Co[nn]Action

Improve the liveability in the Zaatari Camp by creating accessible & inclusive infrastructure

TU Delft
BK Bouwkunde

Date

02-11-21

Authors

Group 2

Christopher Bierach	5120330
Eren Gozde Anil	5263557
Felipe Lopez	5042844
Jun Wen Loo	5205662
Roy Uijtendaal	4607481

Responsible Professor

Dr. Ir. Pirouz Nourian

Instructors

Dr. Ir. Pirouz Nourian
Dr. Ir. Fred Veer
Dr. Charalampos Andriotis
Ir. Hans Hoogenboom
Ir. Shervin Azadi
Ir. Frank Schnater

TEAM

CHRISTOPHER BIERACH

EREN GOZDE ANIL

FELIPE LOPEZ

JUN WEN LOO

ROY UIJTENDAAL

CONTENTS

Introduction 0.0

INTRODUCTION

0.1 SITE ANALYSIS

Earthy 4.0 is based on Zaatari Refugee Camp whereby the team looked into the prevailing issues of the camp and its inhabitants. The main issues can be narrowed down into 3 main scope:

01_URBAN STRUCTURE

The infrastructure is inadequate in terms of accessibility, safety, and connectivity. Based on the study of the infrastructure within the camp, it can be identified from Fig 0.1.1 that they are mostly clustered around key locations at the periphery of each districts. This form of clustering promotes centralization of functions but also led to the disconnectedness between each individual cluster zones.

Hence the team identifies the need to improve the connectivity between these clusters by providing an accessible and safe route for the inhabitants to take.

02_CULTURE

Cultural aspects within the camp is lacking as the urban disarray and repetition in shelters does not promote a sense of identity as well as a sense of belonging & ownership to the place. As refugee camps are meant to be temporary solutions, the focus on promoting rootedness to the location is not a priority. However as most inhabitants stay within the camp for an average of 17 years, these temporal arrangements gradually become a longer term situation.

Therefore, in order to provide for a culturally viable environment for Zaatari inhabitants to have a stronger sense of identity, it is pertinent to explore how their culture can be allowed to flourish within the camp.

03_ACTIVITY

The range of activities and diverse spaces in the camp is insufficient to meet the demand of being occupied. With a lack of employment opportunities, common and communal spaces within and outside the camp for refugees, the refugees are not provided enough avenues for them to hone their skills or engage in productive and leisure endeavours.

Thus in order to keep them occupied, the intervention needs to focus on spaces which allows for exchange of skills and goods while providing public and leisure spaces for the inhabitants of Zaatari Camp.

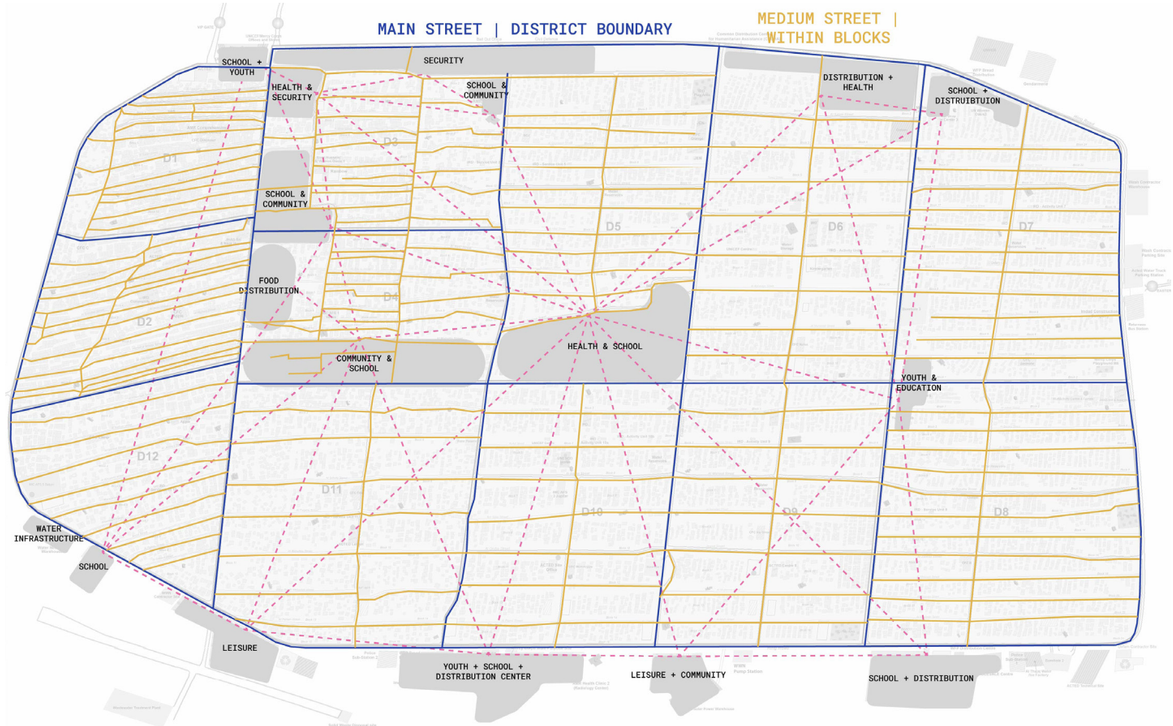


FIG 0.1.1 | ROADS AND CLUSTER OF FUNCTIONS ANALYSIS, DEPICTING AREAS OF POSSIBLE CONNECTIONS BASED ON PROXIMITY AND SIMILARITY IN FUNCTIONS

INTRODUCTION

0.2 DESIGN GOALS & STRATEGIES

To address the 3 issues raised, the design goals of the project are:

01 - Provide a system that could help configure the urban development.
The main design strategy of the project is to firstly identify the different streets connecting the cluster of functions and create local streets based on the number of households required to form a community. The locals are then interviewed on what kind of work spaces they wish to have. These requests will then be allocated at key intersections based on a set of priorities such as district, proximity, use frequency and unit size. Covered walkways will also be introduced based on the combinations of spaces lining the streets.

02 - Encourage independence by providing safe and inclusive routes.
Co[nn]action identifies that there are many vulnerable groups such as women and children who feel unsafe traversing the camp to get to their destinations. With different types of streets available in the camp, primarily the paved Main Street and Medium Street as indicated in Fig 0.2.1, these streets are highly public and traversed frequently by people from around the camp. Hence there is a need to introduce smaller streets named Local Street, which connects with Medium Streets to provide a safer, less public path for the locals.

03 - Provide a place for activity and skills development to flourish.
In order to promote the exchange of skills, goods and culture, the intervention most closely resembles a bazaar as reflected in Fig 0.2.2 whereby the street is thickened to provide spaces for work and commercial activities.

FIG 0.2.1 | MAIN AND MEDIUM STREETS AROUND ZAATARI CAMPS WITH LOCAL, UNPAVED STREETS NOT FORMALISED ON THE MAP.

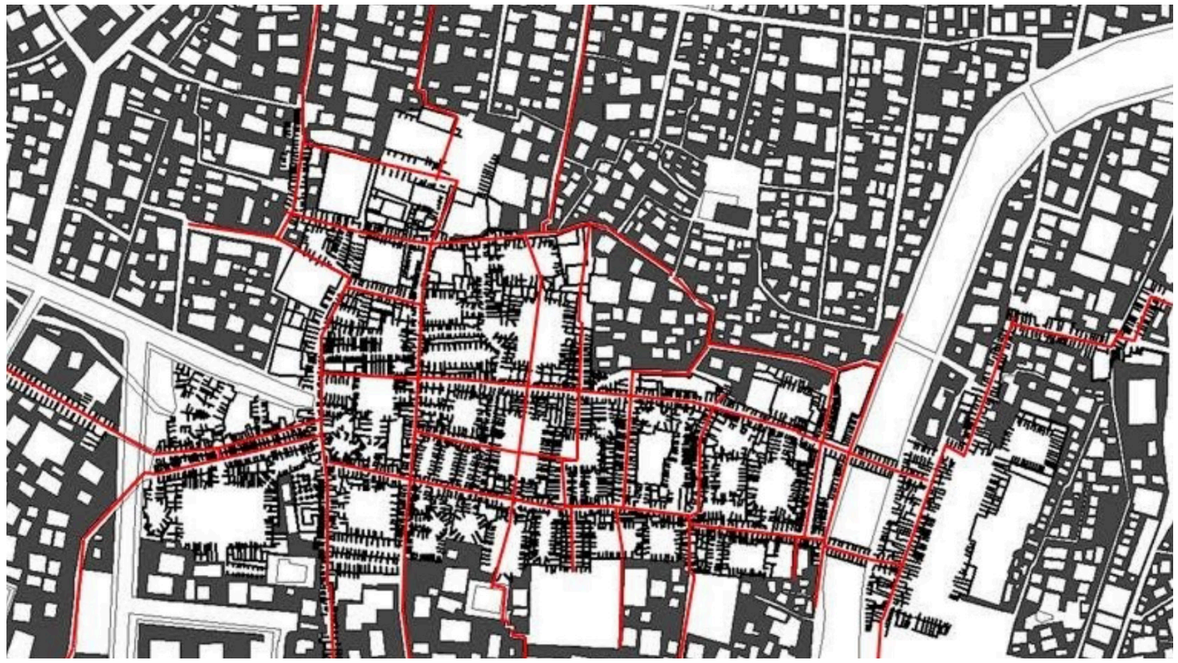


FIG 0.2.2 | BAZAAR TYPOLOGY BY TAGHIZADEHVEHED, NILOUFAR. (2018). A COMPARATIVE STUDY OF COVERED SHOPPING SPACES: COVERED BAZAARS, ARCADES, SHOPPING MALLS.

INTRODUCTION

0.2 DESIGN GOALS & STRATEGIES

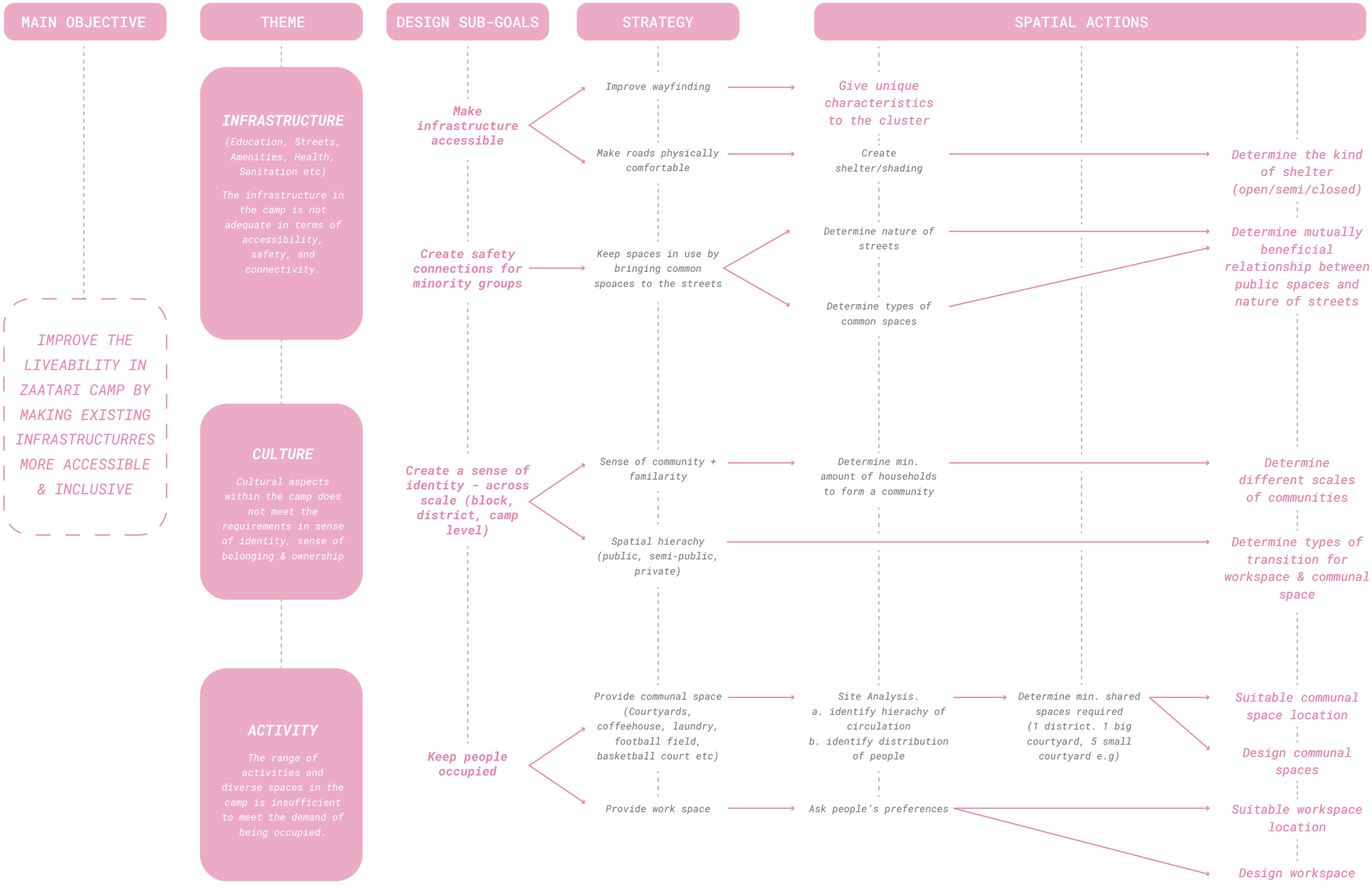
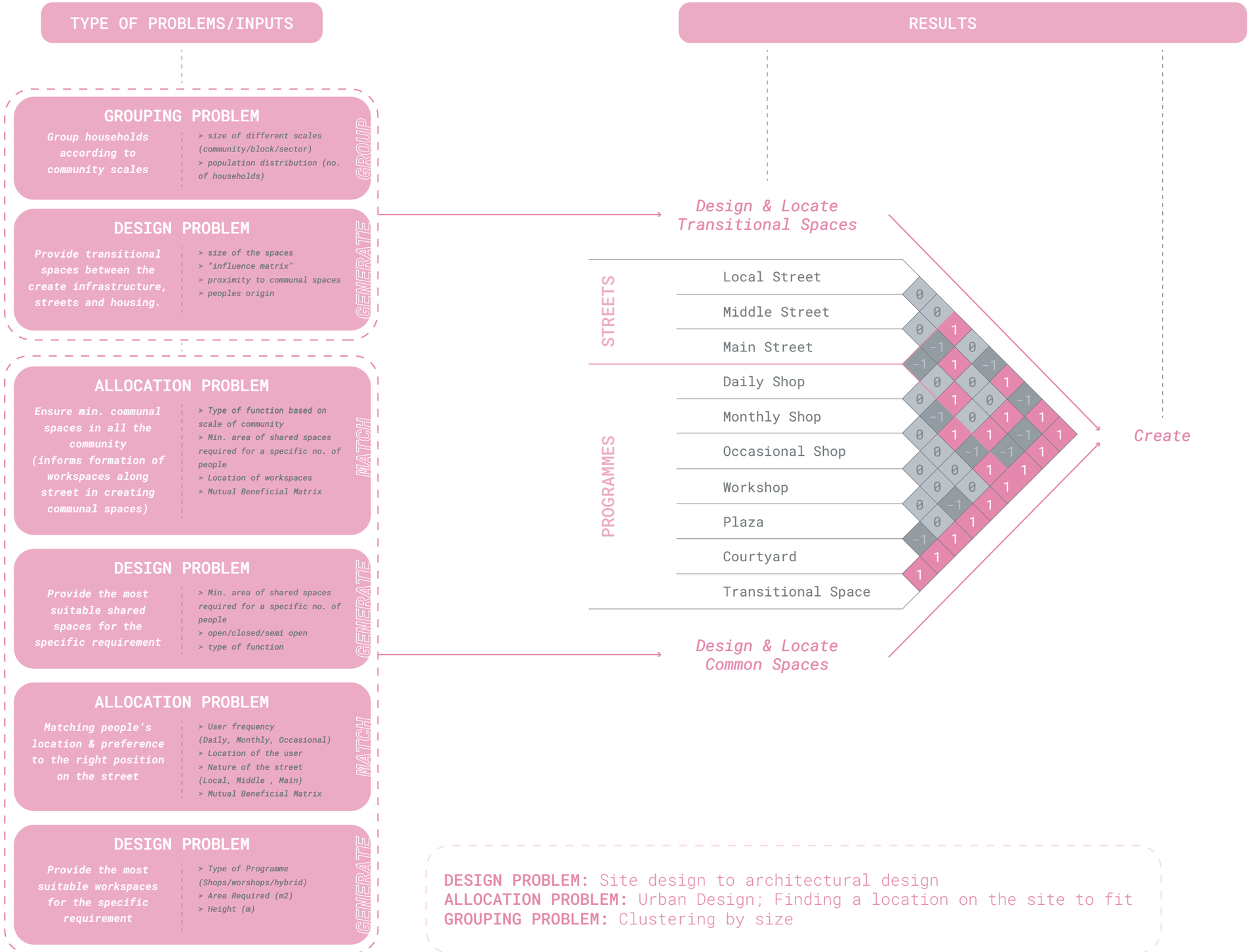


