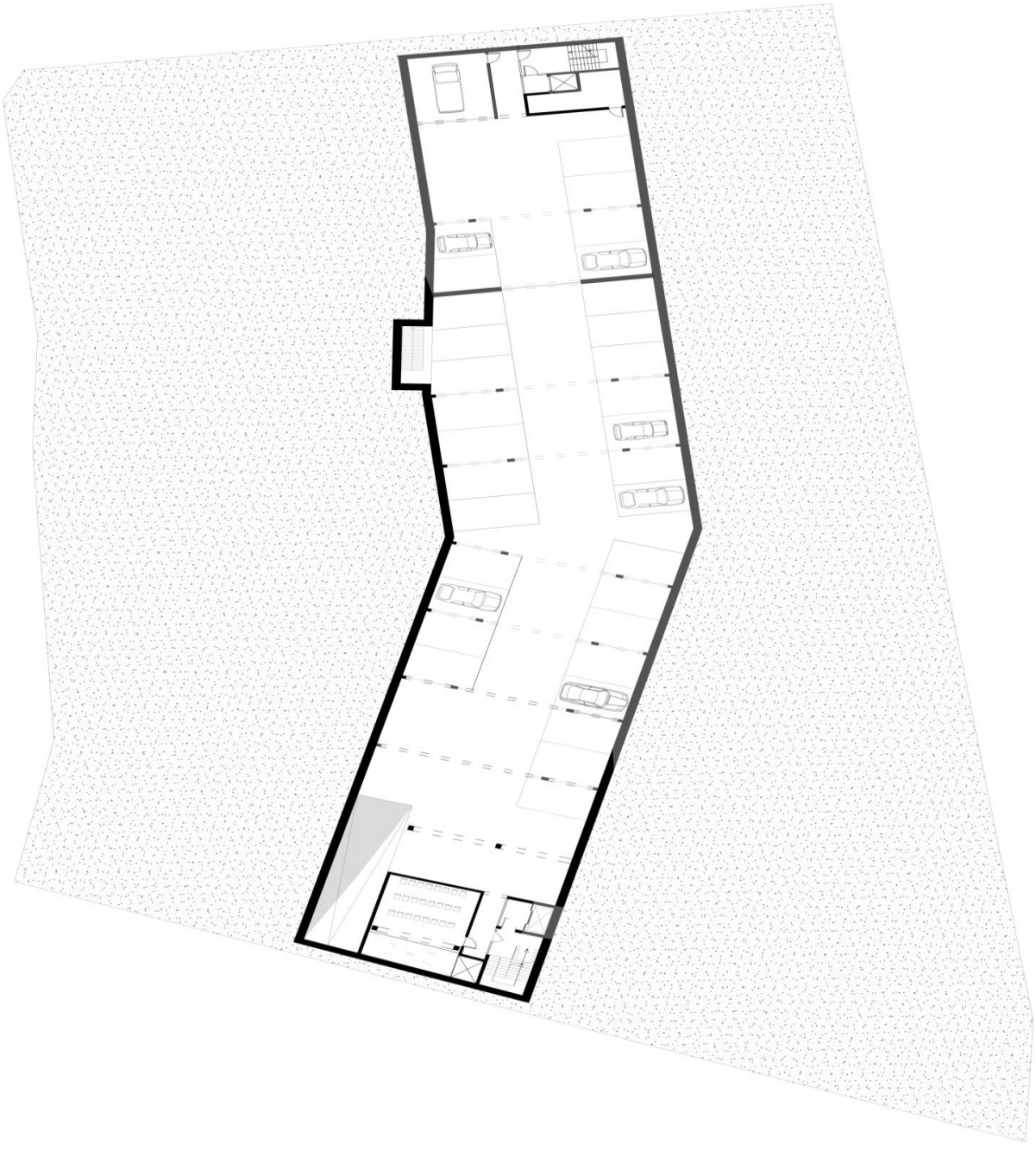
Veelheid aan buitenruimtes en binnenruimtes:

Relaties tussen boven en onder:



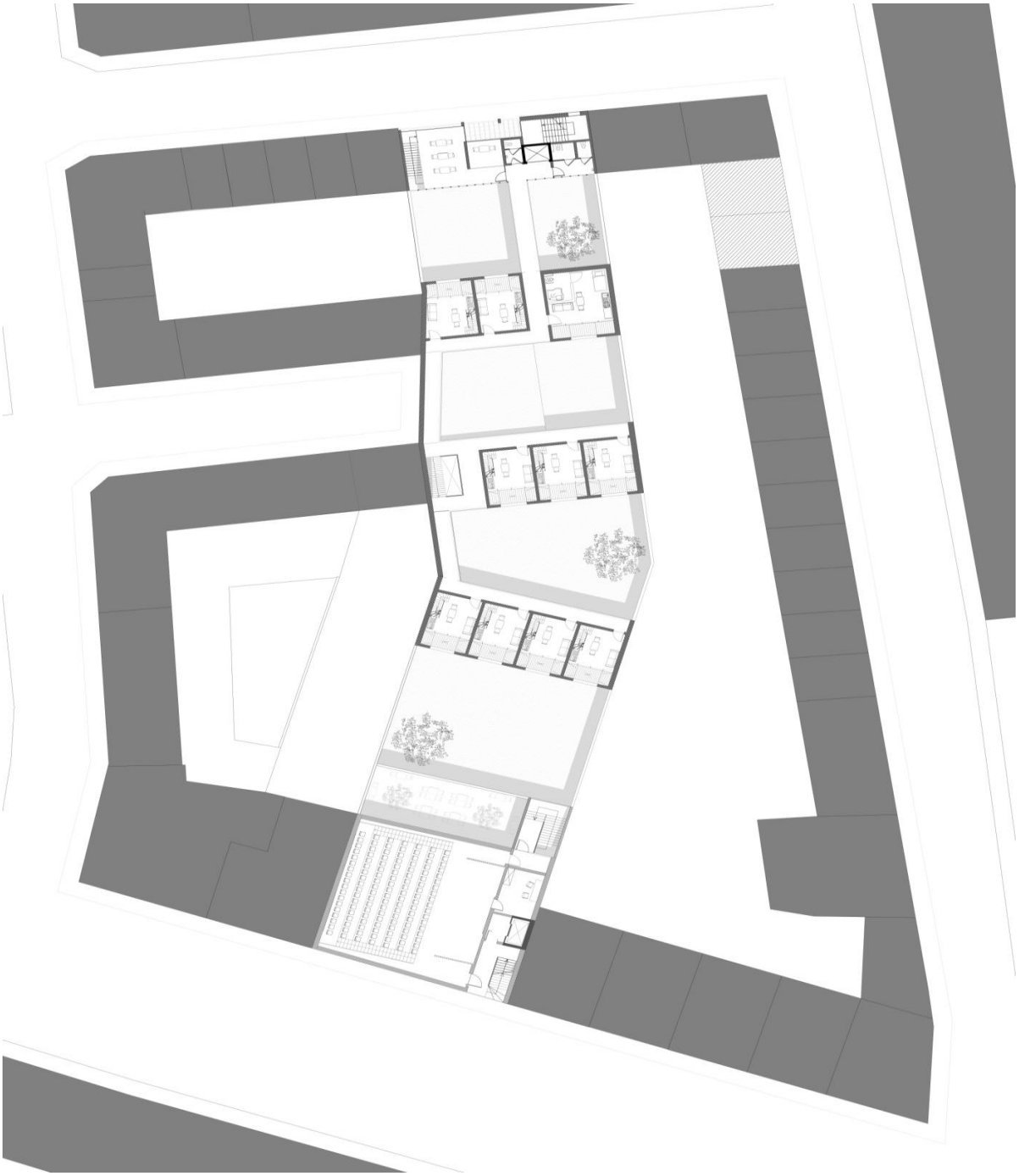
Grondplannen:

- *Kelder*

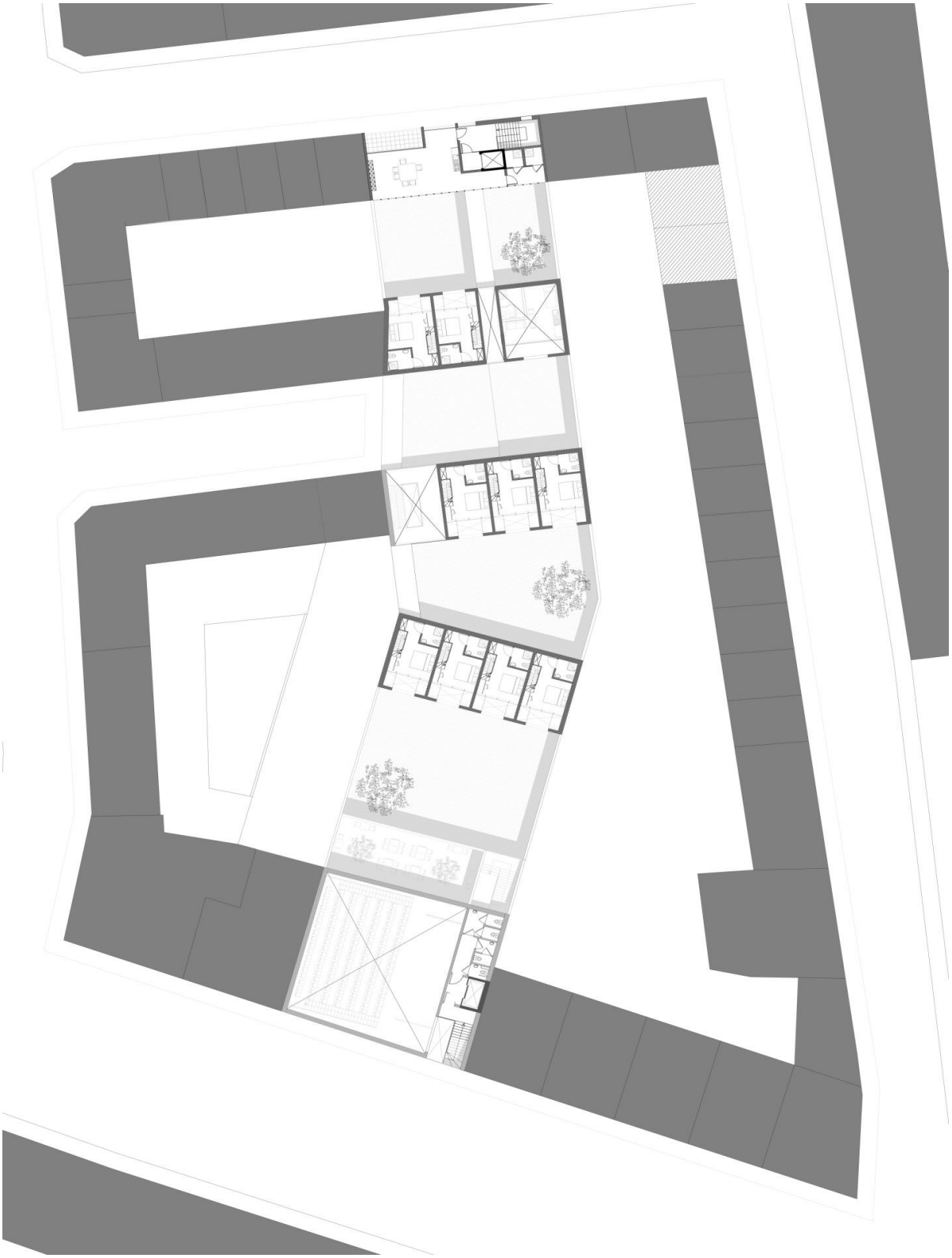


- *Gelijkvloers*

- *Eerste verdieping*

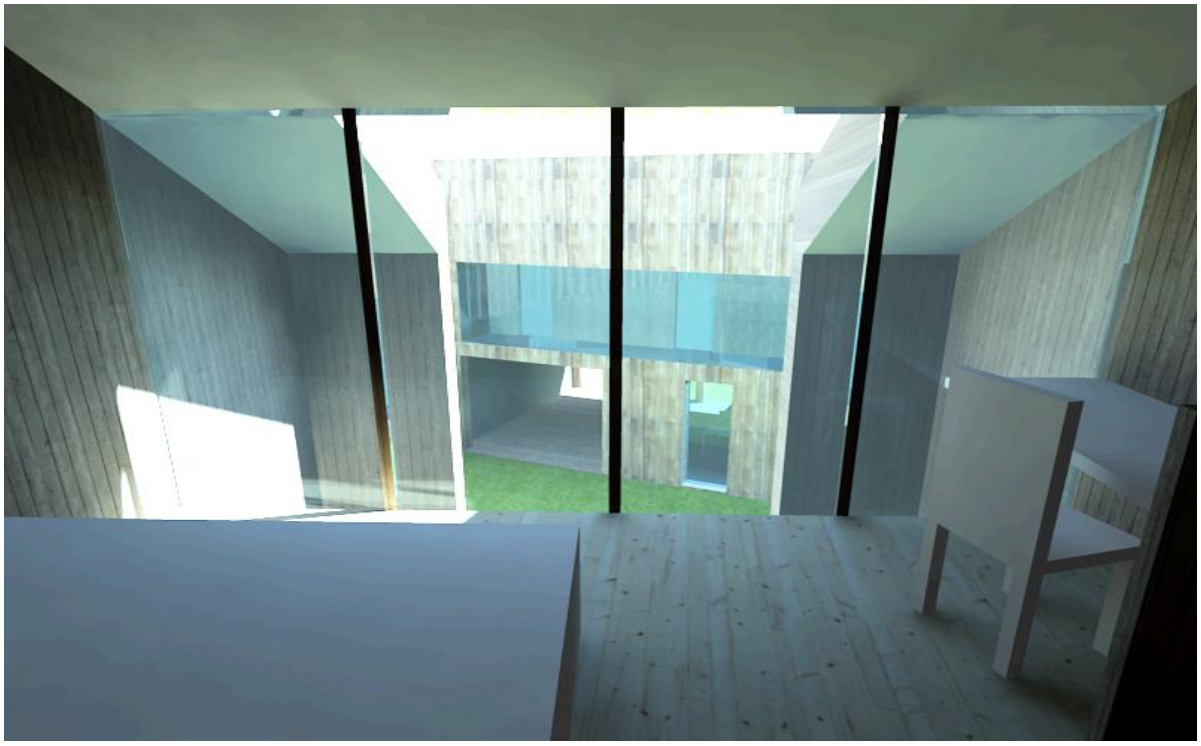


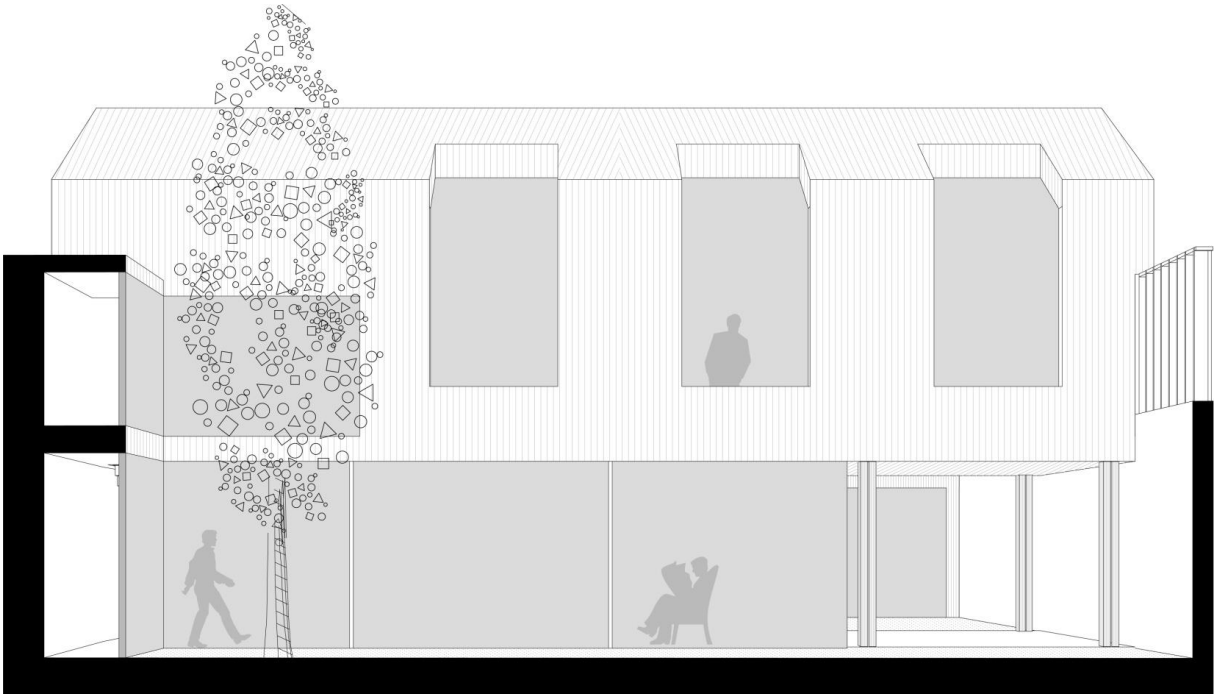
- *Tweede verdieping*



Gevels:

Beelden:





BIJLAGE B: C#-code - Applicatie adjacencymatrix (H2-3.2.1)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("DETENTIEHUIS"); Console.WriteLine();

            XmlDocument docu = new XmlDocument();
            docu.Load("C:\\Users\\Samuel\\Desktop\\C# dingen\\detentiehuis revit model.xml");
            XmlNodeList spaces = docu.GetElementsByTagName("Space");
            Console.WriteLine(spaces.Count + " verschillende ruimtes");
            XmlNodeList surface = docu.GetElementsByTagName("Surface");
            Console.WriteLine(surface.Count + " verschillende oppervlaktes");

            int[,] matrix = new int[spaces.Count, spaces.Count];
            int[,] matrixdoor = new int[spaces.Count, spaces.Count];

            string temp0 = "";
            int val0 = 0;
            string temp1 = "";
            string temp2 = "";
            string temp3 = "";
            string tempx = "";

            List<int> list0 = new List<int>();
            List<string> list = new List<string>();
            List<string> list1 = new List<string>();
            List<string> list2 = new List<string>();

//shadesurfaces er uit halen

            XmlReader reader0 = XmlReader.Create("C:\\Users\\Samuel\\Desktop\\C# dingen\\detentiehuis
            revit model.xml");
            while (reader0.Read())
            {
                if (reader0.NodeType == XmlNodeType.Element && reader0.Name == "Surface")
                {
                    if (reader0.HasAttributes && reader0.GetAttribute("surfaceType") == "Shade")
                    {
                        temp0 = (reader0.GetAttribute("id"));
                        int lengte = reader0.GetAttribute("id").Length;

                        string a = temp0;
                        string b = string.Empty;
                        for (int i = 0; i < lengte; i++)
                        {
                            if (Char.IsDigit(a[i]))
                            {
                                b += a[i];
                            }
                        }
                        val0 = int.Parse(b);
                        list0.Add(val0);
                    }
                }
            }
            Console.WriteLine("eerste shade-surface: su-" + list0[0]);

            Console.ReadLine();

// rakende ruimtes vinden
```

```
XmlReader reader = XmlReader.Create("C:\\Users\\Samuel\\Desktop\\C# dingen\\detentiehuis
revit model.xml");

while (reader.Read())
{
    for (int i = 1; i < list0[0]; i++)
    {
        if (reader.NodeType == XmlNodeType.Element && reader.Name == "Surface")
        {

            if (reader.HasAttributes && reader.GetAttribute("id") == "su-" + i)
            {
                temp1 = (reader.GetAttribute("id"));

                //Console.WriteLine(temp1);
                list.Add(temp1);
            }
        }
    }

    if (reader.NodeType == XmlNodeType.Element && reader.Name == "AdjacentSpaceId")
    {
        if (reader.HasAttributes)
        {
            temp2 = (reader.GetAttribute("spaceIdRef"));

            string a = temp2;
            string b = string.Empty;
            for (int i = 0; i < 6; i++)
            {
                if (Char.IsDigit(a[i]))
                {
                    b += a[i];
                }
            }
            int val2 = int.Parse(b);

            //Console.WriteLine(val2 + " (" + temp2 + ")");
            list.Add(temp2);
        }
    }
}

for (int i = 0; i < list.Count; i++)
{
    for (int j = 1; j < surface.Count + 1; j++)
    {
        if (list[i] == "su-" + j)
        {
            if (list[i + 1] != "su-" + (j + 1) && list[i + 2] != "su-" + (j + 1))
            {
                if (list[i + 1] != list[i + 2])
                {
                    Console.WriteLine("rakend: " + list[i + 1] + " en " + list[i + 2]);

                    string a = list[i + 1];
                    string b = string.Empty;
                    for (int k = 0; k < 6; k++)
                    {
                        if (Char.IsDigit(a[k]))
                        {
                            b += a[k];
                        }
                    }
                    int value1 = int.Parse(b);

                    string c = list[i + 2];
                    string d = string.Empty;
                    for (int k = 0; k < 6; k++)
                    {
                        if (Char.IsDigit(c[k]))
                        {
                            d += c[k];
                        }
```

```
                            }
                            int value2 = int.Parse(d);

                            matrix[value1 - 1, value2 - 1] = 1;
                            matrix[value2 - 1, value1 - 1] = 1;