FIG 0.3.1 | DESIGN STRATEGIES BREAKDOWN (PART 1)

INTRODUCTION

0.2 DESIGN GOALS & STRATEGIES

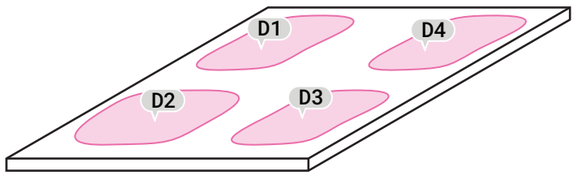


DESIGN PROBLEM: Site design to architectural design
ALLOCATION PROBLEM: Urban Design; Finding a location on the site to fit
GROUPING PROBLEM: Clustering by size

FIG 0.2.1 | DESIGN STRATEGIES BREAKDOWN (PART 2)

INTRODUCTION

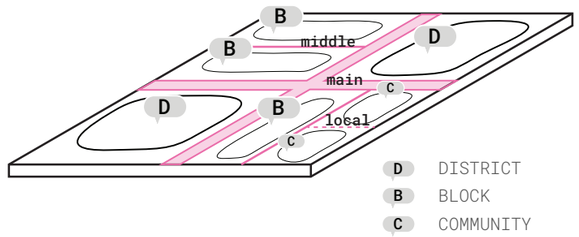
0.2 DESIGN GOALS & STRATEGIES



1. DEFINE DISTRICTS

Problem: **Allocation**

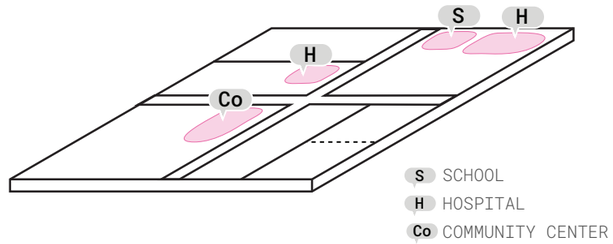
Objective: **Determine main roads**



2. IDENTIFY DIFF. NATURE OF STREETS

Problem: **Allocation**

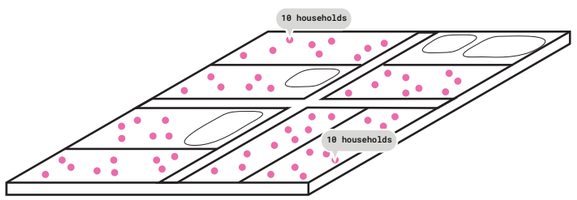
Objective: **Determine what kind of function can be located depending on the nature of the street**



3. IDENTIFY CLUSTERS OF FUNCTIONS

Problem: **Allocation**

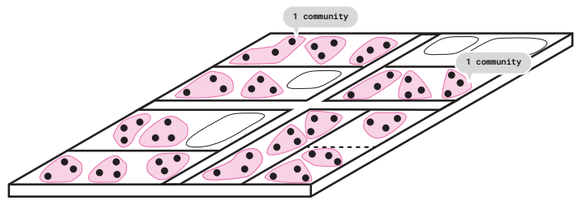
Objective: **Provide the connection between specific amenities**



4. HOUSEHOLD DENSITY DISTRIBUTION

Problem: **Grouping**

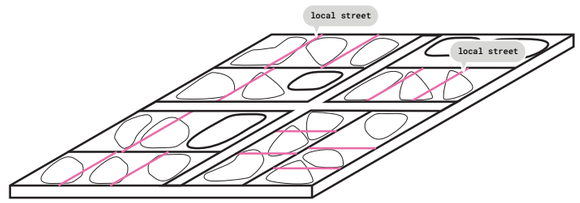
Objective: **Determine population density within blocks**



5. CREATE COMMUNITIES

Problem: **Grouping**

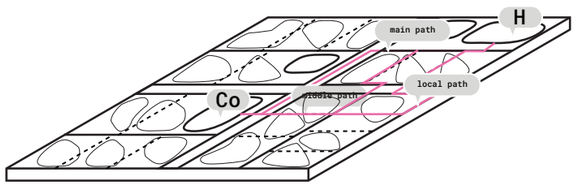
Objective: **Locate the local streets and create communities (16 households)**



6. CREATE LOCAL STREETS

Problem: **Allocation**

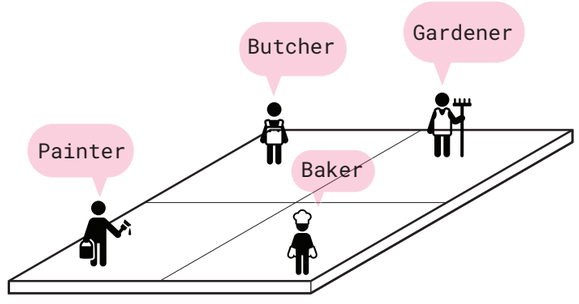
Objective: **Safe paths for vulnerable groups**



7. CREATE ROUTES

Problem: **Allocation**

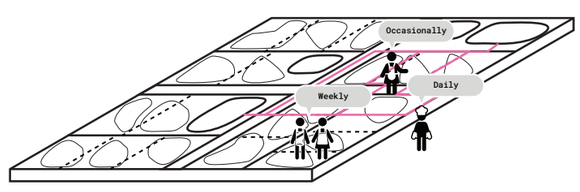
Objective: **Determine different scale of connections bet. existing infrastructure**



8. COLLECT REQUESTS

Problem: **Identify**

Objective: **Detect skills of people**



9. ALLOCATE PROGRAMS

Problem: **Allocation**

Objective: **Organize allocated programs according to REL-Chart**

Configuring 1.0

DESIGN PROBLEMS

1.0 OVERVIEW

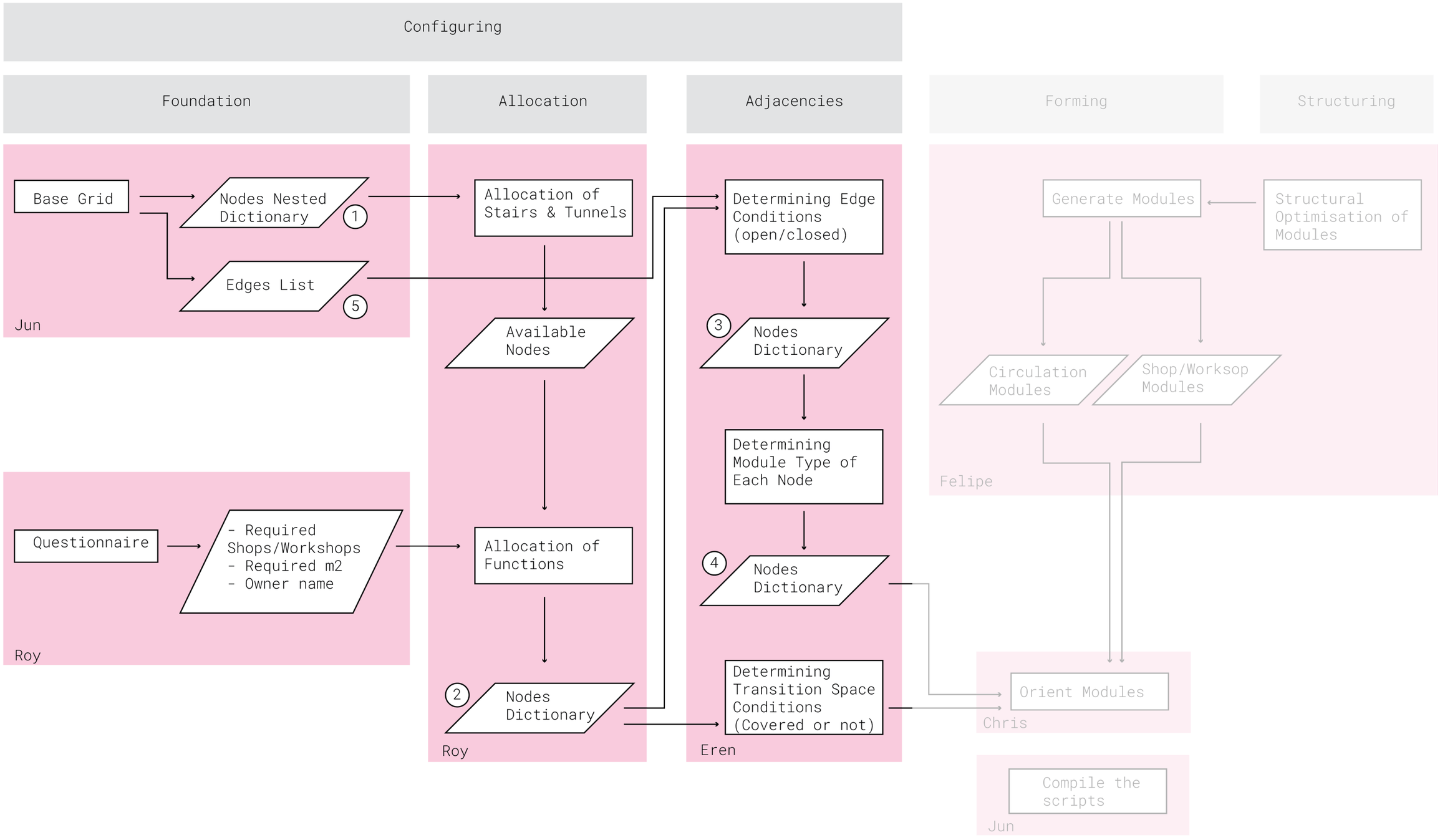


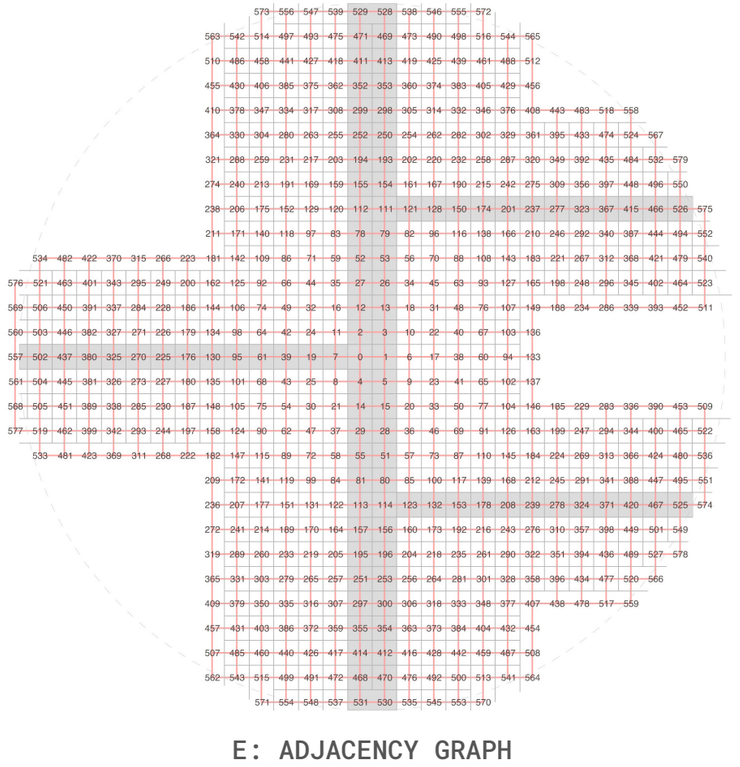
Figure 1.0.1 Complete procces Flowchart

FOUNDATION

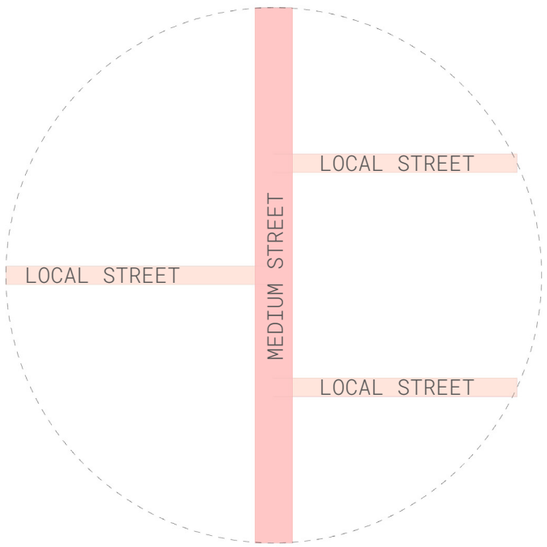
1.1 BASE GRID GENERATION

The site is first abstracted into its street and intersection elements as shown in Fig 1.1 A: **Street Identification**. Based on their nature of accessibility, the Local Street is deemed to be less public than the medium street. Hence the Local Street depth is defined as 3 layers while the Medium Street is 5 layers so as to provide for a better transition from the public nature of each street towards the residential areas behind the shops.

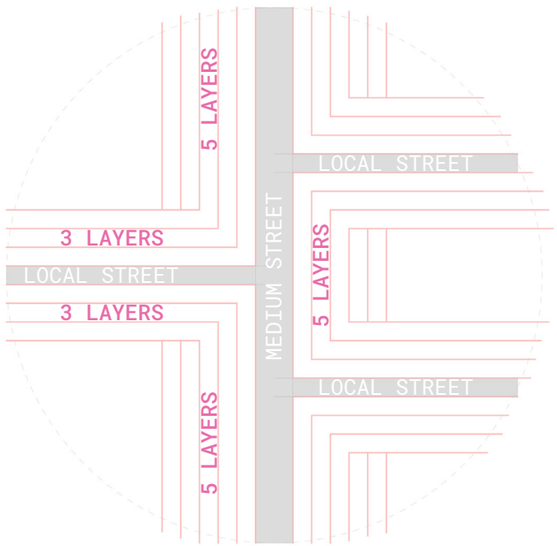
A cell division of 3m was then derived based on street width and minimum shop space of 3x3 = 9m^2. Nodes and adjacency information are then obtained from these cells. A single layer of nodes are added around the grid of cells (as shown in D: Node Generation) to provide a boundary for the grid to inform the follow python processes of Allocation and Adjacency & Module Types to identify boundary conditions. The resulting dual graph is obtained in Rhino-GH with C: Cell Division indicating the boundary of each unit and E: Adjacency Graph indicating



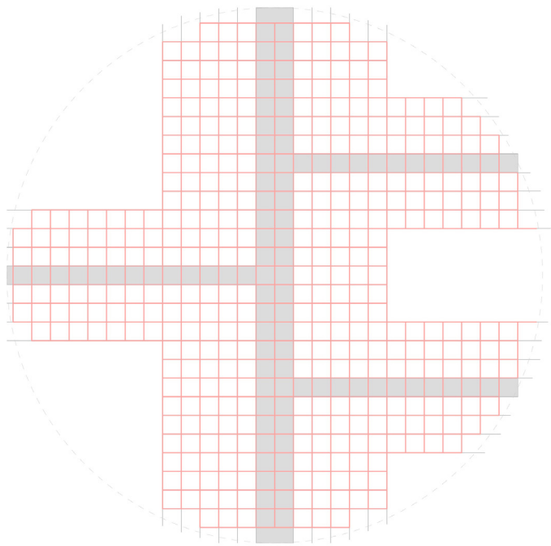
E: ADJACENCY GRAPH



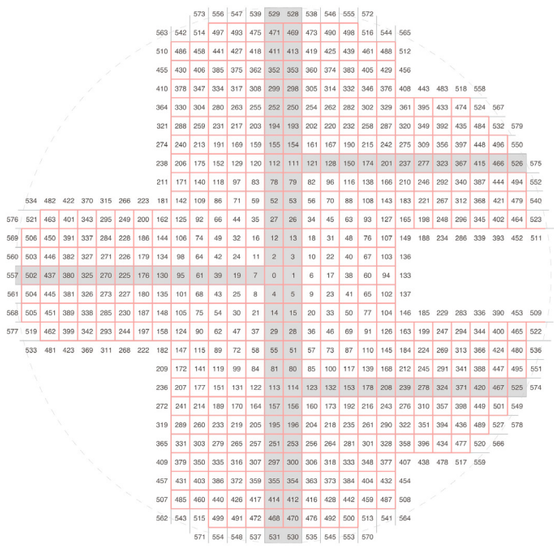
A: STREET IDENTIFICATION



B: STREET DEPTH



C: CELL DIVISION



D: NODE GENERATION

FIG 1.1.1 | BASE GRID GENERATION DIAGRAM

FOUNDATION

1.1 BASE GRID GENERATION | GH & PYTHON FLOWCHART

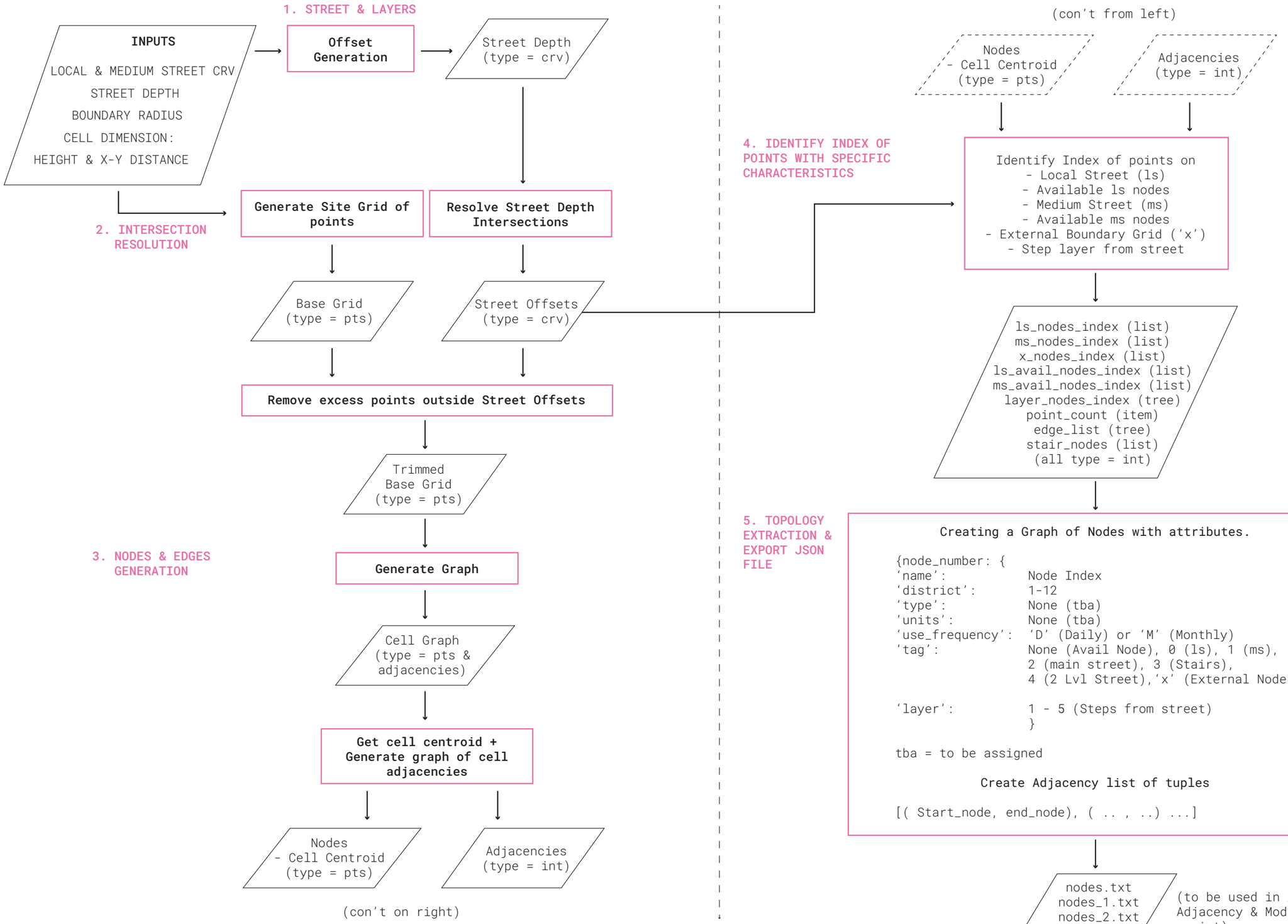


FIG 1.1.2 | FLOWCHART FOR BASE GRID GENERATION IN RHINO-GH & PYTHON.

GATHERING REQUIREMENTS

1.2 QUESTIONNAIRE

In order to get a detailed overview of the refugees' wishes, we want to make it possible for the users to provide direct input in the development of the buildings. In order to limit the degree of influence as little as possible, but to keep the design options manageable within the scope of this course, we have developed a questionnaire consisting of the following five questions.

1. What district are you from?

The starting point for the restructuring method of the infrastructure is based on three scales, of which the district is the largest scale. The aim of the method is to offer residents within a certain district a space that fulfills their needs. The first question aims to cluster residents from the same district, and to have them travel the shortest distance on average from their home to their occupational space.

2. What type of building do you need?

The second question is divided into three answer options. The research showed that the refugees lack a space in which they can trade (*shop*), develop a product or learn a skill (*workshop*), or combine both (*shop and workshop*). Hence, taking into account the scope of the course, we have chosen to only offer shops, workshops, or a combination of both.

3. How many m² do you need?

Thirdly, the users are asked for the desired floor area of the space. To keep the method manageable for this course, a grid of three by three meters is used. This area is occupied by one module of nine square meters floor area, hence only a multiple of nine square meters of floor area can be requested.

3. What is the user frequency?

User frequency refers to how often a particular customer visits the space. A distinction is made between *daily use* (e.g. bakeries, grocery shops, etc.), *monthly use* (e.g. shoemakers, clothing stores, etc.) and *occasional use* (e.g. hardware shops, medical clinics, etc.)

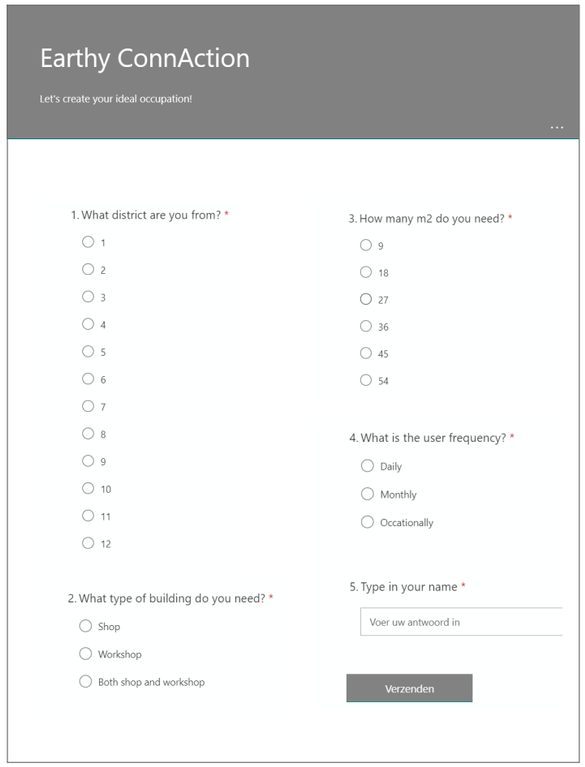


FIGURE 1.2.1: QUESTIONNAIRE SET-UP

Building type	User frequency	District	m2	Tag
Both shop and workshop	**Daily**	**7**	**81**	**Eren**
Shop	Daily	7	9	Jun
Shop	**Monthly**	**7**	**54**	**Roy**
Workshop	Daily	7	9	Felipe
Shop	**Monthly**	**7**	**18**	**Bierocracy**
Both shop and workshop	Daily	7	27	Ahmad
Workshop	**Daily**	**7**	**36**	**Alaa**
Shop	Monthly	7	36	Ali
Workshop	**Daily**	**7**	**9**	**Ameer**
Both shop and workshop	Monthly	7	18	Aamir
Both shop and workshop	**Daily**	**7**	**9**	**Amr**
Shop	Daily	7	9	Anees
Workshop	**Daily**	**7**	**9**	**Awad**
Shop	Monthly	7	9	Ayman
Workshop	**Monthly**	**7**	**27**	**Azhar**
Both shop and workshop	Daily	7	18	Abu Bakr
Workshop	**Daily**	**7**	**27**	**Aasim**
Shop	Monthly	7	36	Ataa
Workshop	**Daily**	**7**	**9**	**Badr Udeen**
Both shop and workshop	Monthly	7	18	Baha Udeen
Shop	**Monthly**	**7**	**9**	**Bahiy Udeen**
Workshop	Daily	7	45	Basel
Shop	**Daily**	**7**	**9**	**Baasim**
Workshop	Daily	7	36	Bishr
Both shop and workshop	**Monthly**	**7**	**18**	**Dawoud**
Workshop	Daily	7	27	Diyaa Udeen
Shop	**Monthly**	**7**	**36**	**Fadl**
Workshop	Monthly	7	27	Fahad
Both shop and workshop	**Daily**	**7**	**36**	**Faakhir**
Both shop and workshop	Monthly	7	9	Fareed
Shop	**Daily**	**7**	**36**	**Faarooq**
Workshop	Monthly	7	18	Fawaz
Shop	**Daily**	**7**	**36**	**Faris**
Workshop	Daily	7	36	Ghaalib
Both shop and workshop	**Monthly**	**7**	**9**	**Ghaazi**
Workshop	Monthly	7	18	Haady
Shop	**Daily**	**7**	**36**	**Hamza**
Workshop	Monthly	7	18	Humam
Both shop and workshop	**Daily**	**7**	**27**	**Haaroon**
Shop	Monthly	7	36	Haashim
Workshop	**Daily**	**7**	**9**	**Haatim**
Shop	Monthly	7	36	Haashim
Shop	**Monthly**	**7**	**36**	**Husaam**
Both shop and workshop	Daily	7	9	Ihsaan
Workshop	**Monthly**	**7**	**36**	**Houd**

FIGURE 1.2.2: QUESTIONNAIRE EXCEL OUTPUT

5. Type in your name

The name tag is solely used to locate the requested space after the configuration.

Assumptions

A number of assumptions have been made for the practical use of the questionnaire. It is assumed that the questionnaire is completed on behalf of a refugee by a professional/staff member within the camp, who can explain and assess based on the refugee's story what should be answered for each question. The final configuration of the bazaar will not take place until enough requests have been made within a certain district. Because the wishes can change significantly over time, we propose to have the questionnaire performed periodically by the staff in the Zaatari camp.

GATHERING REQUIREMENTS

1.2 QUESTIONNAIRE

Dummy data set

In order to reflect reality in the project as detailed as possible, a dummy data set is used. The data set is based on data from the Finn Church Aid and SRD Center (see figure 1.2.3 and figure 1.2.4). The consulted document contains detailed data on livelihoods, current skills and the popularity of different training programs delivered in the Camps. This data was analyzed and converted into dummy data, using the same ratios and percentages.

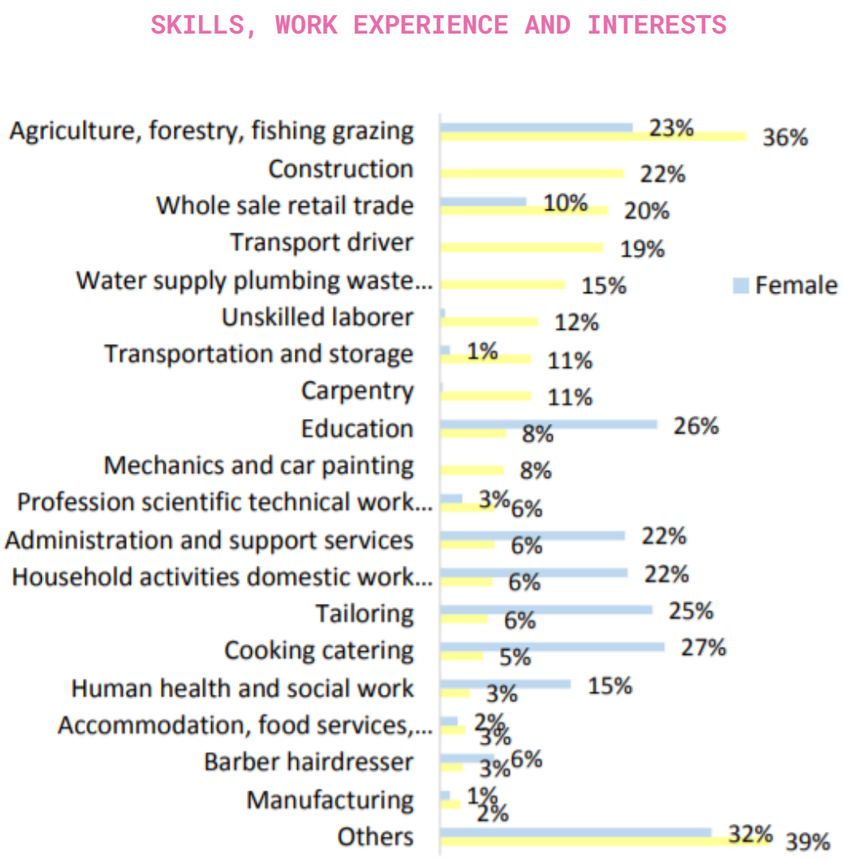


FIGURE 1.2.3: SKILLS, WORK EXPERIENCE AND INTERESTS (FINN CHURCH AID AND SRD CENTER, 2019)

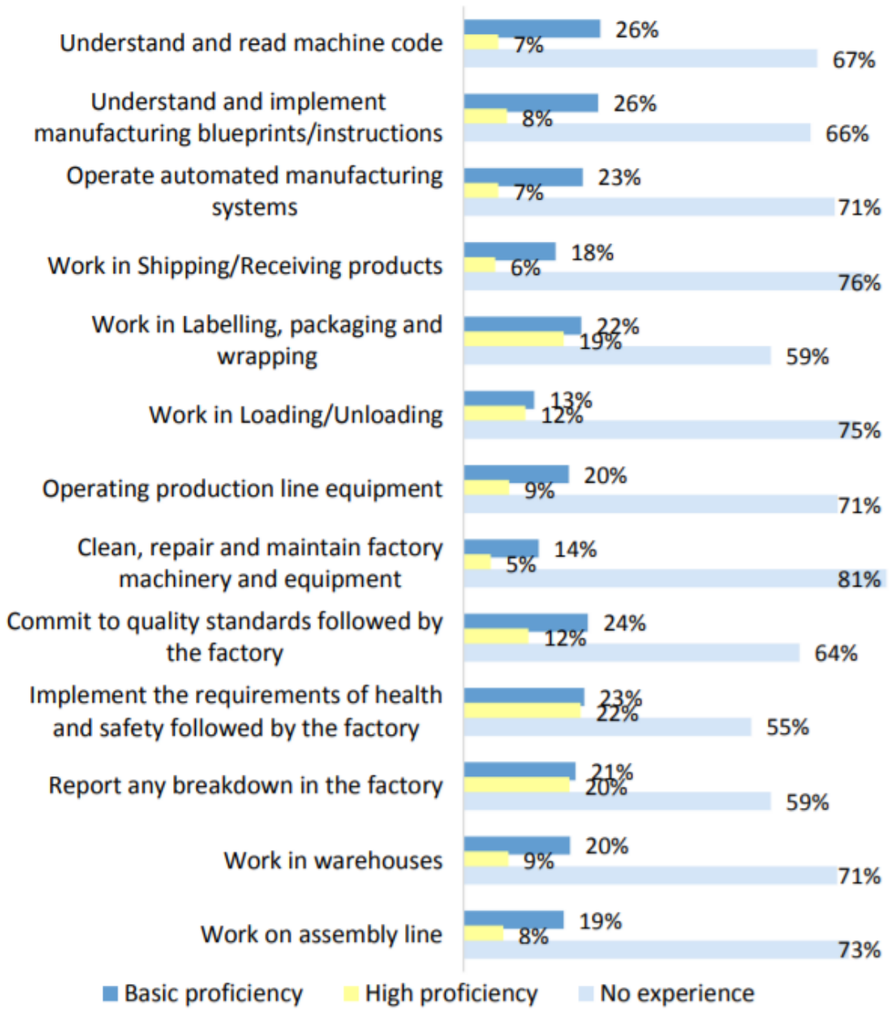


FIGURE 1.2.3: INDUSTRIAL SECTOR (FINN CHURCH AID AND SRD CENTER, 2019)

GATHERING REQUIREMENTS

1.2 QUESTIONNAIRE OUTPUT

The results of the questionnaire are kept in an excel sheet. This data is retrieved by a python script and placed in a list. This list contains the following data:

- The name value, the entering number of the questionnaire, chronological order, starting from 0;
- The number of the district;
- The type of building that is requested;
- The amount of units that is requested;
- The use frequency of the occupation;
- The name of the respondent

For easy use of the requests data, the values are converted into simplified data and the list is converted into a nested dictionary. The keys of the nested dictionary correspond with the name value, and the values of nested dictionary are again a dictionairy containing the following data:

{ ‘name’: the entering number of the questionnaire 0-9999
‘district’: 1-12
‘type’: shop = 1, shop + workshop = 2, workshop = 3
‘units’: the requested area devided by nine, 1-9
‘use_frequency’: daily = D, monthly = M, and occationally = O
‘tag’: the name of the respondent }

Let’s use hamid as an example: Hamid filled in the survey as first respondent, he is from district 7, requests a combination of shop + workshop that is used on a daily basis and the space should cover 27 m^2 / 3 units. The format matches the node dictionary, making the comparison more easy. The output is the requests dictionary:

0: {“name”: 0, “district”: 7, “use_frequency”: “D”, “type”: 2, “units”: 3, “tag”: “Hamid”}

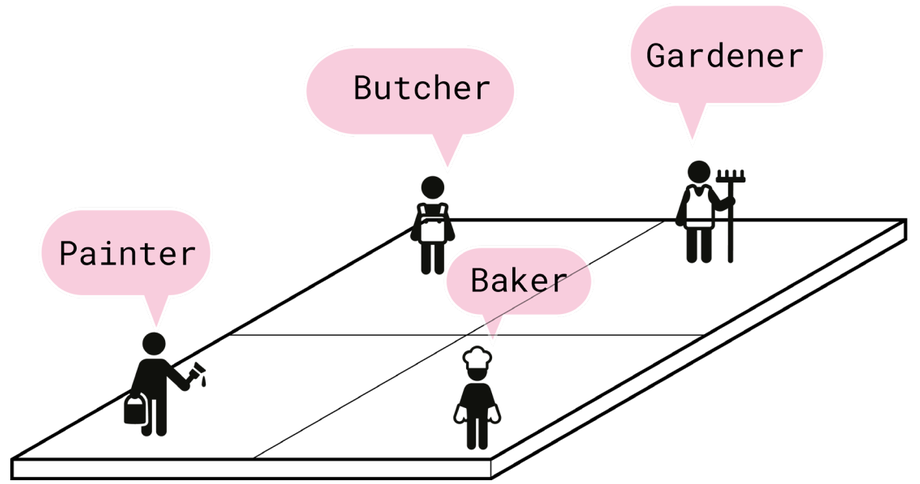


FIGURE 1.2.3: INHABITANT OCCUPATION

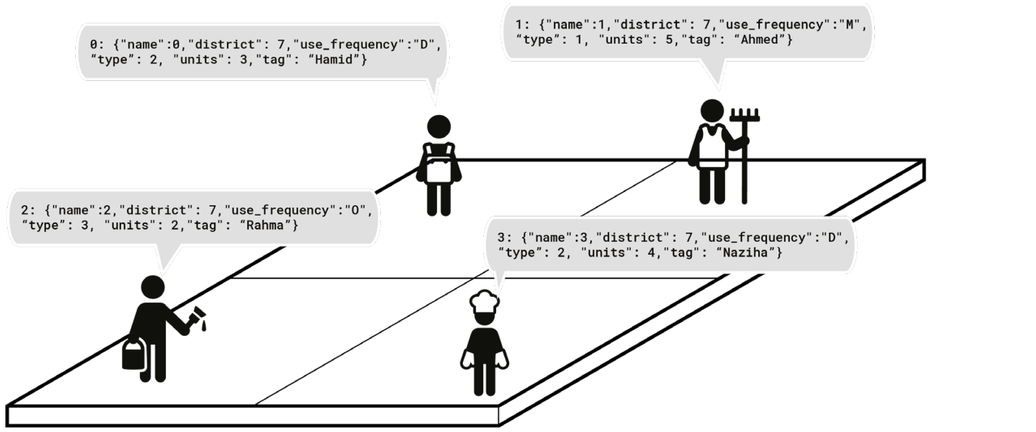


FIGURE 1.2.4: QUESTIONNAIRE OUTPUT

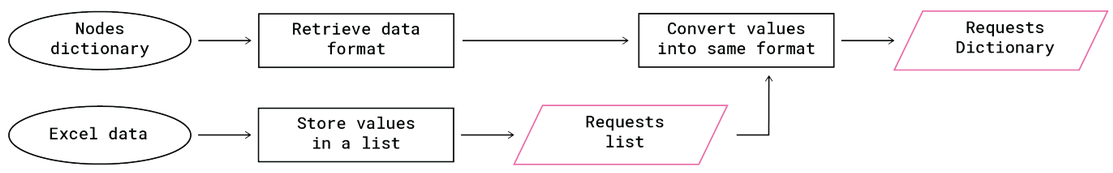


FIGURE 1.2.5: DATA PROCESSING FLOWCHART

GATHERING REQUIREMENTS

1.2 SORTING APPROACH

Design choices

The requests are sorted by degree of priority. This priority depends on the design choices that have been made. The first design choice made, also the one with the highest priority, involves grouping applications within the same districts. This ensures that respondents are located within the minimum walking distance from their residences.

The second design choice is related to the distinction between the different street types. Within the foundation chapter, a distinction is made between three types of streets, namely the main street, medium street and local street. These streets match the three different usage frequencies. Main street - Occational use, medium street - monthly use, and local street - daily use. With this method, all functions are only placed on an associated street. An exception to this occurs at the corners where two types of streets intersect. This is because preference is given to the function with the highest frequency. As a result, the data is prioritized first by daily function, then monthly function, and then occational function.

The third design choice concerns the centering of spaces with a relatively public function. Within the possible building types, the shops are experienced as more open/public, and workshops are experienced as more closed/private.

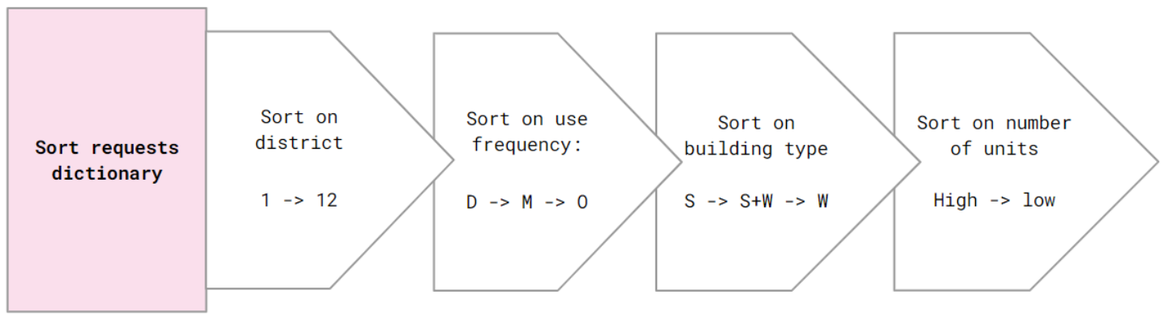


FIGURE 1.2.6: DATA SORTING PRINCIPLE

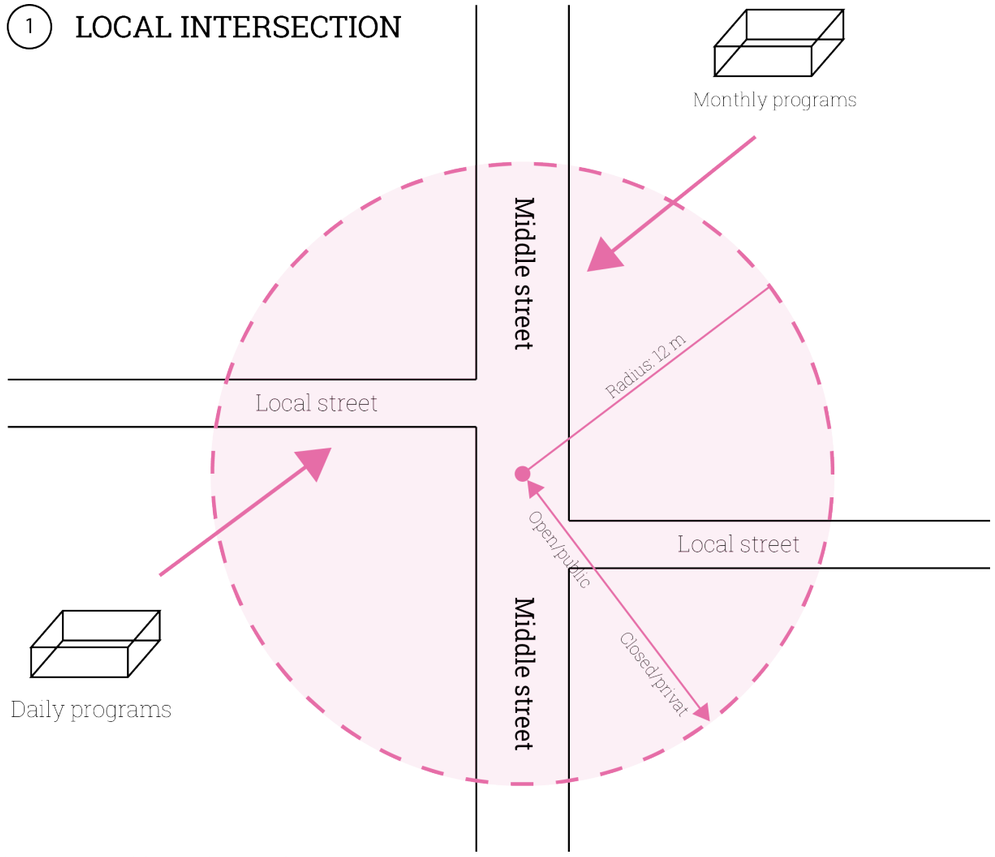


FIGURE 1.2.7: DATA SORTING PRINCIPLE

This means that, calculated from the center of the intersection, the shops should be placed closest. Since the nodes are prioritized by distance from the center, the shops are placed first, followed by the shop + workshop combi, and finally the workshops.

Finally, prioritization is based on the number of modules. It has been decided to place the largest spaces first, to ensure that a suitable place is found.

ALLOCATION OF REQUIREMENTS

1.3 ALLOCATION PRINCIPLE

Placement of the first modules

Since the data of nodes and requests has been formatted in comparable dictionaries, the requests can be matched with available nodes. The first step for placing a request is to find a node that matches the district, the use frequency and has an empty tag. The empty tag indicates that the node is not yet occupied and is not a street node. In addition, it is important that the module is adjacent to a street node, to ensure optimal accessibility for pedestrian traffic. When a node meets these requirements, the first requested module is placed and the dictionary of the node is updated with the values of the request, as shown in figure x.x. If the total amount of requested units is reached, the occupation of the node is official and the algorithm continues with matching the next request in the queue.

Vertical placement of additional modules

While the total amount of requested units is not yet reached, e.g. Ahmed requested 5 units, the next module needs to be placed adjecent to the previous node, to ensure a continuous space for the owner. This means that the adjacent nodes must be checked whether they also meet the previous named requirements. Another requirement that the node must meet is that the node is one layer deeper from the street than the previous node. This leads to a vertical placement approach in which all modules of a specific owner are placed in a line behind each other, perpendicular to the street. This vertical placement continues until the maximum number of layers is reached. The argumentation for the maximum number of layers is explained in the chapter foundation. The maximum number of layers differs per street type: local street three layers, medium street five layers, and main street seven layers.

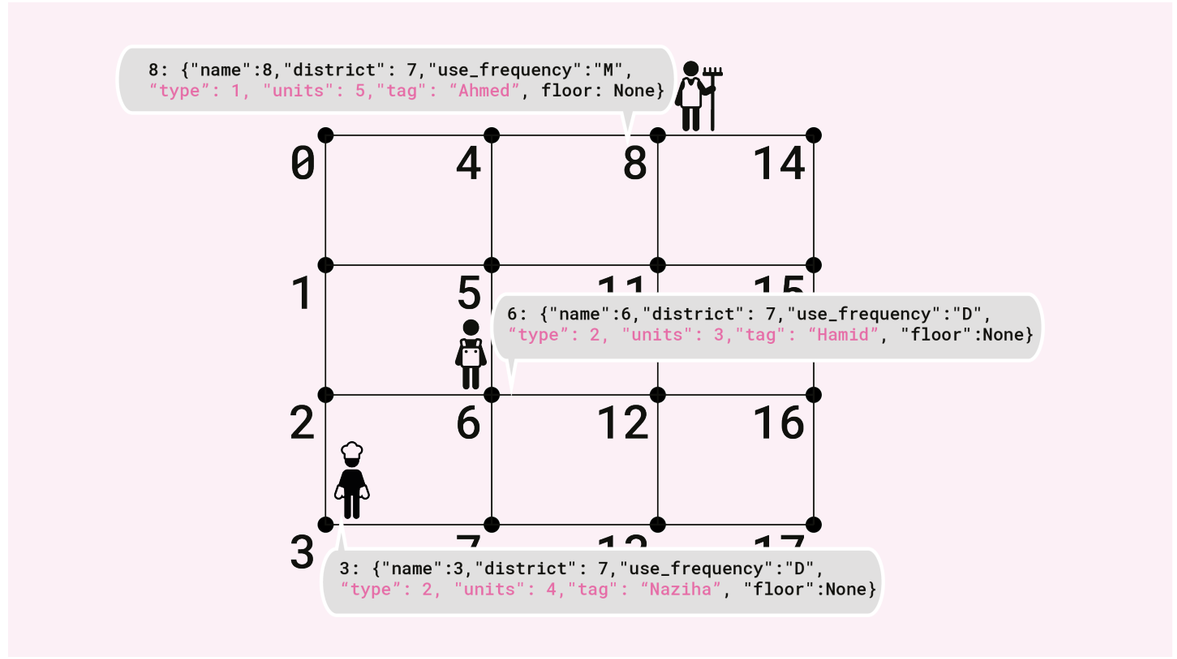


FIGURE 1.3.1: UPDATED NODES DICTIONARY

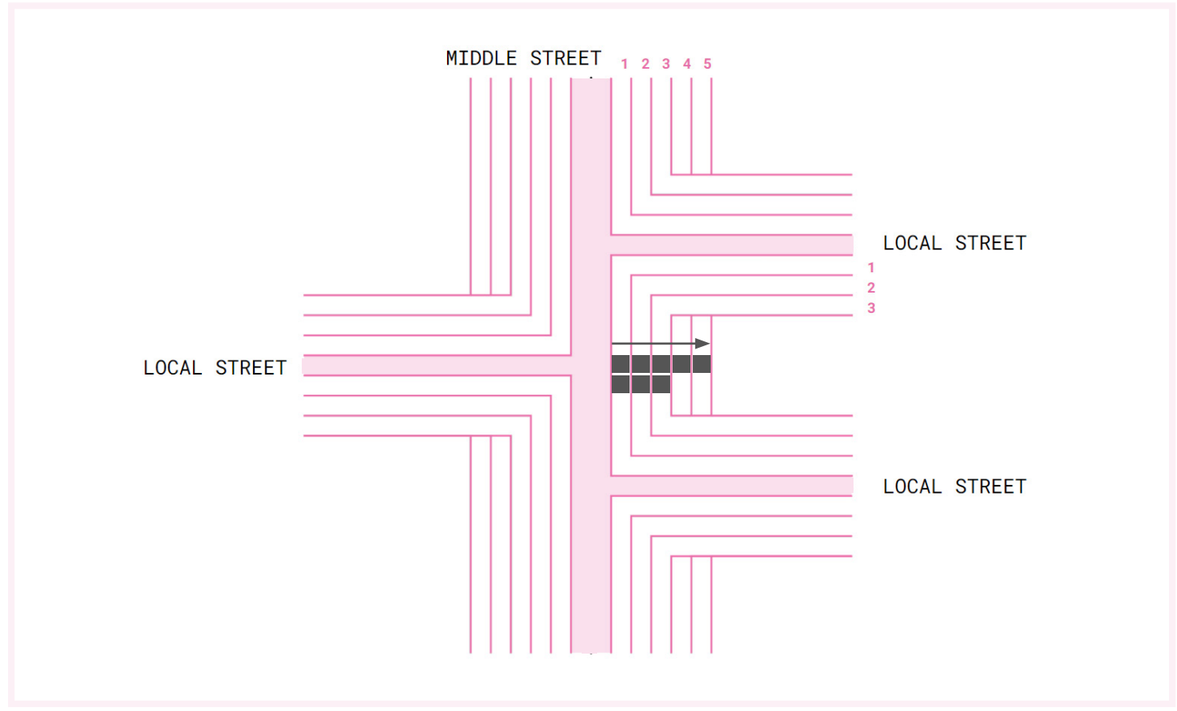


FIGURE 1.3.2: MAXIMUM LAYERS AND VERTICAL PLACEMENT APPROACH

ALLOCATION OF REQUIREMENTS

1.3 ALLOCATION PRINCIPLE

Organic placement

As an alternative placement method the organic placement approach is proposed. The placement of the first requested module happens similar to the vertical placement method. The node should match on district and the use frequency, should be located on layer 1 and should have an empty tag. Once this first module is placed, all neighbors are checked on availability. If a neighbor (preferably with a higher layer value) is available, the next module will be placed. This continues as long as there are requested units left. If it is not possible to place all requested units, all previously occupied nodes will be canceled and deoccupied.

Argumentation chosen approach

For the elaboration of the allocation of the modules, it was decided to use the vertical placement method. The main reason for using this method is the usefulness of the final shape of the continuous spaces. It provides well-arranged spaces that can be easily furnished by the owner. The downside to this method is that the final configuration appears less organic and might leave many blank nodes.