                        }
                    }
                }
            }
        }


        Console.ReadLine();

// Spaces list1 en matrix schrijven

        XmlReader reader1 = XmlReader.Create("C:\\Users\\Samuel\\Desktop\\C# dingen\\detentiehuis
revit model.xml");

        while (reader1.Read())
        {
            if (reader1.NodeType == XmlNodeType.Element && reader1.Name == "Space")
            {
                if (reader1.HasAttributes)
                {
                    tempx = (reader1.GetAttribute("id"));
                    list1.Add(tempx);
                }
            }
        }

        for (int i = 0; i < spaces.Count; i++)
        {
            if (list1[i].Length <= 15)
            {
                Console.Write(list1[i] + "\t"+ "\t");
            }
            else
                Console.Write(list1[i] + "\t");

            for (int j = 0; j < spaces.Count; j++)
            {
                Console.Write(matrix[i, j] + " ");
            }
            Console.WriteLine();
        }

        Console.ReadLine();
```

BIJLAGE C: C#-code - De permutatie (H3-2.2.2)

```
  private void RunScript(List<System.Object> List, ref object Total, ref object B)
{
  string tem = null;

  for (int i = 0; i < List.Count ; i++)
  {
    tem = tem + i.ToString();
  }
  string inputLine = inputLine = tem;

  Recursion rec = new Recursion();
  rec.InputSet = rec.MakeCharArray(inputLine);
  rec.CalcPermutation(0);

  Total = rec.PermutationCount;
  B = rec.tempo;
}

class Recursion
{
  private int elementLevel = -1;
  private int numberOfElements;
  private int[] permutationValue = new int[0];
  public ArrayList tempo = new ArrayList();

  private char[] inputSet;
  public char[] InputSet
  {
    get { return inputSet; }
    set { inputSet = value; }
  }
  private int permutationCount = 0;
  public int PermutationCount
  {
    get { return permutationCount; }
    set { permutationCount = value; }
  }

  public char[] MakeCharArray(string InputString)
  {
    char[] charString = InputString.ToCharArray();
    Array.Resize(ref permutationValue, charString.Length);
    numberOfElements = charString.Length;
    return charString;
  }

  public void CalcPermutation(int k)
  {
    elementLevel++;
    permutationValue.SetValue(elementLevel, k);

    if (elementLevel == numberOfElements)
    {
      OutputPermutation(permutationValue);
    }
    else
    {
      for (int i = 0; i < numberOfElements; i++)
      {
        if (permutationValue[i] == 0)
        {
          CalcPermutation(i);
        }
      }
    }
    elementLevel--;
    permutationValue.SetValue(0, k);
  }
```

```
    private void OutputPermutation(int[] value)
    {
      foreach (int i in value)
      {
        tempo.Add(inputSet.GetValue(i - 1));
      }
      PermutationCount++;
    }
  }
}
```

BIJLAGE D: C#-code - Splitsen van volumes (H3-2.3.2)

```
private void RunScript(List<Brep> GEO, Brep PLANE, double index, ref object Split)
  {
    List<Rhino.Geometry.Brep> temp0 = new List<Rhino.Geometry.Brep>();
    List<Rhino.Geometry.Brep> temp1 = new List<Rhino.Geometry.Brep>();
    List<int> temp2 = new List<int>();

    for (int i = 0; i < GEO.Count ; i++)
    {
      Curve[] crvs = null;
      Point3d[] pts = null;
      bool rc = Rhino.Geometry.Intersect.Intersection.BrepBrep(GEO[i], PLANE, 0.01, out crvs, out pts);

      // aantal niet doorgesneden volumes (crvs = 0) || doorsnijden door het zijvlak (crvs = 5)
      if(crvs.Length != 4)
      {
        temp2.Add(i);
      }

      // alle mogelijkheden voor het aantal doorgesneden volumes in 1 lijst (crvs = 4)
      if(crvs.Length == 4)
      {
        Rhino.Geometry.Brep[] resultBrep = GEO[i].Split(PLANE, 0.01);
        for(int j = 0; j < GEO.Count ; j++)
        {

          // niet doorgesneden volumes erbij zetten
          if(j != i)
          {
            temp0.Add(GEO[j]);
          }
        }

        temp0.Add(resultBrep[0].CapPlanarHoles(0.01));
        temp0.Add(resultBrep[1].CapPlanarHoles(0.01));
      }
    }

    // aantal mogelijkheden
    double b = (GEO.Count - temp2.Count);

    // intverallengte tussen 0 en 1
    double a = 1 / b;

    // de lijst splitsen in functie van indexnummer
    for (int k = 0; k < (GEO.Count - temp2.Count); k++)
    {
      if(( index >= k * a && index < (k + 1) * a) || (index == k / (GEO.Count - temp2.Count - 1) &&
index != 0) )
      {
        for (int l = 0; l < GEO.Count + 1 ;l++)
        {
          temp1.Add(temp0[(k * (GEO.Count + 1)) + l]);
        }
      }
    }

    //lege oplossingen verwijderen
    temp1.Remove(null);
    //doorsnijding door een vlak dat bij alle volumes al doorgesneden is dan de vorige oplossing (temp
= 0)
    if (temp1.Count != 0)
    {
      Split = temp1;
    }
    if (temp1.Count == 0)
    {
      Split = GEO;
    }
  }
```

BIJLAGE E: C#-code - Normalized Polish Expression (H4-3.2.2)

```
private void RunScript(int x, ref object A, ref object B, ref object C, ref object D, ref object E, ref
object F)
  {
    int zoneperm = Convert.ToInt32(Math.Pow(2, (x - 1)));
    List<int> templist = new List<int>();
    List<string> stringlist = new List<string>();
    string[,] stringarray = new string[x - 1, zoneperm];
    List<string> polishlist = new List<string>();

    // permutatie met herhaling van (x-1)keer kiezen uit 2 elementen (0 en 1) (2^(x-1))

    for (int ia = 0; ia < 2; ia++)
    {
      if(x == 2)
      {
        templist.Add(ia);
      }
      for (int ib = 0; ib < 2; ib++)
      {
        if (x == 2)
        {
          break;
        }
        if(x == 3)
        {
          templist.Add(ia);
          templist.Add(ib);
        }

        for (int ic = 0; ic < 2; ic++)
        {
          if (x == 3)
          {
            break;
          }
          if(x == 4)
          {
            templist.Add(ia);
            templist.Add(ib);
            templist.Add(ic);
          }
          for (int id = 0; id < 2; id++)
          {
            if (x == 4)
            {
              break;
            }
            if(x == 5)
            {
              templist.Add(ia);
              templist.Add(ib);
              templist.Add(ic);
              templist.Add(id);
            }
            for (int ie = 0; ie < 2; ie++)
            {
              if (x == 5)
              {
                break;
              }
              if(x == 6)
              {
                templist.Add(ia);
                templist.Add(ib);
                templist.Add(ic);
                templist.Add(id);
                templist.Add(ie);
              }
              for (int ig = 0; ig < 2; ig++)
```

```
{
  if (x == 6)
  {
    break;
  }
  if(x == 7)
  {
    templist.Add(ia);
    templist.Add(ib);
    templist.Add(ic);
    templist.Add(id);
    templist.Add(ie);
    templist.Add(ig);
  }
  for (int ih = 0; ih < 2; ih++)
  {
    if (x == 7)
    {
      break;
    }
    if(x == 8)
    {
      templist.Add(ia);
      templist.Add(ib);
      templist.Add(ic);
      templist.Add(id);
      templist.Add(ie);
      templist.Add(ig);
      templist.Add(ih);
    }
    for (int ii = 0; ii < 2; ii++)
    {
      if (x == 8)
      {
        break;
      }
      if(x == 9)
      {
        templist.Add(ia);
        templist.Add(ib);
        templist.Add(ic);
        templist.Add(id);
        templist.Add(ie);
        templist.Add(ig);
        templist.Add(ih);
        templist.Add(ii);
      }
      for (int ij = 0; ij < 2; ij++)
      {
        if (x == 9)
        {
          break;
        }
        if(x == 10)
        {
          templist.Add(ia);
          templist.Add(ib);
          templist.Add(ic);
          templist.Add(id);
          templist.Add(ie);
          templist.Add(ig);
          templist.Add(ih);
          templist.Add(ii);
          templist.Add(ij);
        }
        for (int ik = 0; ik < 2; ik++)
        {
          if (x == 10)
          {
            break;
          }
          if(x == 11)
          {
            templist.Add(ia);
```

```
  templist.Add(ib);
  templist.Add(ic);
  templist.Add(id);
  templist.Add(ie);
  templist.Add(ig);
  templist.Add(ih);
  templist.Add(ii);
  templist.Add(ij);
  templist.Add(ik);
}
for (int il = 0; il < 2; il++)
{
  if (x == 11)
  {
    break;
  }
  if(x == 12)
  {
    templist.Add(ia);
    templist.Add(ib);
    templist.Add(ic);
    templist.Add(id);
    templist.Add(ie);
    templist.Add(ig);
    templist.Add(ih);
    templist.Add(ii);
    templist.Add(ij);
    templist.Add(ik);
    templist.Add(il);
  }
  for (int im = 0; im < 2; im++)
  {
    if (x == 12)
    {
      break;
    }
    if(x == 13)
    {
      templist.Add(ia);
      templist.Add(ib);
      templist.Add(ic);
      templist.Add(id);
      templist.Add(ie);
      templist.Add(ig);
      templist.Add(ih);
      templist.Add(ii);
      templist.Add(ij);
      templist.Add(ik);
      templist.Add(il);
      templist.Add(im);
    }
    for (int io = 0; io < 2; io++)
    {
      if (x == 13)
      {
        break;
      }
      if(x == 14)
      {
        templist.Add(ia);
        templist.Add(ib);
        templist.Add(ic);
        templist.Add(id);
        templist.Add(ie);
        templist.Add(ig);
        templist.Add(ih);
        templist.Add(ii);
        templist.Add(ij);
        templist.Add(ik);
        templist.Add(il);
        templist.Add(im);
        templist.Add(io);
      }
      for (int ip = 0; ip < 2; ip++)
```

```
                        {
                          if (x == 14)
                          {
                            break;
                          }
                          if(x == 15)
                          {
                            templist.Add(ia);
                            templist.Add(ib);
                            templist.Add(ic);
                            templist.Add(id);
                            templist.Add(ie);
                            templist.Add(ig);
                            templist.Add(ih);
                            templist.Add(ii);
                            templist.Add(ij);
                            templist.Add(ik);
                            templist.Add(il);
                            templist.Add(im);
                            templist.Add(io);
                            templist.Add(ip);
                          }
                          for (int iq = 0; iq < 2; iq++)
                          {
                            {
                              if (x == 15)
                              {
                                break;
                              }
                              if(x == 16)
                              {
                                templist.Add(ia);
                                templist.Add(ib);
                                templist.Add(ic);
                                templist.Add(id);
                                templist.Add(ie);
                                templist.Add(ig);
                                templist.Add(ih);
                                templist.Add(ii);
                                templist.Add(ij);
                                templist.Add(ik);
                                templist.Add(il);
                                templist.Add(im);
                                templist.Add(io);
                                templist.Add(ip);
                                templist.Add(iq);
                              }