Borders

When the maximum layer is reached, the vertical expansion cannot be continued. In this case, it goes back to the first placed module and finds a new adjacent node. If it is still available, vertical placement will resume. This continues in the same way until the requested number of units is reached. In the event that no more nodes are found that meet the requirements, and not all requested units have been placed yet, the entire placement of that person is canceled, and the occupied nodes are being deoccupied again. After this, the same process is repeated for the next request in the queue. This loop will continue as long as there are requests in the queue. When all requests have been attempted to place, but there is no space left on the ground floor, a second floor will be added. The second floor method will be elaborated later this chapter.

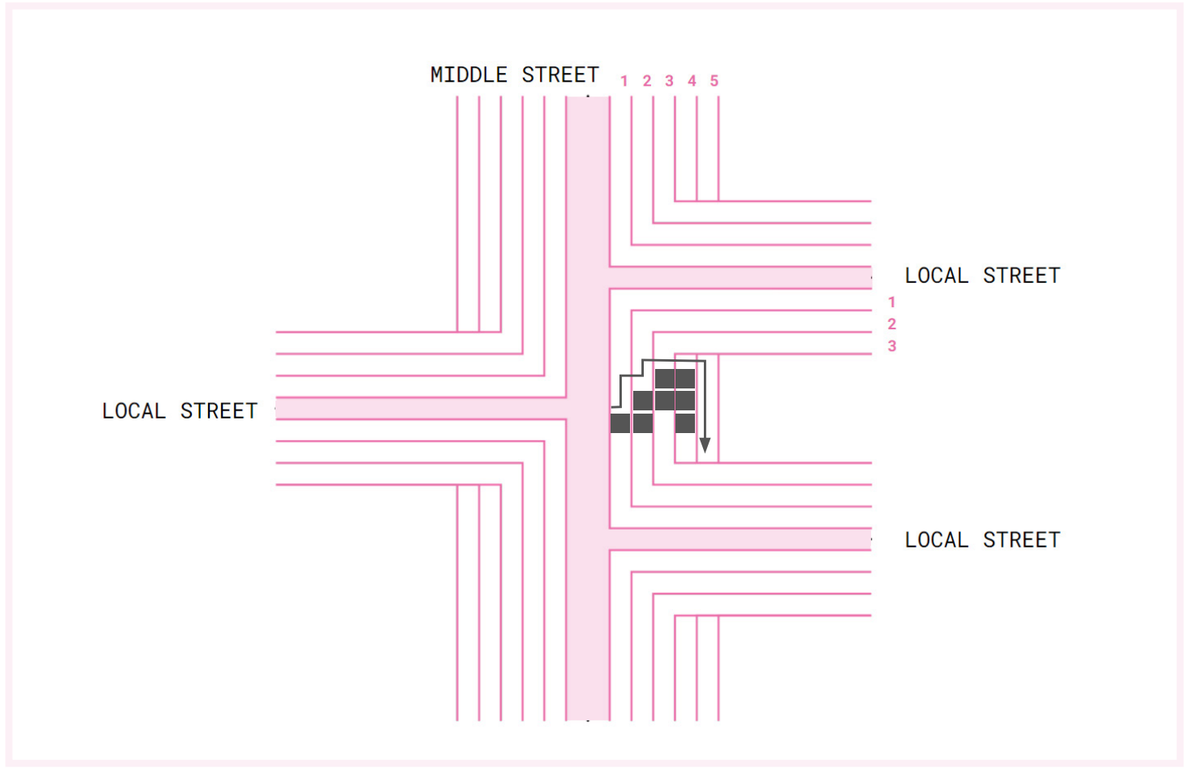


FIGURE 1.3.3: ORGANIC PLACEMENT APPROACH

ALLOCATION OF REQUIREMENTS

1.3 SECOND LEVEL ORGANIZATION

When all requests have been attempted to place, but there is no space left on the ground floor, a second floor will be added. Another requirement for creating a second floor is that at least ten applications have not yet been placed. This ensures that there will be enough supply on the second floor to be worth the circulation.

Stair node allocation

To reach the second floor, self-evidently, stairs are needed. These stairs must be assigned to nodes on the ground floor prior to placing the requests, since these nodes will not be available anymore. To make sure no requests are placed on the occupied stairs node, this nodes tag will become 3. The allocation of the stairs nodes happens according to a set of rules. The first rule concerns the occurrences frequency. Stairs should namely be available every 50 meter, making the maximum walking distance to the closest staircase 25 meter. The stairs should start adjacent to a street node, making it easily visible and accessable for all. The stair module, as will be elaborated in chapter X.X, will occupy all nodes perpendicular to the street, until the maximum layer as shown in image X.X. Since the occupation of stair nodes happens before the placement of the requests, it could happen that there is no need for a second floor, and the stairs lead to nothing. In this case the stair modules are not placed, and the occupied nodes form a alley to the area behind the bazar, mainly residential.

Circulation

The pedestrian circulation on the ground floor happens on the existing streets. However, for the second floor a circulation solution should be developed. It was decided to keep the first layer of the second floor unoccupied and available for pedestrian circulation. In this way we can offer the streets on the ground floor extra high space, turning them into high-quality traffic areas that do not feel cramped. The pedestrian walkways on the second floor are covered and overlook the traffic area on the ground floor. In this way a connection is created between the different building layers enabling a third dimension to the expierence, which contributes to the public character of a traditional bazaar.

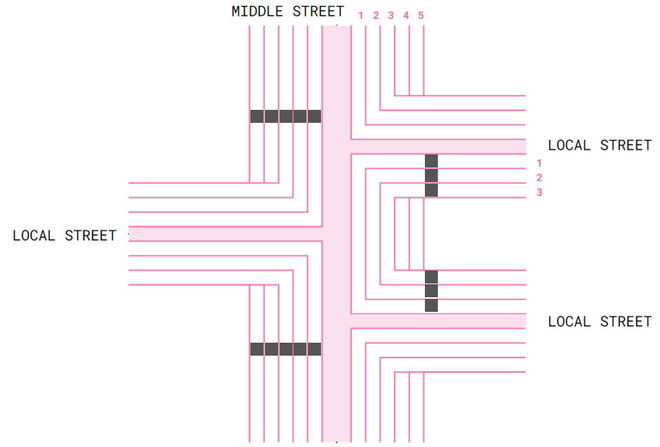


FIGURE 1.3.5: STAIR NODE ALLOCATION

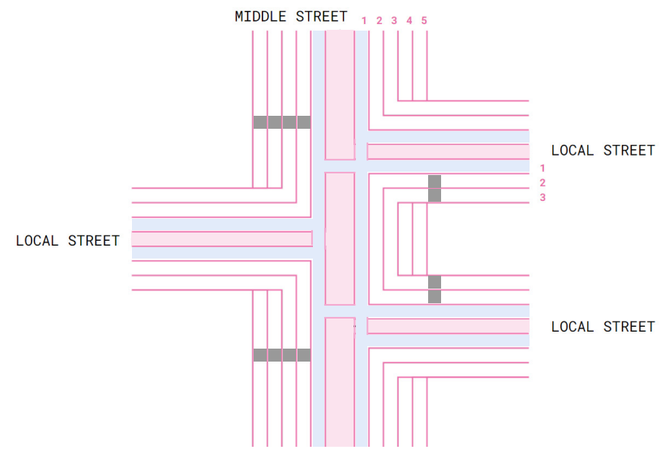


FIGURE 1.3.6: SECOND LEVEL CIRCULATION

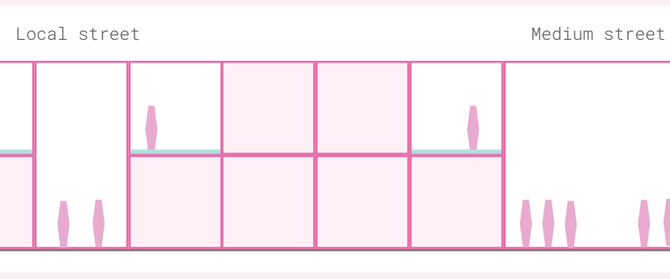


FIGURE 1.3.4: SCHEMATIC SECTION

ALLOCATION OF REQUIREMENTS

1.3 SPATIAL CONFIGURATION

For the final configuration a street joint is chosen where three local streets intersecting a medium street (see figure 1.3.7). The dummy questionnaire requests data was used here to form the spatial configuration. The vertical allocation algorithm has processed all requests and placed them on the field. The result is a spatially high-quality bazaar of contiguous shops and production spaces that perfectly match the wishes of the residents of the Zaatari camp.

Figure 1.3.8 shows the ground floor plan. This clearly shows that the hierarchical division of the shops and workshops has worked out very well. The center of the intersection is only surrounded by shops, which emphasize the public and open character of the bazaar. The further one comes from the center, the more closed and private the spaces become. This emphasizes that the center of the intersection is where the commotion takes place, where pedestrians (customers) are drawn to.

While allocating the list of requests, the algorithm ran into the virtual borders of the field, with unplaced requests left in the list. This means there is need for a second floor, which is shown in a plan in figure 1.3.9. The modules are set back to accommodate foot traffic, shown in blue. It is also clear that the hierarchical division has worked on the second floor just as well, creating an open and public center.

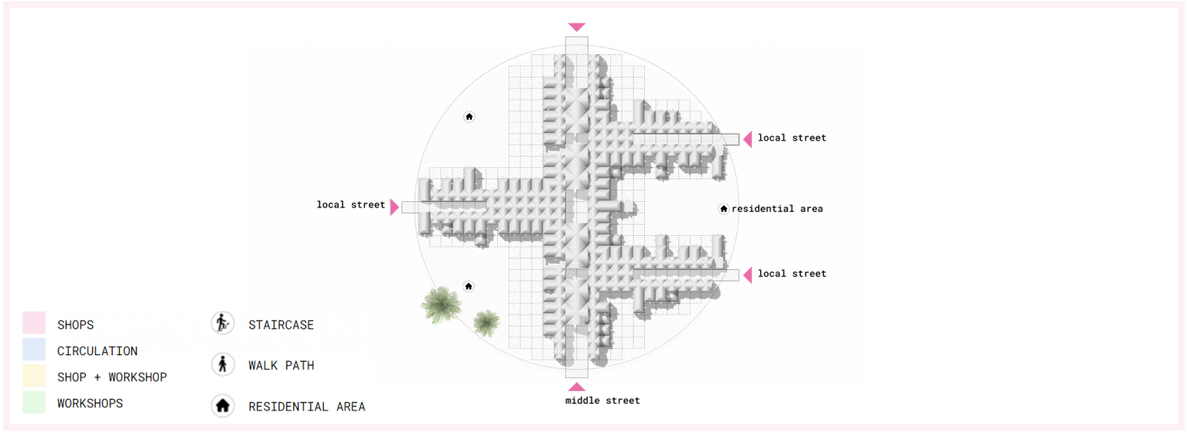


FIGURE 1.3.7: SPATIAL CONFIGURATION

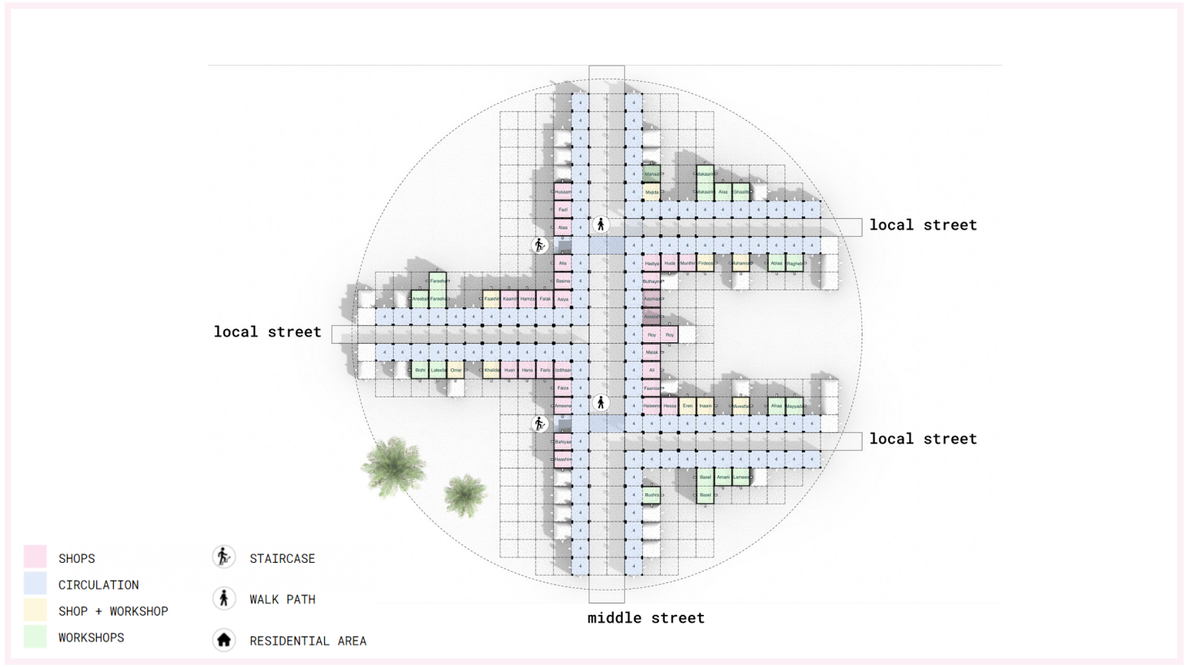


FIGURE 1.3.9: SECOND FLOOR CONFIGURATION

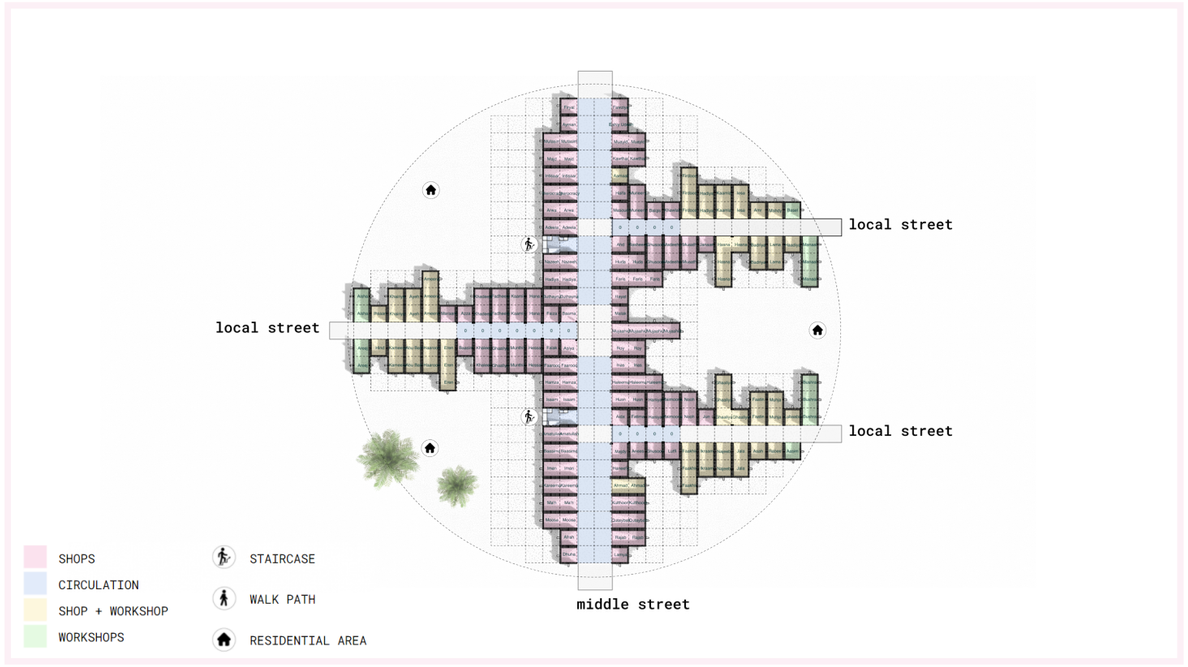


FIGURE 1.3.8: GROUND FLOOR CONFIGURATION

ALLOCATION OF REQUIREMENTS

1.3 SPATIAL CONFIGURATION

Figure 1.3.10 shows the section made over a local street, looking into a medium street. The cross section emphasizes the public character of the center of the intersection, it is more crowded and there is a high density of shops. Towards the side exits of the bazaar, the spaces get more private and less crowded. The section also contains the stair case module, that gives access to the second floor. The blue circulation units on the left and right are also stair cases, giving access to the second floor from the local streets.

Figure 1.3.11 shows the section made over a medium street, looking into two local streets. It is clearly shown that the medium street is very busy and has many adjecent shops. As can be seen the middle street is mainly covered. The canopy opens up where the middle street intersects with another street. This contributes to the way finding aspect of the pedestrian traffic inside the bazaar.