                            }}}}}}}}}}}}}}}}

// 0 en 1 vervangen door V en H

for (int i = 0; i < templist.Count; i++)
{
  if (templist[i] == 0)
  {
    stringlist.Add("V");
  }
  if (templist[i] == 1)
  {
    stringlist.Add("H");
  }
}

// alles ordenen in een matrix - per kolom een reeks V en H's

int counterint = 0;

for (int i = 0;i < zoneperm;i++)
{
  for(int j = 0;j < x - 1;j++)
  {
    stringarray.SetValue(stringlist[j + counterint], j, i);
  }
```

```
      counterint = counterint + (x - 1);
    }

    B = stringlist;
    A = stringarray;

    // de numberarray

    // eerst gepermuteerde lijst maken van alle mogelijke combinaties van x aantal cijfers
    List<string> inputlist = new List<String>();

    string tem = null;

    for (int i = 0; i < x ; i++)
    {
      tem = tem + i.ToString();
    }

    string inputLine = inputLine = tem;

    Recursion rec = new Recursion();
    rec.InputSet = rec.MakeCharArray(inputLine);
    rec.CalcPermutation(0);

    for (int re = 0; re < rec.tempo.Count;re++)
    {
      inputlist.Add(rec.tempo[re].ToString());
    }

    // vervolgens deze lijst opslaan in een matrix

    int[,] intarray = new int[x, (inputlist.Count) / x];

    for (int co = 0; co < (inputlist.Count) / x ; co++)
    {
      for (int ro = 0; ro < x ; ro++)
      {
        intarray.SetValue(Convert.ToInt32(inputlist[ro + x * co]), ro, co); // het is een string en
moet getal worden
      }
    }

    C = inputlist;

    // variabelen

    for (int y = 0; y < intarray.GetLength(1);y++)
    {

      for (int s = 0; s < zoneperm; s++)  // voor elke kolom in de slices matrix
      {

        int numbercount = x;
        int slicescount = x - 1;
        int polishcount = 2 * x - 1;
        int[,] sliceact = new int[1, 1]{{1}};
        int[,] numberact = new int[1, 1]{{2}};

        string[,] polish = new string[polishcount, 1];

        // volgende stukken met 3 regels
        // voor de tussenliggende tekens

        for (int i = 3; i < (numbercount * 2) - 2; i++)  // tussenliggende tekens
        {
          int polishlength = polish.GetLength(1);

          for (int j = 0; j < polishlength;j++)  // voor elke kolom in de polish oplossing
          {
            //regel 1

            if(sliceact[0, j] == numberact[0, j] - 1)
            {
              polish.SetValue(intarray[numberact[0, j], y].ToString(), i, j);
```

```
                numberact[0, j]++;
              }

                //regel 2

              else
              {
                if (numberact[0, j] < numbercount - 1 && sliceact[0, j] < slicescount - 1)
                {
                  // tijdelijke matrices om de numberact-, sliceact- en polish matrix vergroten
                  int[,] tempnumberact = new int[1, numberact.GetLength(1) + 1];
                  int[,] tempsliceact = new int[1, sliceact.GetLength(1) + 1];
                  string[,] temppolish = new string[polish.GetLength(0), polish.GetLength(1) + 1];

                  //koppieer de waarden van de referentiematrixen naar de tijdelijke
                  for (int g = 0; g < polish.GetLength(0);g++)
                  {
                    for (int h = 0; h < polish.GetLength(1);h++)
                    {
                      temppolish.SetValue(polish[g, h], g, h);
                    }
                    temppolish.SetValue(polish[g, j], g, polish.GetLength(1));
                  }
                  for (int h = 0; h < polish.GetLength(1);h++)
                  {
                    tempsliceact.SetValue(sliceact[0, h], 0, h);
                    tempnumberact.SetValue(numberact[0, h], 0, h);
                  }

                  tempsliceact.SetValue(sliceact[0, j], 0, polish.GetLength(1));
                  tempnumberact.SetValue(numberact[0, j], 0, polish.GetLength(1));

                  //sla de tijdelijken op als de referentiematrixen
                  polish = temppolish;
                  sliceact = tempsliceact;
                  numberact = tempnumberact;

                  // steeds 2 oplossingen

                  polish.SetValue(intarray[numberact[0, j], y].ToString(), i, j);
                  numberact[0, j]++;

                  polish.SetValue(stringarray[sliceact[0,    polish.GetLength(1)    -    1],    s],    i,
polish.GetLength(1) - 1);
                  sliceact[0, polish.GetLength(1) - 1]++;

                }
                else
                {

                  // regel 3

                  if (sliceact[0, j] == slicescount - 1)
                  {
                    polish.SetValue(intarray[numberact[0, j], y].ToString(), i, j);
                    numberact[0, j]++;
                  }
                  if (numberact[0, j] == numbercount - 1)
                  {
                    polish.SetValue(stringarray[sliceact[0, j], s], i, j);
                    sliceact[0, j]++;
                  }
                }
              }
            }
          }

          // eerste 2 zijn een getal, 3e is een operator en laatste is ook een operator

          for (int t = 0; t < polish.GetLength(1) ;t++)
          {
            // eerste 2 zijn een number

            polish.SetValue(intarray[0, y].ToString(), 0, t);
```

```
          polish.SetValue(intarray[1, y].ToString(), 1, t);

          // de 3e waarde is een slice

          polish.SetValue(stringarray[0, s], 2, t);

          // laatste teken is een slice

          polish.SetValue(stringarray[slicescount - 1, s], polishcount - 1, t);
        }

        // omzetten naar 1 string en in lijst opslagen

        int polishlength2 = polish.GetLength(1);

        for (int q = 0; q < polishlength2;q++)
        {
          string defstring = "";

          for (int i = 0; i < polishcount;i++)
          {
            defstring = defstring.Insert(0, polish[polishcount - 1 - i, q].ToString());
          }
          // als er VV of HH voorkomt is dit ongeldig
          int tester = 0;

          for (int i = 0; i < polishcount - 1 ; i++)
          {
            if (polish[i, q].ToString() == polish[i + 1, q].ToString())
            {
              tester++;
            }
          }
          if (tester == 0)
          {
            polishlist.Add(defstring);
          }
        }
        D = polish;
      }
    }
    E = polishlist;
  }
```

BIJLAGE F: C#-code - Topologische Matrix (H4-3.3.2)

```
// uitdrukking omkeren

private void RunScript(string x, ref object A)
  {
    double getal = x.Length;
    string lijn = "";
    List<string> invertlist = new List<string>();

    for (int i = 0; i < getal; i++)
    {
      invertlist.Add(x[i].ToString());
    }
    for (int i = 0; i < invertlist.Count;i++)
    {
      lijn = lijn.Insert(0, invertlist[i]);
    }
    A = lijn;
  }

private void RunScript(string x, int y, ref object A, ref object B, ref object C, ref object D, ref
object lijst, ref object boom, ref object ruimtes, ref object Adjlijst, ref object Adj, ref object
tester, ref object E)
  {

// string naar lijst omzetten

    List<char> temp = new List<char>();

    foreach (char c in x)
    {
      temp.Add(c);
    }

    A = temp;

    // lijst van integers

    List<int> temp1 = new List<int>();

    for (int i = 0; i < temp.Count; i++)
    {
      if (Char.IsDigit(temp[i]))
      {
        temp1.Add(int.Parse(temp[i].ToString()));
      }
      else
      {
        if (temp[i].ToString() == "V")
        {
          temp1.Add(-1);
        }
        else
        {
          temp1.Add(-2);
        }
      }
    }

    B = temp1;

    // lijst van matrices

    List<int[,]> matrixlijst = new List<int[,]>();

    for (int i = 0; i < temp1.Count; i++)
    {
      int[,] a = new int[1, 1]{{temp1[i]}};
      matrixlijst.Add(a);
```

```
    }

    C = matrixlijst;

    // the loop

    int splitsen = 1;
    while(splitsen > 0)
    {

      for (int i = 0; i < (matrixlijst.Count) - 2;i++)
      {
        // verticale split
        if ( matrixlijst[i][0, 0] == -1 && matrixlijst[i + 1][0, 0] >= 0 && matrixlijst[i + 2][0, 0] >=
0 )
        {
          int rij1 = matrixlijst[i + 1].GetLength(0);
          int kolom1 = matrixlijst[i + 1].GetLength(1);
          int rij2 = matrixlijst[i + 2].GetLength(0);
          int kolom2 = matrixlijst[i + 2].GetLength(1);

          int rij3 = 0;
          int kolom3 = 0;

          // gelijk aantal rijen

          if (rij1 == rij2)
          {
            rij3 = rij2;
            kolom3 = kolom1 + kolom2;
            int[,] change = new int[rij3, kolom3];

            for( int r = 0; r < rij1;r++)
            {
              for (int k = 0; k < kolom1;k++)
              {
                change.SetValue(matrixlijst[i + 1][r, k], r, k);
              }
            }
            for( int r = 0; r < rij2;r++)
            {
              for (int k = 0; k < kolom2;k++)
              {
                change.SetValue(matrixlijst[i + 2][r, k], r, k + kolom1);
              }
            }

            //matrixlijst aanpassen met de nieuwe matrix

            List<int[,]> matrixtemp = new List<int[,]>();

            if (i != 0)
            {
              for (int m = 0; m < i;m++)
              {
                matrixtemp.Add(matrixlijst[m]);
              }
              matrixtemp.Add(change);

              if (i + 3 < matrixlijst.Count)
              {
                for (int n = i + 3; n < matrixlijst.Count;n++)
                {
                  matrixtemp.Add(matrixlijst[n]);
                }
              }
              matrixlijst = matrixtemp;
            }
            else
            {
              matrixtemp.Add(change);
              matrixlijst = matrixtemp;
            }
          }
```

```
// verschillend aantal rijen

if (rij1 != rij2)
{

  // vind het kleinste getal dat deelbaar is door beide rij-aantallen

  for (int p = 1; p < 999;p++)
  {
    rij3 = p;
    if(rij3 % rij1 == 0 && rij3 % rij2 == 0) break;
  }

  // kolommen optellen

  kolom3 = kolom1 + kolom2;
  int[,] change = new int[rij3, kolom3];