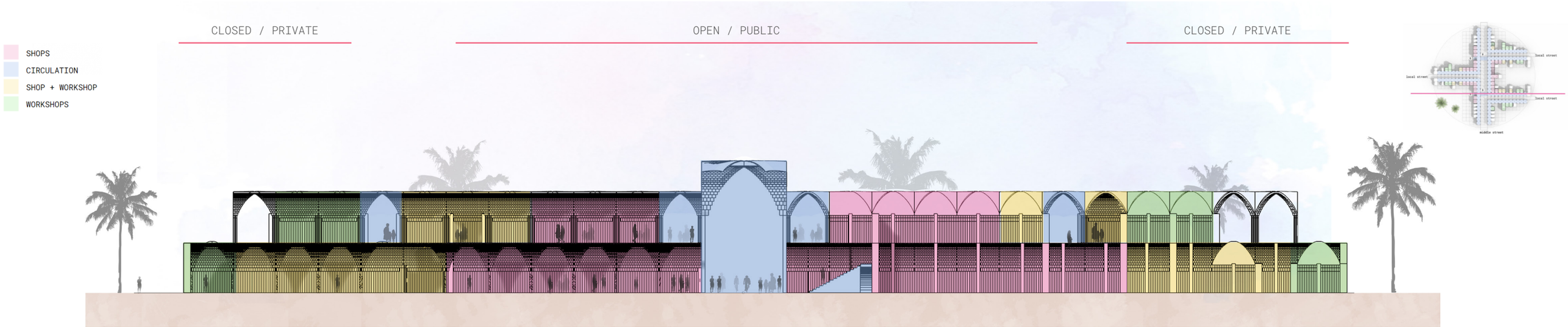


FIGURE 1.3.10: SECOND FLOOR CONFIGURATION

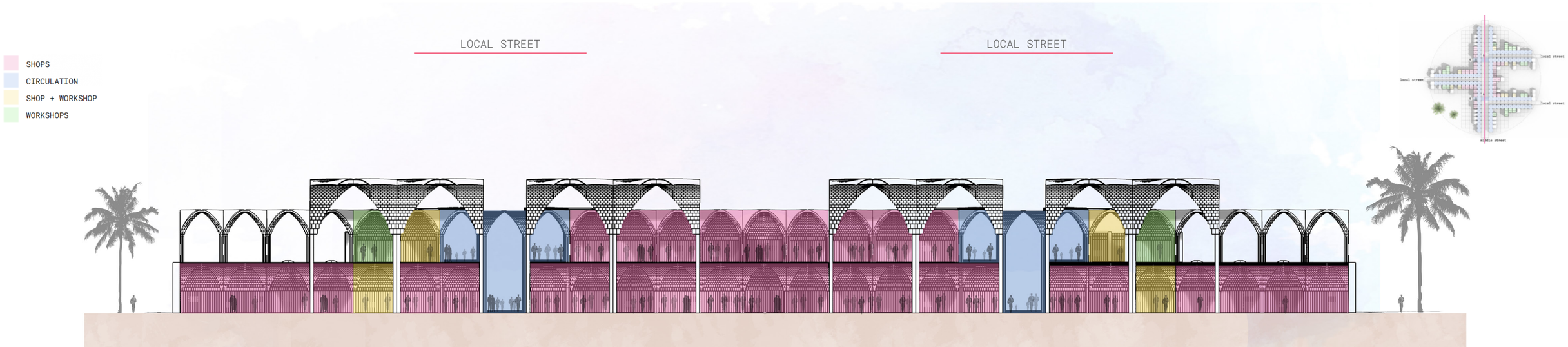


FIGURE 1.3.11: SECOND FLOOR CONFIGURATION

ADJACENCIES & MODULE TYPES

1.4 METHODOLOGY

Main aim of this part of the whole work flow is to assess the adjacencies between the nodes and assign adequate module types to the nodes. Overall, the general work flow is based on adding data to the nested dictionary and transferring it to the further steps. For Adjacencies & Module Types stage, data is received from Allocation of Requirements stage and Base Grid stage in the form of a nested nodes dictionary and adjacency list(edge list) respectively. The nodes dictionary holds the necessary information such as the "tag" attribute which can be the owner's name, out of the boundary ("x") or empty node (None). Second important attribute in this stage is "floor", in other words whether there is a module above that node(1) or node above is empty(0). Adjacency list comes in tuples (node1,node2)which holds the information of which node is connected to which nodes.

Nodes Dictionary:
{0: {'tag': 1,'units': None,'district': 7,'name': 0,'use_frequency': 'M','type': None,'floor': 0},1: {'tag': "John",'units': 2,'district': 7,'name': 0,'use_frequency': 'D','type': 1,'floor': 1},...}

Adjacency List: [(0, 1),(0, 4),(1, 5),(1, 2),...]
Each available node has 4 adjacencies/edges and consequently 4 neighbors. Not available nodes, nodes out of the boundary, may have fewer neighbor however, since modules will not be assigned to those, having less neighbors is not a disadvantage.

To give a short summary of the process,first step after importing the necessary dictionary and list is to draw a graph utilizing Networkx and extracting an adjacency dictionary. After this step start and end nodes are compared and a new attribute,"connection" is implemented into the nested dictionary for each node. Module types and their variations are named and another attribute,"module" is added to the nodes dictionary.

Adjacency Dictionary: {0:[1,4,_,_],1:[0,2,5,_],5:[1,4,6,11],...}

Updated Nodes Dictionary: {...,1: {'tag': "John",'units': 2,'district': 7,'name': 0,'use_frequency': 'D','type': 1,'floor': 1,'connection':[0,1,0,1],'module':30.1},...}

The last step is to transfer the data to the next stage: Orientation.

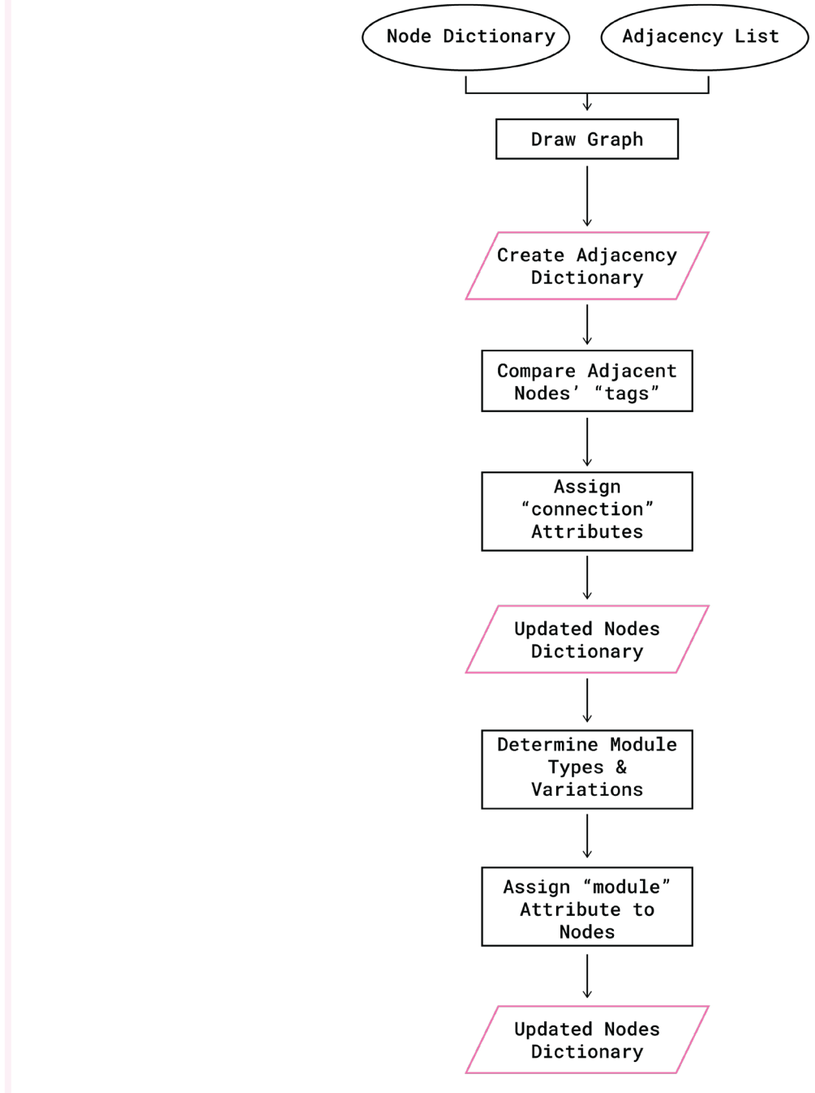


FIGURE 1.4.1 | FLOWCHART

ADJACENCIES & MODULE TYPES

1.4 ASSESSING ADJACENCIES | BASE GRAPH

First step is to import the data(.txt) from Github which is updated during the allocation and generation of base grid. After having the nested nodes dictionary and adjacency list, Networkx is used to draw the graph. We chose to work with graphs as a medium because with the Networkx library examining the adjacencies, finding neighboring nodes and storing data in a node is advantageous. The graph composed of the given nodes dictionary and adjacency list is drawn without any positions but only with the node numbers and their connections. It should be mentioned that each node represents a cell it is situated in. Adjacency dictionary is created by using the graph.

Nodes Dictionary:
{0: {'tag': 1,'units': None,'district': 7,'name': 0,'use_frequency': 'M','type': None,'floor': 0},1: {'tag': "John",'units': 2,'district': 7,'name': 0,'use_frequency': 'D','type': 1,'floor': 1},...}

Adjacency List: [(0, 1),(0, 4),(1, 5),(1, 2),...]

Adjacency Dictionary: {0:[1,4,_,_],1:[0,2,5,_],5:[1,4,6,11],...}

Nodes Dictionary holds the information such as "name","user frequency","units","tag", "floor" which are explained in the 1.3 Allocation of Requirements Section. During this stage the most important attributes are "tag" and "floor".

"tag" attribute: 0 (int format) -> represents local street
1 (int format) -> represents middle street
2 (int format) -> represents main street
3 (int format) -> represents staircase nodes
4 (int format) -> represents walk-paths on upper levels
"owner name" (str format) -> shop/workshop owner
"x" (str format) -> node out of the boundary
None -> unoccupied/empty/available node

"floor" attribute: 0 (int format) -> node above the current node is empty
1 (int format) -> node above the current node is occupied

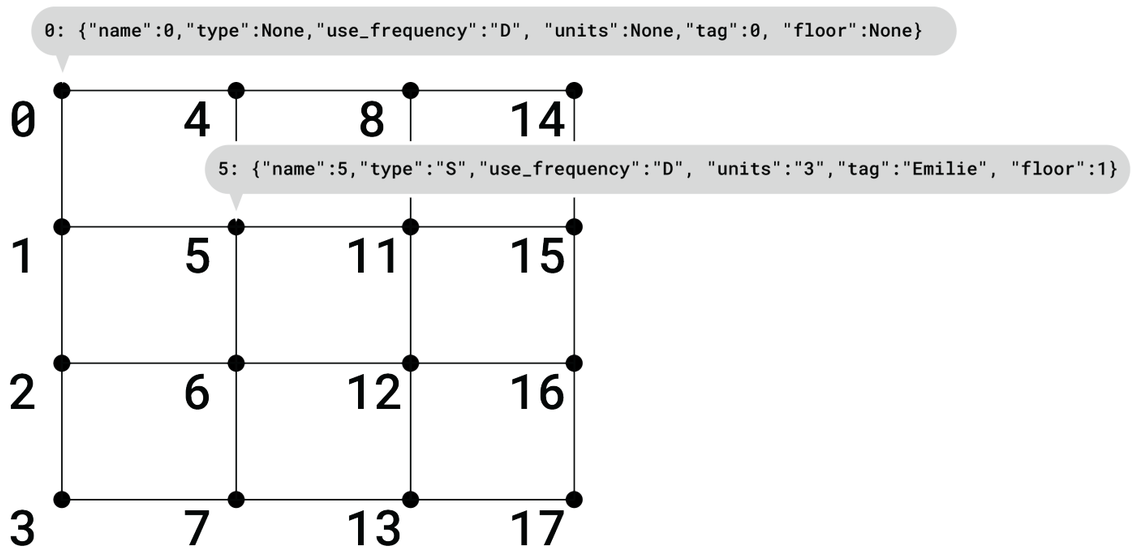


FIGURE 1.4.2 | BASE GRAPH

ADJACENCIES & MODULE TYPES

1.4 ASSESSING ADJACENCIES

After setting up the necessary data types, adjacencies will be examined by comparing the “tag” attributes. As we are only checking the adjacencies of occupied nodes to assign a module, street nodes, nodes with the “tag” “x” or None are negligible. Each node has a tag and represents a cell as illustrated in figure 1.4.4.

“tag” attribute: 0 (int format) -> represents local street
1 (int format) -> represents middle street
2 (int format) -> represents main street
3 (int format) -> represents staircase nodes
4 (int format) -> represents walk-paths on upper levels
“owner name” (str format) -> shop/workshop owner
“x” (str format) -> node out of the boundary
None -> unoccupied/empty/available node

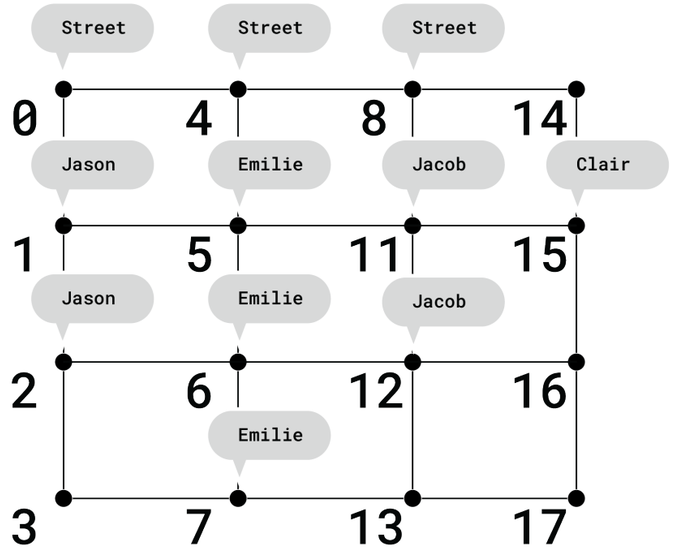


FIGURE 1.4.3 | GRAPH WITH “TAG” ATTRIBUTES

0 Street	4 Street	8 Street	14 Street
1 Jason	5 Emilie	11 Jacob	15 Clair
2 Jason	6 Emilie	12 Jacob	16 None
3 X	7 Emilie	13 None	17 None

FIGURE 1.4.4 | CELLS WITH “TAG” ATTRIBUTES

ADJACENCIES & MODULE TYPES

1.4 ASSESSING ADJACENCIES | COMPARISON

The logic behind comparing “tag” attributes is that if two nodes/cells have the same owner, data stored in “tag” attribute, then these nodes should belong to the same building and they should be accessible within.
In this step adjacency dictionary is used to compare the key (current node) to each value (neighbors). If the “tag” attributes are the same owner name or if one of the nodes is a street node then the connection between the compared nodes should be open(0), if the compared nodes belong to different owners the connection should be closed(1), or if one of the compared nodes’ “tag” is None or “x” then the connection should be closed but a buttress will be placed for structural reasons (2) as illustrated on the graph in figure 1.4.5 and with cells in figure 1.4.6. These comparisons are used to assign a new attribute to the nodes dictionary: “connection”

Adjacency Dictionary:

```
{1: [0, 5,_,_], 2: [1,3,6,_], 3: [2, 7], 4: [0,5,8,_],
 5: [1,4,6,11],...}
```

```
5: [1,4,6,11]
start node (key)    neighbouring nodes (value)
```

“connection” attribute possible representatives:

```
0 -> open connection
1 -> closed connection
2 -> closed connection&buttress
```

“connection”: [1,0,0,1]

closed, open, closed, open

Updated Nodes Dictionary:

```
{..., 5: {‘tag’: “Emilie”, ‘units’: 3, ‘district’: 7,
‘name’: 0, ‘use_frequency’: ‘M’, ‘type’: None, ‘floor’: 0,
‘connection’: [1,0,0,1]},...}
```

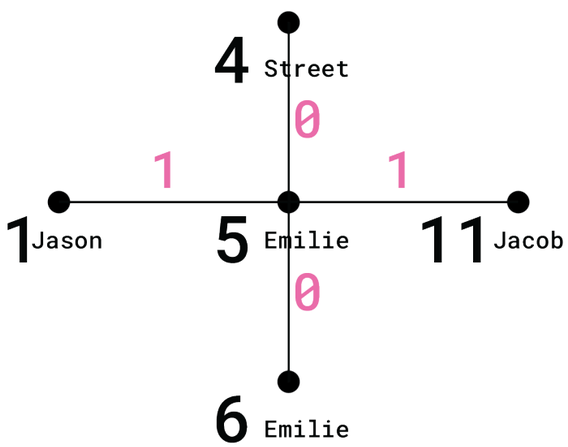


FIGURE 1.4.5 | PARTIAL GRAPH EXAMPLE OF COMPARISON

FIGURE 1.4.6 | COMPARISON RULES

ADJACENCIES & MODULE TYPES

1.4 ASSESSING ADJACENCIES | COMPARISON

After comparing each node to its neighbors we acquire a graph similar to figure 1.4.7. If this graph is drawn with a topological point of view with only the nodes and the existing connections, it can be visualized as in figure 1.4.8. Pink lines in the same illustration represents the actual connections so if two nodes are connected with a pink line nodes, access between those nodes is available. This data will be used to determine the module types in later steps.

“connection” attribute possible representatives:

0 -> open connection
1 -> closed connection
2 -> closed connection&buttress

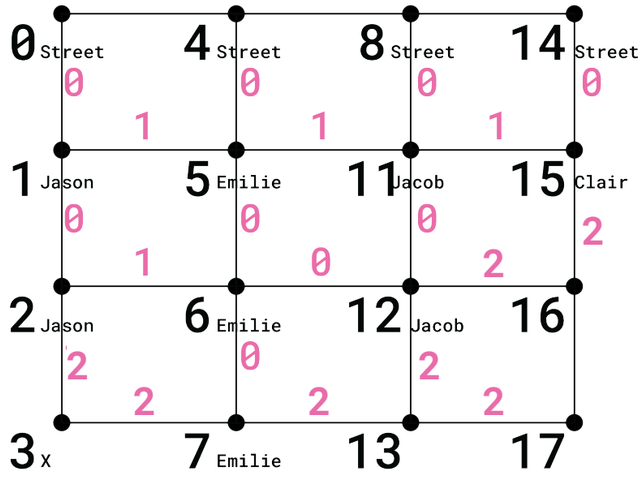


FIGURE 1.4.7 | GRAPH WITH “CONNECTION” ATTRIBUTE

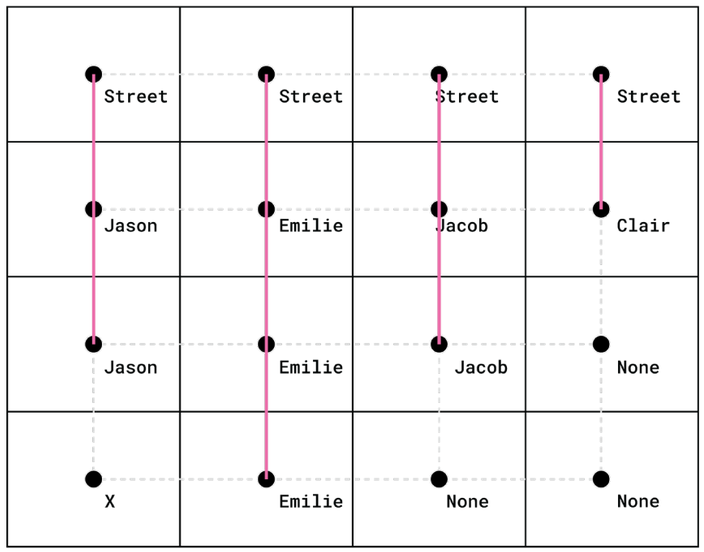


FIGURE 1.4.8 | “CONNECTION” VISUALISATION ON CELLS MAP

ADJACENCIES & MODULE TYPES

1.4 ASSIGNING MODULE TYPES | DEFINING TYPES

“connection” attribute possible representatives:

0 -> open connection

1 -> closed connection

2 -> closed connection&buttress

“connection”: [1,0,0,1]

With the given options of “connection” representatives, there are a number of combinations of 4 numbers, since there are 4 neighbors. All these combinations are given in figure 1.4.9. These combinations include nodes with all open connections, only one closed connection, two closed connections adjacent to each other,three closed connections, two closed connections but on the opposite sides which are illustrated on the top row. Variations below are based on the number of closed and buttressed connections. Theoretically speaking each closed connection(1) can be a buttressed connection(2), therefore any possibility is taken into account.

updated node dictionary:

{..., 5: {‘tag’: “Emilie”, ‘units’: 3,
‘district’: 7, ‘name’: 0, ‘use_frequency’: ‘M’,
‘type’: None, ‘floor’: 0, ‘connection’: [1,0,0,1]},...}

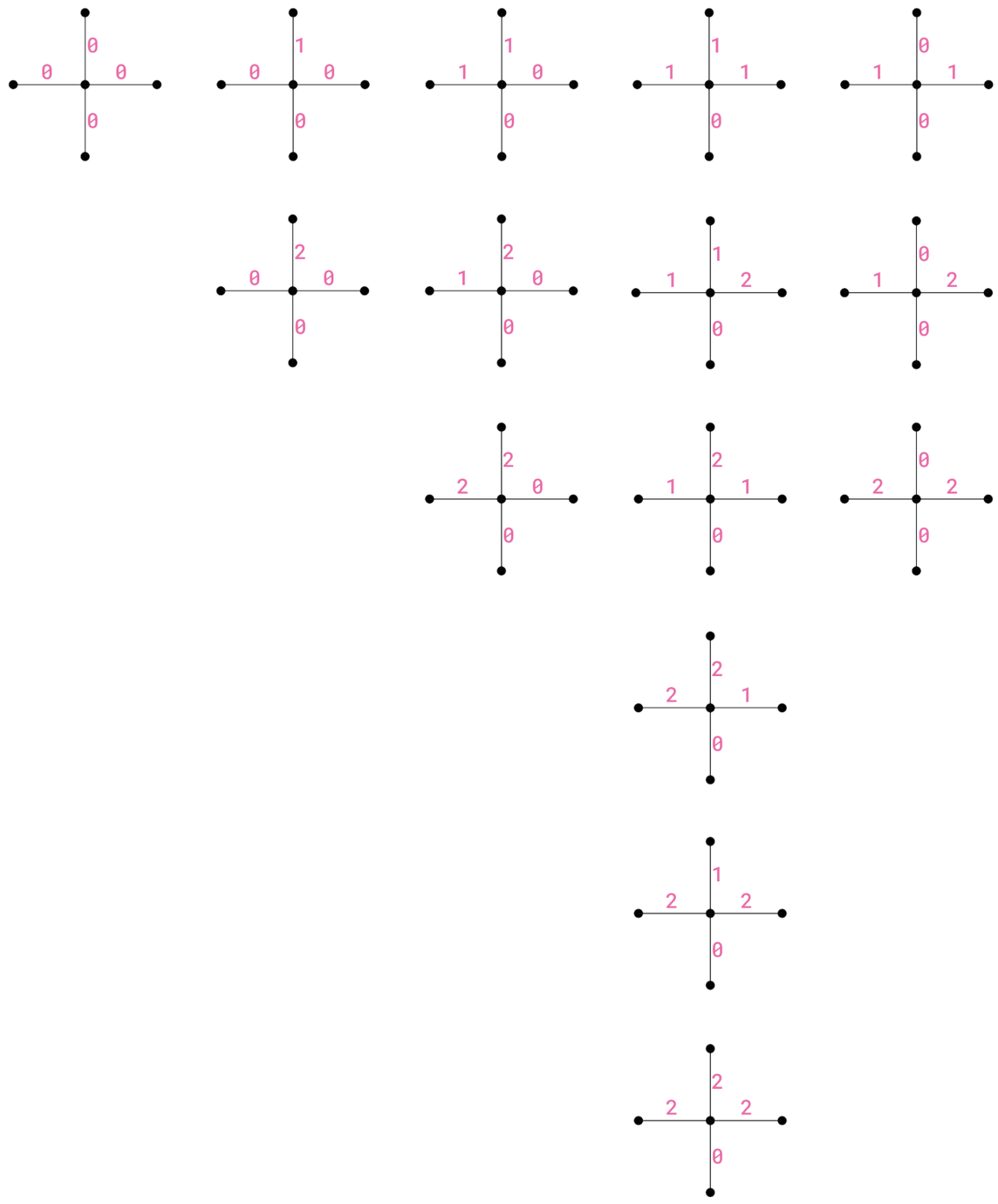


FIGURE 1.4.9 | POSSIBLE “CONNECTION” CONDITION COMBINATIONS

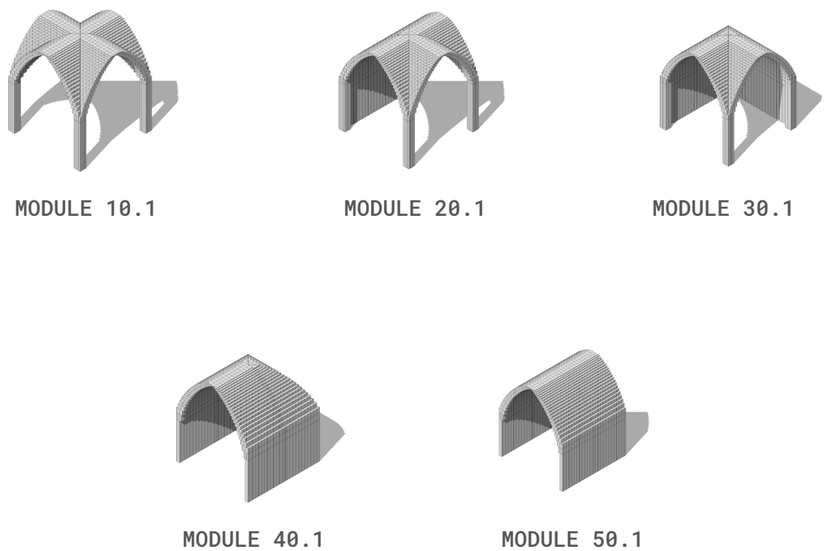
ADJACENCIES & MODULE TYPES

1.4 ASSIGNING MODULE TYPES | UNFILLED MODULE

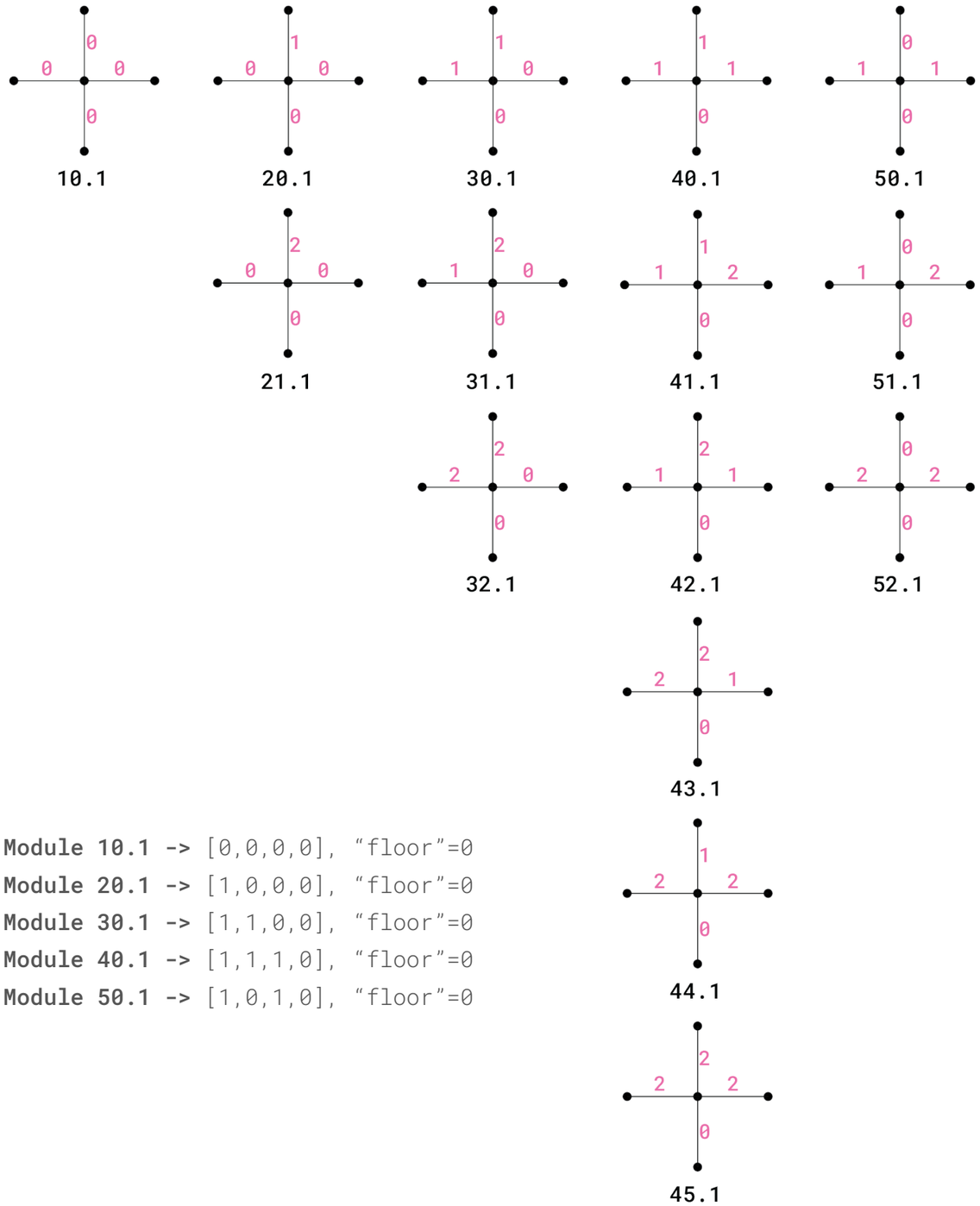
A module name is assigned to each main combination(10.1, 20.1, 30.1, 40.1, 50.1). These module types are visualized architecturally in figure 1.4.10. The number after the decimal point, whether it is 1 or 2, represents whether that is a stackable module or not. If there is no module above then the ceiling is unfilled, consequently it is not possible to walk on top.

Module 10.1 -> all sides open [0,0,0,0]
Module 20.1 -> one side closed [1,0,0,0]
Module 30.1 -> two sides closed (adjacent sides) [1,1,0,0]
Module 40.1 -> three sides closed [1,1,1,0]
Module 50.1 -> two sides closed (opposite sides) [1,0,1,0]

Rest of the variations are named with the same tens decimal number based on the main module name as illustrated in the figure 1.4.11. Naming of the modules are done with float numbers for the ease of application on our side however different integers could be given to represent each module type. Working with integers would be preferable due to computational reasons.



1.4.10 MODULE TYPES ARCHITECTURAL VISUALIZATION (UNFILLED)



Module 10.1 -> [0,0,0,0], “floor”=0
Module 20.1 -> [1,0,0,0], “floor”=0
Module 30.1 -> [1,1,0,0], “floor”=0
Module 40.1 -> [1,1,1,0], “floor”=0
Module 50.1 -> [1,0,1,0], “floor”=0

1.4.11 MODULE TYPES TOPOLOGICAL VISUALIZATION (UNFILLED)

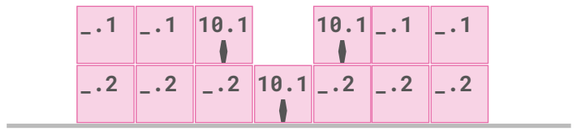
ADJACENCIES & MODULE TYPES

1.4 ASSIGNING MODULE TYPES | FILLED MODULE

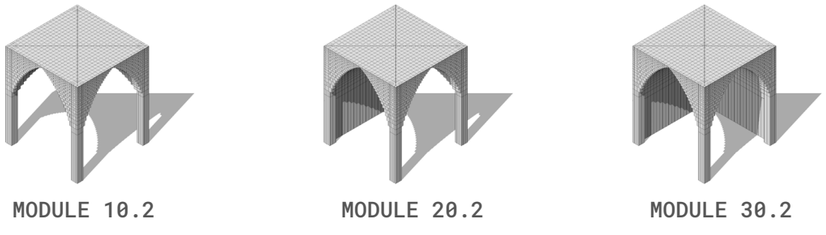
Same connection combinations are used for the filled modules which are designed to have another module on top (if “floor” attribute is 1) and/or be walkable on the upper story. If there is a module above then the ceiling is filled, consequently it is possible to walk on top and the number after the decimal point is 2.