  //matrix 1 verschalen

  int[,] mx1 = new int[rij3, kolom1];
  int scale1 = rij3 / rij1;

  for( int r = 0; r < rij1;r++)
  {
    for (int k = 0; k < kolom1;k++)
    {
      for (int q = 0; q < scale1;q++)
      {
        mx1.SetValue(matrixlijst[i + 1][r, k], q + (scale1 * r), k);
      }
    }
  }

  //matrix 2 verschalen

  int[,] mx2 = new int[rij3, kolom2];
  int scale2 = rij3 / rij2;

  for( int r = 0; r < rij2;r++)
  {
    for (int k = 0; k < kolom2;k++)
    {
      for (int q = 0; q < scale2;q++)
      {
        mx2.SetValue(matrixlijst[i + 2][r, k], q + (scale2 * r), k);
      }
    }
  }

  //matrix change = matrix 1 + matrix 2

  for( int r = 0; r < mx1.GetLength(0);r++)
  {
    for (int k = 0; k < mx1.GetLength(1);k++)
    {
      change.SetValue(mx1[r, k], r, k);
    }
  }
  for( int r = 0; r < mx2.GetLength(0);r++)
  {
    for (int k = 0; k < mx2.GetLength(1);k++)
    {
      change.SetValue(mx2[r, k], r, k + kolom1);
    }
  }

  //matrixlijst aanpassen met de nieuwe matrix

  List<int[,]> matrixtemp = new List<int[,]>();

  if (i != 0)
  {
```

```
            for (int m = 0; m < i;m++)
            {
              matrixtemp.Add(matrixlijst[m]);
            }
            matrixtemp.Add(change);

            if (i + 3 < matrixlijst.Count)
            {
              for (int n = i + 3; n < matrixlijst.Count;n++)
              {
                matrixtemp.Add(matrixlijst[n]);
              }
            }
            matrixlijst = matrixtemp;
          }
          else
          {
            matrixtemp.Add(change);
            matrixlijst = matrixtemp;
          }
        }
      }

      // horizontale split
      if ( matrixlijst[i][0, 0] == -2 && matrixlijst[i + 1][0, 0] >= 0 && matrixlijst[i + 2][0, 0] >=
0 )
      {
        int rij1 = matrixlijst[i + 1].GetLength(0);
        int kolom1 = matrixlijst[i + 1].GetLength(1);
        int rij2 = matrixlijst[i + 2].GetLength(0);
        int kolom2 = matrixlijst[i + 2].GetLength(1);

        int rij3 = 0;
        int kolom3 = 0;

        // gelijk aantal kolommen

        if (kolom1 == kolom2)
        {
          kolom3 = kolom2;
          rij3 = rij1 + rij2;
          int[,] change = new int[rij3, kolom3];

          for( int r = 0; r < rij2;r++)
          {
            for (int k = 0; k < kolom2;k++)
            {
              change.SetValue(matrixlijst[i + 2][r, k], r, k);
            }
          }
          for( int r = 0; r < rij1;r++)
          {
            for (int k = 0; k < kolom1;k++)
            {
              change.SetValue(matrixlijst[i + 1][r, k], r + rij2, k);
            }
          }

          //matrixlijst aanpassen met de nieuwe matrix

          List<int[,]> matrixtemp = new List<int[,]>();

          if (i != 0)
          {
            for (int m = 0; m < i;m++)
            {
              matrixtemp.Add(matrixlijst[m]);
            }

            matrixtemp.Add(change);

            if (i + 3 < matrixlijst.Count)
            {
              for (int n = i + 3; n < matrixlijst.Count;n++)
```

```
      {
        matrixtemp.Add(matrixlijst[n]);
      }
    }
    matrixlijst = matrixtemp;
  }
  else
  {
    matrixtemp.Add(change);
    matrixlijst = matrixtemp;
  }
}

// verschillend aantal kolommen

if (kolom1 != kolom2)
{

  // vind het kleinste getal dat deelbaar is door beide kolom-aantallen

  for (int p = 1; p < 999;p++)
  {
    kolom3 = p;
    if(kolom3 % kolom1 == 0 && kolom3 % kolom2 == 0) break;
  }

  // rijen optellen

  rij3 = rij1 + rij2;
  int[,] change = new int[rij3, kolom3];

  //matrix 1 verschalen

  int[,] mx1 = new int[rij1, kolom3];
  int scale1 = kolom3 / kolom1;

  for( int r = 0; r < rij1;r++)
  {
    for (int k = 0; k < kolom1;k++)
    {
      for (int q = 0; q < scale1;q++)
      {
        mx1.SetValue(matrixlijst[i + 1][r, k], r, q + (scale1 * k));
      }
    }
  }

  //matrix 2 verschalen

  int[,] mx2 = new int[rij2, kolom3];
  int scale2 = kolom3 / kolom2;

  for( int r = 0; r < rij2;r++)
  {
    for (int k = 0; k < kolom2;k++)
    {
      for (int q = 0; q < scale2;q++)
      {
        mx2.SetValue(matrixlijst[i + 2][r, k], r, q + (scale2 * k));
      }
    }
  }

  //matrix change = matrix 1 + matrix 2
  for( int r = 0; r < mx2.GetLength(0);r++)
  {
    for (int k = 0; k < mx2.GetLength(1);k++)
    {
      change.SetValue(mx2[r, k], r, k);
    }
  }
  for( int r = 0; r < mx1.GetLength(0);r++)
  {
    for (int k = 0; k < mx1.GetLength(1);k++)
```

```
            {
              change.SetValue(mx1[r, k], r + rij2, k);
            }
          }

          //matrixlijst aanpassen met de nieuwe matrix

          List<int[,]> matrixtemp = new List<int[,]>();
          if (i != 0)
          {
            for (int m = 0; m < i;m++)
            {
              matrixtemp.Add(matrixlijst[m]);
            }
            matrixtemp.Add(change);

            if (i + 3 < matrixlijst.Count)
            {
              for (int n = i + 3; n < matrixlijst.Count;n++)
              {
                matrixtemp.Add(matrixlijst[n]);
              }
            }
            matrixlijst = matrixtemp;
          }
          else
          {
            matrixtemp.Add(change);
            matrixlijst = matrixtemp;
          }
        }
      }
    }

    // stop the while loop

    List<int> t = new List<int>();

    for (int w = 0; w < matrixlijst.Count;w++)
    {
      if (matrixlijst[w][0, 0] < 0)
      {
        t.Add(w);
      }
    }
    splitsen = t.Count;
  }
  // print de topologische matrixlijst

  D = matrixlijst[0];

}
```

BIJLAGE G: C#-code - Nabijheidmatrix (H4-3.4.2)

```
private void RunScript(string x, int y, ref object A, ref object B, ref object C, ref object D, ref
object lijst, ref object boom, ref object ruimtes, ref object Adjlijst, ref object Adj, ref object
tester, ref object E)
  {
    // ------** ADJACENCY MATRIX **------

    // topo matrix in lijst versie

    List<int> tempi = new List<int>();

    for(int i = 0; i < matrixlijst[0].GetLength(0);i++)

    {
      for(int j = 0; j < matrixlijst[0].GetLength(1);j++)
      {
        tempi.Add(matrixlijst[0][i, j]);
      }
    }
    lijst = tempi;

    // topo matrix in boom versie

    DataTree<System.Object> re = new DataTree<System.Object>();

    for(int i = 0; i < matrixlijst[0].GetLength(0);i++)

    {
      for(int j = 0; j < matrixlijst[0].GetLength(1);j++)
      {
        re.Add(matrixlijst[0][i, j], new GH_Path(i));
      }
    }
    boom = re;

    // number of spaces

    int getal = 0;

    for (int q = 0; q < tempi.Count; q++)
    {
      if (tempi[q] > getal)
      {
        getal = tempi[q];
      }
    }
    ruimtes = getal + 1;

    // adjacency matrix

    int[,] d = new int[(getal + 1), (getal + 1)];

    for(int i = 0; i < matrixlijst[0].GetLength(0);i++)
    {
      for(int j = 0; j < matrixlijst[0].GetLength(1);j++)
      {
        if((i + 1) < matrixlijst[0].GetLength(0))
        {
          d.SetValue(1, matrixlijst[0][i, j], matrixlijst[0][i + 1, j]);
          d.SetValue(1, matrixlijst[0][i + 1, j], matrixlijst[0][i, j]);
        }
        if((i - 1) >= 0)
        {
          d.SetValue(1, matrixlijst[0][i, j], matrixlijst[0][i - 1, j]);
          d.SetValue(1, matrixlijst[0][i - 1, j], matrixlijst[0][i, j]);
        }
        if((j + 1) < matrixlijst[0].GetLength(1))
        {
          d.SetValue(1, matrixlijst[0][i, j], matrixlijst[0][i, j + 1]);
```

```
      d.SetValue(1, matrixlijst[0][i, j + 1], matrixlijst[0][i, j]);
    }
    if((j - 1) >= 0)
    {
      d.SetValue(1, matrixlijst[0][i, j], matrixlijst[0][i, j - 1]);
      d.SetValue(1, matrixlijst[0][i, j - 1], matrixlijst[0][i, j]);
    }
  }
}
for(int i = 0; i < d.GetLength(0);i++)
{
  for(int j = 0; j < d.GetLength(1);j++)
  {
    if(i == j)
    {

      // hoeveel eenheden is de ruimte groot
      int count = new int();
      List<int> temp2 = new List<int>();

      for(int p = 0; p < tempi.Count;p++)
      {
        if (tempi[p] == i)
        {
          temp2.Add(1);
        }
      }
      count = temp2.Count;


      d.SetValue(count, i, j);
    }
  }
}
Adjlijst = d;

// adjacencymatrix in boom versie

DataTree<System.Object> re1 = new DataTree<System.Object>();

for(int i = 0; i < d.GetLength(0);i++)

{
  for(int j = 0; j < d.GetLength(1);j++)
  {
    re1.Add(d[i, j], new GH_Path(i));
  }
}

Adj = re1;

// test of het voldoet aan een inputmatrix

// inputmatrix

int[,] inputmatrix = new int[y, y];

for (int i = 0; i < inputmatrix.GetLength(0);i++)
{
  for (int j = 0; j < inputmatrix.GetLength(1);j++)
  {
    inputmatrix.SetValue(-1, i, j);
  }
}
inputmatrix.SetValue(1, 0, 0); // ruimte 1 en 2 hebben een relatie
inputmatrix.SetValue(0, 0, 2); // ruimte 1 en 2 hebben een relatie

E = inputmatrix;

// test

int testerwaarde = 0;
int teller = 0;
bool pingtest = false;
```

```
  for (int i = 0; i < d.GetLength(0);i++)
  {
    for (int j = 0; j < d.GetLength(1);j++)
    {
      if (inputmatrix[i, j] == -1)
      {
        teller++;
      }
      if (inputmatrix[i, j] != -1 && d[i, j] == inputmatrix[i, j])
      {
        testerwaarde++;
      }
    }
  }
  if (testerwaarde == (d.GetLength(0) * d.GetLength(1)) - teller)
  {
    pingtest = true;
  }
  tester = pingtest;
}
```

BIJLAGE H: Java-code - Applicatie (H5-1.3)

```
import de.bezier.math.combinatorics.*;

//CONTROLP5
import controlP5.*;

ControlP5 cp5;
PFont font = createFont("arial", 20);

PGraphics p1;
PGraphics p2;

// VARIABELEN
ArrayList<Solution> solutions = new ArrayList<Solution>();
int[][] CONSTRAINTS;

int ZONECOUNT;

//APPLICATIE
void setup()
{
  //Create viewport
  size(1024, 768);
  p1 = createGraphics(256, 768);
  p2 = createGraphics(768, 768);

  //Create GUI items
  cp5 = new ControlP5(this);

  cp5.addTextfield("input")
    .setPosition(20, 170)
      .setSize(200, 40)
        .setFont(font)
          .setColor(color(255, 255, 0))
            .setAutoClear(false)
              .setFocus(true)
                ;

  // rest van de setup staat verder bij public void input (omdat dit in functie is van de inputwaarde)
}

void draw()
{
  //LEFT VIEWPORT
  p1.beginDraw();
  p1.background(128);
  p1.endDraw();

  //RIGHT VIEWPORT
  p2.beginDraw();
  p2.background(51);


  for (int i = 0; i < solutions.size(); i++)
  {
    Solution s = solutions.get(i);
    int size = s.topomatrix[0][0];
    p2.stroke(255, 102, 0);
    p2.ellipse(width/2*(i+1)/2, height/2*(i+1)/2, size, size);
  }

  //het aantal zones rechtsonderaan weergeven
  if (ZONECOUNT > 1) {
    p2.text("Het aantal zone's is" + " " + Integer.toString(ZONECOUNT), 625, 758);
  }
  else{
    p2.text("Geef een getal in groter dan 1", 578, 758);
  }

  p2.endDraw();
```

```
  image(p1, 0, 0);
  image(p2, 256, 0);
}

public void input(String theText) {

  // TEST OF DE INPUT EEN INTEGER IS

  String inttest = theText;
  Boolean b = true;
  try {
    int inttesting = Integer.parseInt(inttest);
    System.out.println(inttesting);
  }
  catch(NumberFormatException nFE) {
    b = false;
  }

  if (b == true) {

    // AANTAL ZONES
    ZONECOUNT = Integer.parseInt(theText);
    if (ZONECOUNT >1) {
      println("Het aantal zones is " + ZONECOUNT + ".");

      // EIGENLIJKE VOID SETUP (PER VERANDERING VAN AANTAL ZONES)

      GenerateSolutions(ZONECOUNT);

        //Create GUI items for constraints
        for (int z = 0; z < ZONECOUNT;z++){
          for (int w = 0; w < ZONECOUNT;w++){
            if (z<w){
         cp5 = new ControlP5(this);

         PFont font2 = createFont("arial", max(40-5*ZONECOUNT,5));

          cp5.addTextfield("[" + Integer.toString(z) + "," + Integer.toString(w) + "]")
            .setPosition(20 + ((200-20*(ZONECOUNT-1))/ZONECOUNT + 20)*w, 300 + ((200-20*(ZONECOUNT-
1))/ZONECOUNT + 20)*z)
              .setSize((200-20*(ZONECOUNT-1))/ZONECOUNT,(200-20*(ZONECOUNT-1))/ZONECOUNT)
                .setFont(font2)
                  .setColor(color(255, 255, 0))
                    .setAutoClear(false)
                      .setFocus(false)
                ;
            }
          }
        }



    }
  }
  if (b == false || ZONECOUNT <= 1) {
    println("Geef een getal in groter dan 1.");
  }
}

public void GenerateSolutions(int x)
{
  //ALGORITME VOOR HET CREEEREN VAN ALLE MOGELIJKE TOPO MATRICES
  //in solution de topomatrix en de polish expression opslaan
  //de herhalingspermutatie in een ander tabblad steken

// (1) POLISH EXPRESSION

///VARIABELEN

 int zoneperm = (int)(Math.pow(2,(x-1)));
 ArrayList<Integer> templist = new ArrayList<Integer>();
 ArrayList<String> stringlist = new ArrayList<String>();
 String[][] stringarray = new String[x - 1][zoneperm];
 ArrayList<String> polishlist = new ArrayList<String>();
```

```
///OPERATORMATRIX
//// permutatie met herhaling van (x-1)keer kiezen uit 2 elementen (0 en 1) (2^(x-1))

char Operators[] = "VH".toCharArray();
Variation variation = new Variation( Operators.length, x-1 );
while ( variation.hasMore() )
{
    int[] v = variation.next();
    for ( int j = 0; j < v.length; j++ )
    {
        stringlist.add(Character.toString( Operators[ v[j] ] ));
    }
    }

//// alles ordenen in een matrix - per kolom een reeks V's en H's

int counterint = 0;

for (int i = 0; i < zoneperm; i++){
  for (int j = 0; j < (x - 1);j++){
    stringarray[j][i] = stringlist.get(j+counterint);
  }
  counterint = counterint + (x -1);
}

///CIJFERMATRIX
//// eerst gepermuteerde lijst maken van alle mogelijke combinaties van (ZONECOUNT) aantal cijfers

    ArrayList<Integer> inputlist = new ArrayList<Integer>();
    Permutation permutation = new Permutation( x );
    while ( permutation.hasMore() )
{
    int[] p = permutation.next();
    for ( int j = 0; j < p.length; j++ )
    {
        inputlist.add(p[j]);
    }
}

////vervolgens de lijst opslaan in een matrix

int[][] intarray = new int[x][(inputlist.size()) / x];
    for (int co = 0; co < (inputlist.size()) / x ; co++)
    {
      for (int ro = 0; ro < x ; ro++)
      {
        intarray[ro][co] = inputlist.get(ro + x * co);
      }
    }

///POLISH MATRIX OPSTELLEN

////loop over elke kolom in de cijfermatrix
for (int y = 0; y < intarray[0].length;y++){

  //loop over elke kolom in de operatormatrix
  for (int s = 0; s<zoneperm;s++){

    //variabelen

    int numbercount = x;
    int slicescount = x - 1;
    int polishcount = (2*x)-1;
    int[][] sliceact = new int[1][1];
    sliceact[0][0] = 1;
    int[][] numberact = new int[1][1];
    numberact[0][0] = 2;
    String[][] polish = new String[polishcount][1];

    //loop over elk tussenliggende tekens (behalve eerste 3 en laatste) met 3 regels
    for (int i = 3; i<(numbercount * 2)-2;i++){
      int polishlength = polish[0].length;
```

```
  //loop over elke kolom in de polishmatrix
  for (int j = 0; j < polishlength ;j++){

    // regel 1
    if (sliceact[0][j] == numberact[0][j]-1){
      polish[i][j] = Integer.toString(intarray[numberact[0][j]][y]);
      numberact[0][j]++;
    }

    // regel 2
    else{
      if (numberact[0][j] < (numbercount -1) && sliceact[0][j] < (slicescount - 1)){

        // tijdelijke matrices om numberact, sliceact en polish te vergroten
        int[][] tempnumberact = new int[1][numberact[0].length+1];
        int[][] tempsliceact = new int[1][sliceact[0].length+1];
        String[][] temppolish = new String[polish.length][polish[0].length+1];

        // koppieer de waarden van de referentiematrixen naar de tijdelijke
        for (int g = 0; g < polish.length ; g++){
         for (int h = 0; h < polish[0].length ;h++){
           temppolish[g][h] = polish[g][h];
         }
         temppolish[g][polish[0].length] = polish[g][j];
        }
        for (int h = 0; h < polish[0].length;h++){
          tempsliceact[0][h] = sliceact[0][h];
          tempnumberact[0][h] = numberact [0][h];
        }
        tempsliceact[0][polish[0].length] = sliceact[0][j];
        tempnumberact[0][polish[0].length] = numberact[0][j];

        // sla de tijdelijk op als de referentiematrixen
        polish = temppolish;
        sliceact = tempsliceact;
        numberact = tempnumberact;

        // steeds 2 oplossingen
        polish[i][j] = Integer.toString(intarray[numberact[0][j]][y]);
        numberact[0][j]++;
        polish[i][polish[0].length-1] = stringarray[sliceact[0][polish[0].length-1]][s];
        sliceact[0][polish[0].length-1]++;
      }

      // regel 3
      else{
        if (sliceact[0][j] == slicescount - 1){
          polish[i][j] = Integer.toString(intarray[numberact[0][j]][y]);
          numberact[0][j]++;
        }
        if (numberact[0][j] == numbercount -1){
          polish[i][j] = stringarray[sliceact[0][j]][s];
          sliceact[0][j]++;
        }
      }
    }
  }
}

    for (int t = 0; t < polish[0].length;t++){

// de eerste 2 tekens zijn een cijfer
polish[0][t] = Integer.toString(intarray[0][y]);
polish[1][t] = Integer.toString(intarray[1][y]);

// het 3e teken is een slice
polish[2][t] = stringarray[0][s];

// het laatste teken is een slice
polish[polishcount - 1][t] = stringarray[slicescount - 1][s];
}

//omzetten naar 1 string en in een lijst opslagen
int polishlength2 = polish[0].length;
```

```
    for (int q = 0; q < polishlength2 ; q++){
      String defstring = "";
      for (int i = 0; i < polishcount ; i++){
        defstring = polish[polishcount-1-i][q] + defstring;
      }
      // als er VV of HH voorkomt is dit ongeldig
      int tester = 0;
      for (int i = 0; i < polishcount -1; i++){
        if ((polish[i][q].charAt(0)) == (polish[i+1][q].charAt(0))){
          tester = tester + 1;
        }

      }
      if (tester == 0){
        polishlist.add(defstring);
      }
    }
  }
}

println(polishlist.size());

// (1-2) UITDRUKKING OMKEREN

  ArrayList<String> polishinvertlist = new ArrayList<String>();

  for (int m = 0; m < polishlist.size();m++) {
    int getal = polishlist.get(m).length();
    String lijn = "";
    ArrayList<String> invertlist = new ArrayList<String>();

    for (int i = 0; i < getal ; i++){
      invertlist.add(Character.toString(polishlist.get(m).charAt(i)));
    }
    for (int i = 0; i < invertlist.size();i++){
      lijn = invertlist.get(i) + lijn;
    }

    polishinvertlist.add(lijn);
  }

// (2) TOPOLOGISCHE MATRIX

  ArrayList<int[][]> topomatrixlijst = new ArrayList<int[][]>();

/// grote loop voor elke uitdrukking
for (int m = 0; m < polishinvertlist.size();m++) {


  //// tijdelijke lijst met alle tekens apart
  ArrayList<Character> temp = new ArrayList<Character>();

  for ( int i = 0; i < polishinvertlist.get(m).length();i++){
    temp.add(polishinvertlist.get(m).charAt(i));
  }

  //// transformeer deze lijst naar integers (V = -1) en (H = -2)
  ArrayList<Integer> temp1 = new ArrayList<Integer>();

  for( int i = 0; i < temp.size(); i++){
    if (Character.isDigit(temp.get(i))){
      temp1.add(Integer.parseInt(Character.toString(temp.get(i))));
    }
    else{
      if (temp.get(i) == ("V").charAt(0)){
        temp1.add(-1);
      }
      else{
        temp1.add(-2);
      }
    }
  }

  //// lijst van matrices
```

```
  ArrayList<int[][]>matrixlijst = new ArrayList<int[][]>();

  for (int i = 0; i < temp1.size(); i++){
    int[][] a = new int[1][1];
    a[0][0] = temp1.get(i);
    matrixlijst.add(a);
  }

  //// de loop om topologische matrix op te stellen
  int splitsen = 1;

  while (splitsen > 0){
    for (int i = 0; i < (matrixlijst.size() - 2) ; i ++){

      // verticale split
      if     (matrixlijst.get(i)[0][0]     ==     -1     &&     matrixlijst.get(i+1)[0][0]     >=0     &&
matrixlijst.get(i+2)[0][0] >=0){
        int rij1 = matrixlijst.get(i+1).length;
        int kolom1 = matrixlijst.get(i+1)[0].length;
        int rij2 = matrixlijst.get(i+2).length;
        int kolom2 = matrixlijst.get(i+2)[0].length;

        int rij3 = 0;
        int kolom3 = 0;

        // gelijk aantal rijen
        if(rij1 == rij2){
          rij3 = rij2;
          kolom3 = kolom1 + kolom2;
          int[][]change = new int[rij3][kolom3];

          for( int r = 0; r < rij1;r++){
            for (int k = 0; k < kolom1; k++){
              change[r][k+kolom2] = matrixlijst.get(i+1)[r][k];
            }
          }
          for (int r = 0; r < rij2;r++){
            for (int k =0; k < kolom2 ; k++){
              change[r][k] = matrixlijst.get(i+2)[r][k];
            }
          }

          // matrixlijst aanpassen aan nieuwe matrix
          ArrayList<int[][]> matrixtemp = new ArrayList<int[][]>();

          if (i != 0){
            for (int n = 0; n < i; n++){
              matrixtemp.add(matrixlijst.get(n));
            }
            matrixtemp.add(change);

            if (i+3 < matrixlijst.size()){
              for (int n = i+3; n < matrixlijst.size(); n++){
                matrixtemp.add(matrixlijst.get(n));
              }
            }

            matrixlijst = matrixtemp;
          }
          else{
            matrixtemp.add(change);
            matrixlijst = matrixtemp;
          }
        }

        // verschillend aantal rijen
        if (rij1 != rij2){

          // vind het kleinste getal dat deelbaar is door beide rij-aantallen
          for (int p = 1; p < 999; p++){
            rij3 = p;
            if(rij3 % rij1 == 0 && rij3 % rij2 ==0){
             break;
            }
```

```
          }

          //kolommen optellen
          kolom3 = kolom1 + kolom2;
          int[][] change = new int[rij3][kolom3];

          //matrix 1 verschalen
          int[][]mx1 = new int[rij3][kolom1];
          int scale1 = rij3/rij1;

          for (int r = 0; r < rij1; r++){
            for (int k =0; k < kolom1 ; k++){
              for (int q =0; q < scale1; q++){
                mx1[q + (scale1 * r)][k] = matrixlijst.get(i+1)[r][k];
              }
            }
          }

          //matrix 2 verschalen
          int[][]mx2 = new int[rij3][kolom2];
          int scale2 = rij3 / rij2;

          for (int r = 0; r < rij2; r++){
           for (int k = 0; k < kolom2; k++){
            for (int q = 0; q < scale2; q++){
              mx2[q + (scale2*r)][k] = matrixlijst.get(i+2)[r][k];
            }
           }
          }

          //matrix change = matrix 1 + matrix 2

          for (int r = 0; r < mx1.length;r++){
            for (int k = 0; k < mx1[0].length;k++){
              change[r][k+kolom2] = mx1[r][k];
            }
          }

          for (int r = 0; r < mx2.length;r++){
            for (int k = 0; k < mx2[0].length;k++){
              change[r][k] = mx2[r][k];
           }
          }

          //matrixlijst aanpassen met de nieuwe matrix
          ArrayList<int[][]> matrixtemp = new ArrayList<int[][]>();

          if (i!=0){
            for (int n =0; n < i; n++){
              matrixtemp.add(matrixlijst.get(n));
            }
            matrixtemp.add(change);

            if (i + 3 < matrixlijst.size()){
             for (int n = i+3; n < matrixlijst.size();n++){
              matrixtemp.add(matrixlijst.get(n));
             }
            }
            matrixlijst = matrixtemp;
          }
          else{
            matrixtemp.add(change);
            matrixlijst = matrixtemp;
          }
        }
      }

      // horizontale split
      if    (matrixlijst.get(i)[0][0]    ==    -2    &&    matrixlijst.get(i+1)[0][0]    >=    0    &&
matrixlijst.get(i+2)[0][0] >=0){
        int rij1 = matrixlijst.get(i+1).length;
        int kolom1 = matrixlijst.get(i+1)[0].length;
        int rij2 = matrixlijst.get(i+2).length;
        int kolom2 = matrixlijst.get(i+2)[0].length;
```

```
int rij3 = 0;
int kolom3 = 0;

// gelijk aantal kolommen
if (kolom1 == kolom2){
  kolom3 = kolom2;
  rij3 = rij1+rij2;
  int[][] change = new int[rij3][kolom3];

  for (int r = 0; r  <rij2 ; r++){
   for (int k = 0; k < kolom2 ; k++){
     change[r+rij1][k] = matrixlijst.get(i+2)[r][k];
   }
  }
  for (int r = 0; r < rij1;r++){
    for (int k =0; k < kolom1; k++){
     change[r][k] = matrixlijst.get(i+1)[r][k];
    }
  }

  //matrixlijst aanpassen met de nieuwe matrix
  ArrayList<int[][]> matrixtemp = new ArrayList<int[][]>();

  if (i != 0){
   for (int n = 0; n < i; n++){
    matrixtemp.add(matrixlijst.get(n));
   }
   matrixtemp.add(change);

   if( i+3 < matrixlijst.size()){
    for (int n = i+3; n < matrixlijst.size();n++){
     matrixtemp.add(matrixlijst.get(n));
    }
   }
   matrixlijst = matrixtemp;
  }
  else{
    matrixtemp.add(change);
    matrixlijst = matrixtemp;
  }
}

// verschillend aantal kolommen
if (kolom1 != kolom2){

  //vind het kleinste getal dat deelbaar is door beide kolom-aantallen
  for (int p = 1; p < 999; p++){
    kolom3 = p;
    if (kolom3 % kolom1 == 0 && kolom3 % kolom2 ==0){
     break;
    }
  }

  //rijen optellen
  rij3 = rij1 + rij2;
  int[][]change = new int[rij3][kolom3];

  //matrix 1 verschalen
  int[][] mx1 = new int[rij1][kolom3];
  int scale1 = kolom3 / kolom1;

  for (int r = 0; r<rij1;r++){
   for (int k = 0; k < kolom1;k++){
    for (int q = 0; q<scale1;q++){
      mx1[r][q+(scale1*k)] = matrixlijst.get(i+1)[r][k];
    }
   }
  }

  //matrix 2 verschalen
  int[][] mx2 = new int[rij2][kolom3];
  int scale2 = kolom3 / kolom2;
```

```
          for (int r = 0; r < rij2; r++){
           for (int k = 0; k < kolom2; k++){
            for (int q = 0; q < scale2; q ++){
             mx2[r][q+(scale2*k)] = matrixlijst.get(i+2)[r][k];
            }
           }
          }

          //matrix change = matrix 1 + matrix 2
          for (int r = 0; r < mx2.length ; r++){
           for (int k = 0; k < mx2[0].length ; k++){
            change[r+rij1][k] = mx2[r][k];
           }
          }

          for (int r = 0; r < mx1.length ; r++){
            for (int k = 0; k < mx1[0].length;k++){
             change[r][k] = mx1[r][k];
            }
          }

          //matrixlijst aanpassen met de nieuwe matrix
          ArrayList<int[][]> matrixtemp = new ArrayList<int[][]>();

          if(i!=0){
            for (int n = 0; n < i; n++){
              matrixtemp.add(matrixlijst.get(n));
            }
            matrixtemp.add(change);

            if (i+3 < matrixlijst.size()){
              for (int n = i + 3; n < matrixlijst.size();n++){
                matrixtemp.add(matrixlijst.get(n));
              }
            }
            matrixlijst = matrixtemp;
          }

          else{
            matrixtemp.add(change);
            matrixlijst = matrixtemp;
          }
        }
      }
    }

    // stop while loop
    ArrayList<Integer> t = new ArrayList<Integer>();

    for (int w = 0; w < matrixlijst.size(); w++){
     if (matrixlijst.get(w)[0][0] < 0){
      t.add(w);
     }
    }
    splitsen = t.size();
  }

  //voor elke uitdrukking een oplossing wegschrijven naar de lijst
  topomatrixlijst.add(matrixlijst.get(0));

}


// (3) ADJACENCY MATRIX

///loopen voor elke topomatrix
  ArrayList<int[][]> adjacencymatrixlist = new ArrayList<int[][]>();

  for (int m = 0; m < topomatrixlijst.size(); m++){

    int[][] d = new int[x][x];

    // stel de relaties in
    for (int i = 0; i < topomatrixlijst.get(m).length;i++){
```

```
   for (int j = 0; j < topomatrixlijst.get(m)[0].length;j++){
    if ((i+1) < topomatrixlijst.get(m).length){
      d[topomatrixlijst.get(m)[i][j]][topomatrixlijst.get(m)[i+1][j]] = 1;
      d[topomatrixlijst.get(m)[i+1][j]][topomatrixlijst.get(m)[i][j]] = 1;
    }
    if ((i-1) >=0){
      d[topomatrixlijst.get(m)[i][j]][topomatrixlijst.get(m)[i-1][j]] = 1;
      d[topomatrixlijst.get(m)[i-1][j]][topomatrixlijst.get(m)[i][j]] = 1;
    }
    if ((j+1) < topomatrixlijst.get(m)[0].length){
      d[topomatrixlijst.get(m)[i][j]][topomatrixlijst.get(m)[i][j+1]] = 1;
      d[topomatrixlijst.get(m)[i][j+1]][topomatrixlijst.get(m)[i][j]] = 1;
    }
    if ((j-1) >=0){
      d[topomatrixlijst.get(m)[i][j]][topomatrixlijst.get(m)[i][j-1]] = 1;
      d[topomatrixlijst.get(m)[i][j-1]][topomatrixlijst.get(m)[i][j]] = 1;
    }
   }
  }

  // topomatrix in lijst
  ArrayList<Integer> tempi = new ArrayList<Integer>();

  for (int i = 0; i < topomatrixlijst.get(m).length; i++){
   for (int j = 0; j < topomatrixlijst.get(m)[0].length; j++){
     tempi.add(topomatrixlijst.get(m)[i][j]);
   }
  }

  // groottes van de ruimtes
  for (int i = 0; i < d.length ; i++){
    for (int j = 0; j < d[0].length ; j++){
      if (i == j){
       Integer count = 0;
       ArrayList<Integer> temp2count = new ArrayList<Integer>();

       for (int p = 0; p < tempi.size(); p++){
         if (tempi.get(p) == i){
           temp2count.add(1);
         }
       }
       count = temp2count.size();
       d[i][j] = count;
      }
    }
  }

  // sla elke adjacencymatrix op
  adjacencymatrixlist.add(d);

}

// (4) TEST OF HET VOLDOET

///VOORBEELD INPUTMATRIX (moet nog vervangen worden met GUI)
  int[][]inputmatrix = new int[x][x];
  for(int i = 0; i < inputmatrix.length;i++){
   for (int j = 0; j < inputmatrix[0].length;j++){
    inputmatrix[i][j] = -1;
   }
  }
  inputmatrix[0][1] = 1; //ruimte 0 en 1 hebben een relatie
  inputmatrix[0][2] = 1; //ruimte 0 en 2 hebben geen relatie
  inputmatrix[0][3] = 1; //ruimte 0 en 2 hebben geen relatie
  inputmatrix[0][5] = 0;
  inputmatrix[2][3] = 1;
  inputmatrix[3][5] = 1;
  inputmatrix[2][4] = 1;
  inputmatrix[1][2] = 1;


///VOORBEELD INPUTMATRIX2 (moet nog vervangen worden met GUI)
  String[][] inputmatrix2 = new String[x][x];
  for(int i = 0; i < inputmatrix2.length;i++){
```

```
     for (int j = 0; j < inputmatrix2[0].length;j++){
      inputmatrix2[i][j] = "-";
     }
    }
    inputmatrix2[0][5] = ">"; //ruimte 0 is kleiner dan ruimte 1
    //inputmatrix2[0][2] = "<"; //ruimte 0 is kleiner dan ruimte 2


  ///EFFECTIEVE CODE
  //// voor inputmatrix 1
    ArrayList<Boolean>testlijst = new ArrayList<Boolean>();

    //loop voor elke adj matrix
    for (int m = 0 ; m < adjacencymatrixlist.size();m++){
      Integer testerwaarde = 0;
      Integer teller = 0;
      Boolean pingtest = false;

      for (int i = 0; i < adjacencymatrixlist.get(m).length;i++){
        for (int j = 0; j < adjacencymatrixlist.get(m)[0].length;j++){
          if (inputmatrix[i][j] == -1){
           teller++;
          }
          if (inputmatrix[i][j] != -1 && adjacencymatrixlist.get(m)[i][j] == inputmatrix[i][j]){
            testerwaarde++;
          }
        }
      }

      if           (testerwaarde           ==           ((adjacencymatrixlist.get(m).length)           *
(adjacencymatrixlist.get(m)[0].length)) - teller){
       pingtest = true;
      }

      testlijst.add(pingtest);
    }

  //// voor inputmatrix 2
    ArrayList<Boolean>testlijst2 = new ArrayList<Boolean>();

    // loop voor elke adj matrix
    for (int m = 0; m < adjacencymatrixlist.size();m++){
      Integer testerwaarde = 0;
      Integer teller = 0;
      Boolean pingtest = false;

      for (int i = 0; i < adjacencymatrixlist.get(m).length;i++){
       for (int j = 0; j < adjacencymatrixlist.get(m)[0].length;j++){
        if (inputmatrix2[i][j] == "-"){
          teller++;
        }
        if (inputmatrix2[i][j] != "-" && inputmatrix2[i][j] == "<" && adjacencymatrixlist.get(m)[i][i]
< adjacencymatrixlist.get(m)[j][j]){
          testerwaarde++;
        }
        if (inputmatrix2[i][j] != "-" && inputmatrix2[i][j] == ">" && adjacencymatrixlist.get(m)[i][i]
> adjacencymatrixlist.get(m)[j][j]){
          testerwaarde++;
        }
       }
      }
      if           (testerwaarde           ==           ((adjacencymatrixlist.get(m).length)           *
(adjacencymatrixlist.get(m)[0].length)) - teller){
       pingtest = true;
      }

      testlijst2.add(pingtest);
    }

    //OVERGEBLEVEN AANTAL OPLOSSINGEN

    Integer teltrue = 0;
    for (int i =0; i < testlijst.size();i++){
      if (testlijst.get(i) == true){
```

```
        teltrue++;
      }
    }

    Integer teltrue2 = 0;
    for (int i =0; i < testlijst2.size();i++){
      if (testlijst2.get(i) == true){
        teltrue2++;
      }
    }

    Integer teltrue3 = 0;
    for (int i =0; i < testlijst2.size();i++){
      if (testlijst2.get(i) == true && testlijst.get(i) == true){
        teltrue3++;
      }
    }


  //VOORBEELD PRINT POLISH EXPRESSION EN TOPOMATRIX EN ADJACENCYMATRIX EN TEST AAN CONSTRAINT
  for (int testwaarde = 0; testwaarde < 15000; testwaarde++){
    if(testlijst.get(testwaarde)==true){
  println();
  println(polishlist.get(testwaarde));

  for (int i = 0; i < topomatrixlijst.get(testwaarde).length;i++){
    for (int j = 0; j < topomatrixlijst.get(testwaarde)[0].length;j++){
      print(topomatrixlijst.get(testwaarde)[i][j]);
    }
    println();
  }
  println();

  for (int i = 0; i < adjacencymatrixlist.get(testwaarde).length;i++){
    for (int j = 0; j < adjacencymatrixlist.get(testwaarde)[0].length;j++){
      print(adjacencymatrixlist.get(testwaarde)[i][j]);
    }
    println();
  }
  println();
  println(testlijst.get(testwaarde));
  println(testlijst2.get(testwaarde));
  println();
  }
  }

  println("overgebleven oplossingen RELATIE = " + teltrue);
  println("overgebleven oplossingen GROOTTE = " + teltrue2);
  println("overgebleven oplossingen SAMEN = " + teltrue3);

}
```

8. BIBLIOGRAFIE

A., N., Shaw, J., & Simon, H. (1959). *Report on a General Problem-solving program.* Pittsburgh, PA, United States: Carnegie Institute of Technology.

Autodesk. (2003). *Building Information Modeling.* San Rafael, CA.

Bell, M. (1989). From 2D computer-aided draughting to 2D and 3D computer-aided design. *Design Studies* , 112-117.

Cagdas, G. (1996). A shape grammar model for designing row-houses. *Design Studies 17* , 35-51.

Claus, H. (2009). De huizen, een concept voor de Belgische gevangenis van de eenentwintigste eeuw. *De orde van de dag (48)* , 39-43.

Duarte José, e. a. (2007). Unveiling the structure of the Marrakech Medina: A shape grammar and an interpreter for generating urban form. *Artificial Intelligence for Engineering, Desing, Analysis and Manufacturing* , 317-349.

Duarte, J. P. (2001). *Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses.* Massachusetts: Massachusetts Institute of Technology.

Duarte, J. P. (1993). *Order and Diversity Within a Modular Search for Housing: A Computational Approach.* Massachusetts: Massachusetts Institute of Technology.

Duarte, J. (2005). Towards the mass customization of housing: the grammar of Siza's houses at Malagueira. *Environment and Planning B: Planning and Design* , 347-380.

Eastman, C. P. (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors.* Hoboken, New Jersey.

Habib, Y., Sait, S., & Adiche, H. (2001). Evolutionary algorithms, simulated anneealing and tabu search: a comparative study. *Engineering Applications of Artificial Intelligence 14* , 167-181.

HELME, L., DERIX, C., & GAMLESÆTER, Å. (2012). *Spatial Configuration: Semi-Automatic Methods for Layout Generation in Practice.* London: Aedas|R&D, Aedas London.

Kalay, Y. E. (1985). Redefining the role of computers in architecture: from drafting/modelling tools to knowledge-based design assistants. *Computer-Aided Design* , 319-328.

Kilian, A. (2006). *Design exploration through bidirectional modeling of constraints.*

Krish, S. (2011). A practical generative design method. *Computer-Aided Design 43 (1)* , 88-10.

Lawson, B. (1998). Towards a computer-aided architectural design process: a journey of several mirages. *Computers in Industry 35* , 47-57.

Liggett, R. S. (2000). Automated facilities layout: past, present and future. *Automation in Construction, 9(2)* , 197 - 215.

Maeda, J. (2001). *Design by Numbers.* MIT Press.

National Building Information Model Standard Project Committee. (sd). *F.A.Q.* Opgeroepen op juli 14, 2013, van Buidling Smart Alliance: http://www.buildingsmartalliance.org/index.php/nbims/faq/

Oosterhuis, K. (2012). Simply complex, toward a new kind of building. *Frontiers of Architectural Research* , 411-420.

Sait, S. M. (1999). *VLSI Physical Design Automation: Theory and Practice.* Singapore: World Scientific.

Steadman, P. (1983). Architectural morphology: an introduction to the geometry of building plans. Taylor & Francis.

Stiny, G. (1981). A note on the description of designs. *Environment and Planning* , 257-267.

Turrin, M., van Buelow, P., & Rudi, S. (2011). Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Advanced Engineering Informatics 25* , 656-675.

van Nederveen, G. A. (1992). Modelling multiple views on buildings. *Automation in Construction 1(3)* , 215-224.

Wang, L.-T., Chang, Y.-W., & Cheng, K.-T. (. (2009). *Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon).* Morgan Kaufmann.

Wang, Y., & Duarte, J. (2002). Automatic generation and fabrication of designs. *Automation in Construction 11* , 291-302.

Wong, D. F. (1986). A new algorithm for floorplan design. *Design Automation Conf.* , 101-107.