“floor” attribute : 0 (int format) -> node above the current node is empty
1 (int format) -> node above the current node is occupied

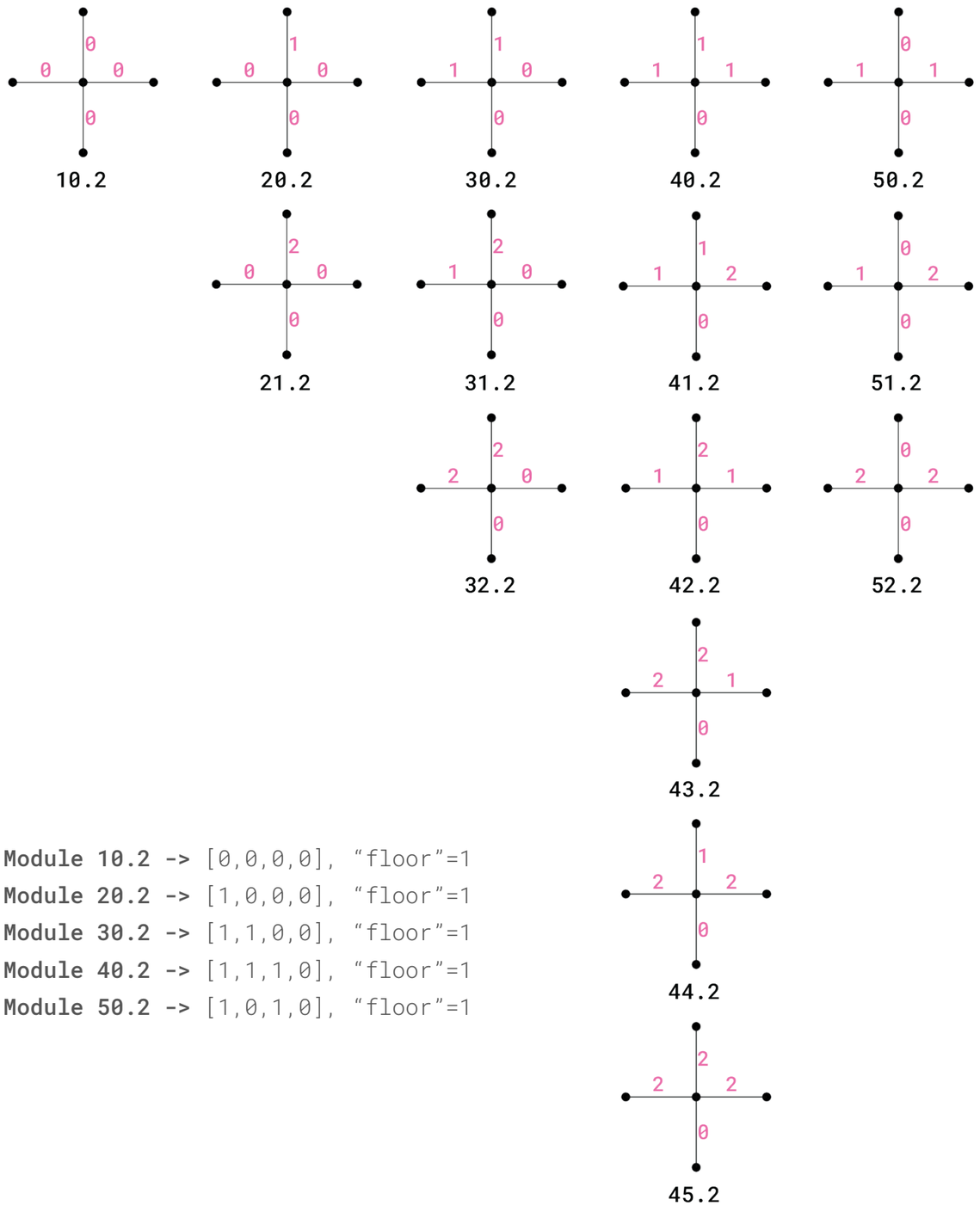
On the second story, since there is a 1 unit long setback which will allow circulation, modules below are filled if there is a shop above. To determine which nodes will be walkable we looked into the walk-path nodes (“tag”=4) If that node has an occupied neighbor then the node above that node should be assigned a module which is walkable. Module type 10.1 is assigned to circulation modules since it allows for access to/from all directions as shown in figure 1.4.12.



1.4.12 MODULE TYPES DIAGRAM IN SECTION



1.4.13 MODULE TYPES ARCHITECTURAL VISUALIZATION (FILLED)



Module 10.2 -> [0,0,0,0], “floor”=1
Module 20.2 -> [1,0,0,0], “floor”=1
Module 30.2 -> [1,1,0,0], “floor”=1
Module 40.2 -> [1,1,1,0], “floor”=1
Module 50.2 -> [1,0,1,0], “floor”=1

1.4.14 MODULE TYPES TOPOLOGICAL VISUALIZATION (FILLED)

ADJACENCIES & MODULE TYPES

1.4 ASSIGNING MODULE TYPES | EXCEPTIONS

Some module types have the same number of closed and open connections including module 30.1 and 50.1. And the list corresponding to the “connection” attributes are not order. Therefore a module type can be represented by many different combinations of the same numbers. For an example, module type 20 has one closed edge(1) and three open edges(0). However these numbers may appear in different orders. Therefore we checked if a node’s “connection” attributes are within the list of possibilities which can be created by combinations of related 4 numbers.
Module 20.1 -> [1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]

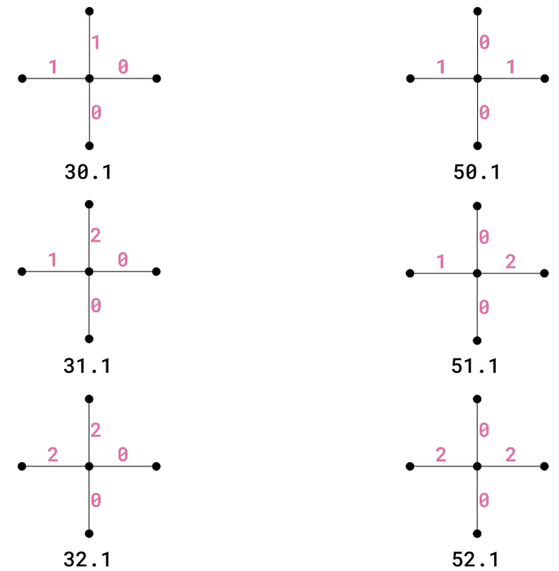
Therefore before assigning the generic modules exceptions are made. In order to choose between module types 30 and 50, first a new dictionary composed of each node as a key, and only closed edges (1 or 2) as values is formed. By forming this dictionary, unnecessary information is disregarded. Afterwards, neighbors of the neighboring node of the current node are find. If two neighboring nodes have the same neighbor excluding the current node of assessment, then the closed edges are adjacent to each other (module family 30); otherwise the closed edges are on opposite sides (module family 50) as illustrated in figure 1.4.16.

Closed Edge Dictionary:

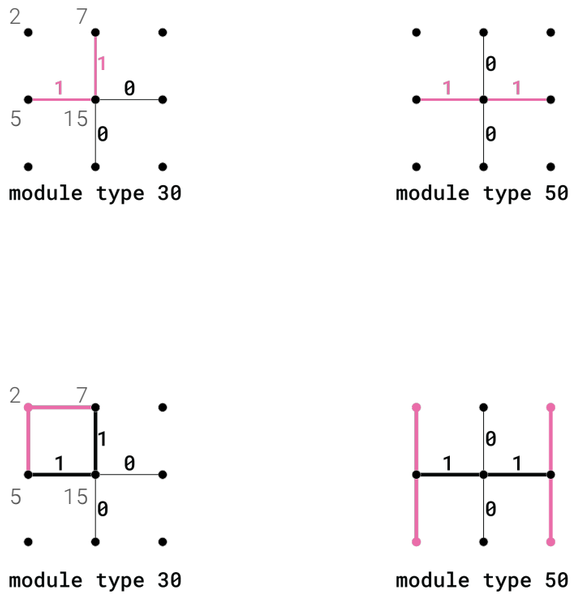
{..., 15: [7,5], 5:[15,2,_,_], 7: [15,2,_,_], 2: [5,7,_,_], ...}

In the example dictionary above, it is indicated that node 15(current node) has two closed connection neighbors: node 7(current node) and node 5 (current node). Node 7 (current node) is connected to node 15 (current node) and node 2(neighbor of neighbor). Node 5 is connected to node 15 (current node) and node 2 (neighbor of neighbor) as well. Therefore since the neighbor of the neighbor of the current nodes are the same node then module 30 should be assigned to node 15.

Module 30.1 -> two sides closed (adjacent sides) [1,1,0,0]
Module 31.1 -> two sides closed (adjacent sides) [1,2,0,0]
Module 32.1 -> two sides closed (adjacent sides) [2,2,0,0]
Module 50.1 -> two sides closed (opposite sides) [1,0,1,0]
Module 51.1 -> two sides closed (opposite sides) [2,0,1,0]
Module 52.1 -> two sides closed (opposite sides) [2,0,1,0]



1.4.15 MODULE TYPES 30 & 50



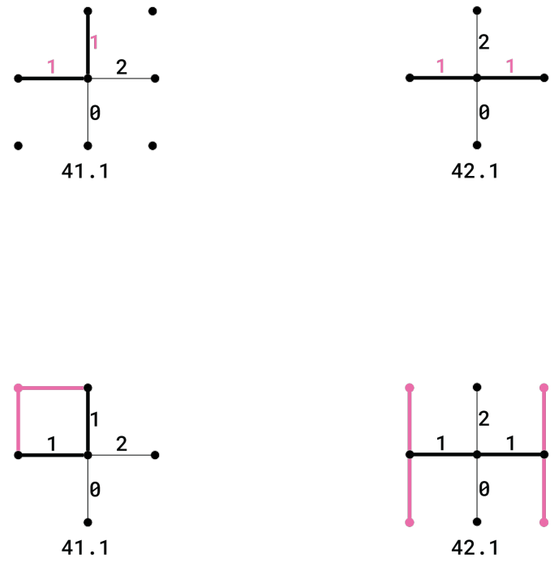
1.4.16 MODULE TYPES 30 & 50 EXCEPTIONS

ADJACENCIES & MODULE TYPES

1.4 ASSIGNING MODULE TYPES | EXCEPTIONS

Similarly module types 41 and 42 have the same number of closed edges(1) therefore whether these closed connections are next to each other or on the opposite sides should be checked. Similar approach to module type 30 and 50 exceptions, are followed. Nodes with “connection” attribute [0,1,1,2] or any combination of these 4 numbers are gathered into a list. Afterwards a new dictionary holding only these nodes as keys and the closed edges(1) as values are formed. Last step is to check whether neighbors of the current node, which we are assigning a module to, has more than two common neighbors. If there are 2 common neighbors of neighbors of current node than module type 41 is assigned to the current node, otherwise type 42 is assigned. We are looking for more than 2 common neighbors since neighbor’s neighbor can be the current node as well as another node.

Module 41.1 -> two sides closed (adjacent sides) [1,1,2,0]
Module 42.1 -> two sides closed (opposite sides) [1,2,1,0]

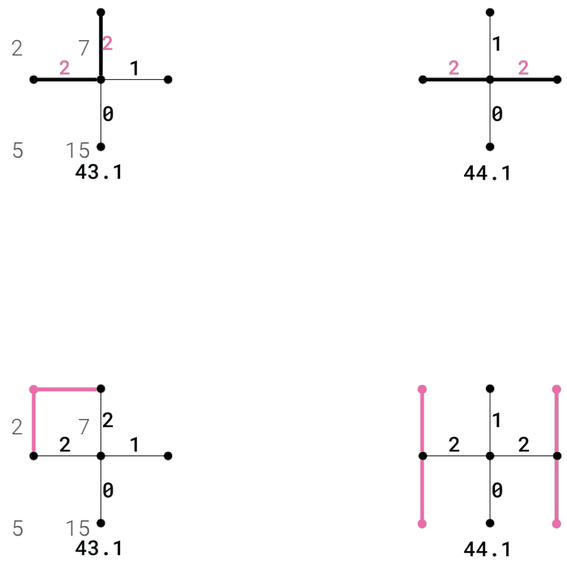


1.4.17 MODULE TYPES 41 & 42 EXCEPTIONS

Same method is followed to distinguish between module type 43 and 44 as they both have the same combination of 4 numbers([2,2,1,0]. A new dictionary of reinforced/buttressed edges is created only for the nodes which have “connection”=[2,1,11,0] or any other combination of these 4 numbers. And whether the nodes connected by the buttressed edges have the same neighbors is checked as illustrated in figure 1.4.17.

Reinforced Edge Dictionary_43_44:
{..., 15: [7,5], 5:[15,2,_,_], 7: [15,2,_,_], 2: [5,7,_,_], ...}

To explain further, in the exemplary edge dictionary above, we are trying to assign a module to node 15(current node). Current node is connected to other nodes with two buttressed(2), one open(0), one closed(1) edges. Buttressed edges connect nodes 7 and 5 to the current node. Nodes 7 and 5 have node 15 as a neighbor,1st common neighbor, as well as node 2. Therefore module type 43 is assigned to node 15.



1.4.18 MODULE TYPES 43 & 44 EXCEPTIONS

ADJACENCIES & MODULE TYPES

1.4 OUTPUT

After following the aforementioned steps, all the shop/workshop/shop&workshop nodes are assigned a module type. In figure 1.4.19, closed connections,walls, are illustrated in pink.

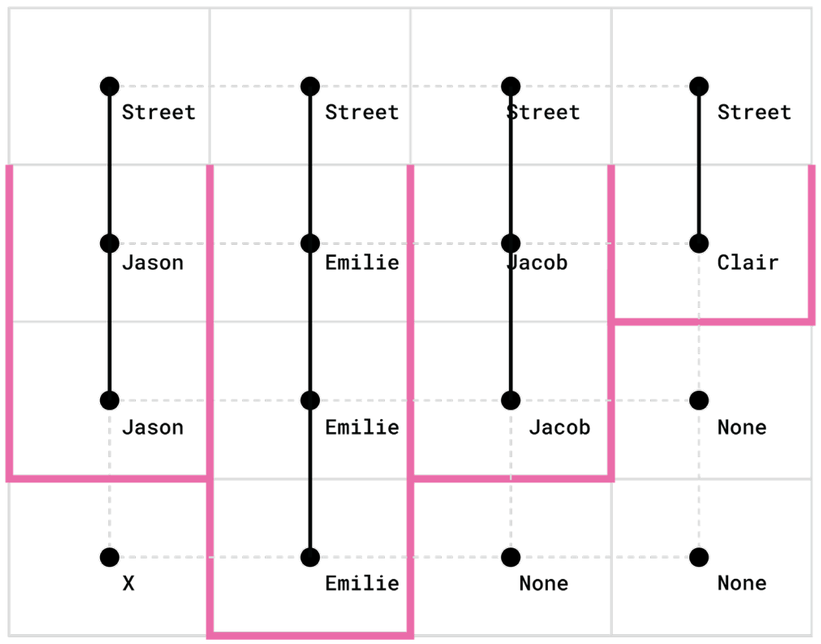
To summarize the conditions for having open or closed connections:

for start node and end node:
- **if start node and end node have the same “tag”(owner)**
- **if one of these nodes is a street node(“tag”=0,1,2)**
  - **->> then the connection is open(0)**
- **if one of the nodes is out of the boundary “tag”=”x”**
- **if one of the nodes is an empty node “tag”=None**
  - **->> then the connection is closed and buttressed (2)**
- **if start and end node have different “tag”**
  - **->> then the connection is closed(1)**

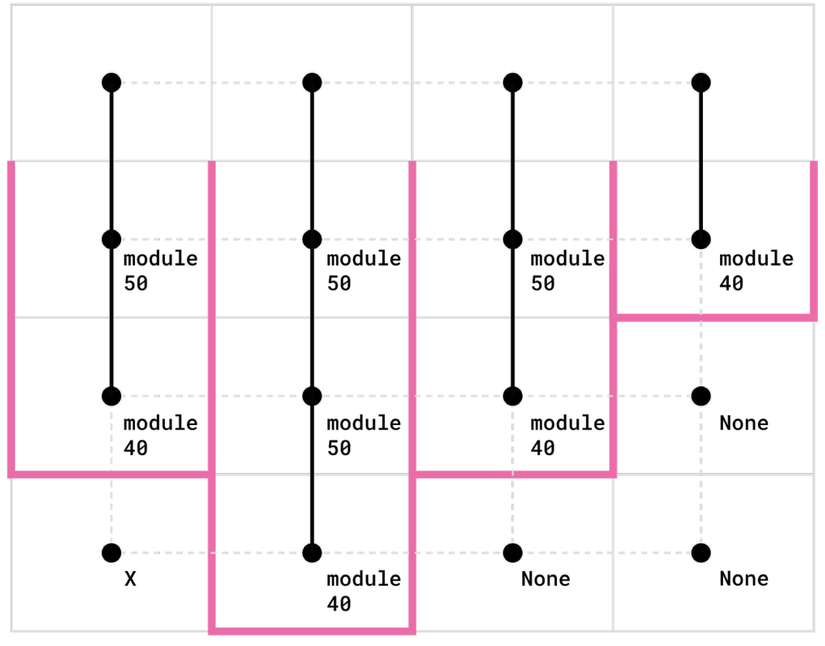
After connections are all determined corresponding module types are assigned to the shop/workshop/shop&workshop nodes as visualized in figure 1.4.20.

final node dictionary:

{..., 5: {‘tag’: “Emilie”, ‘units’: 3, ‘district’: 7, ‘name’: 0, ‘use_frequency’: ‘M’, ‘type’: None, ‘floor’: 0, ‘connection’: [1,0,0,1], **‘module’:50.2**},...}



1.4.19 CELL MAP WITH CLOSURE (CLOSED WALLS SHOWN IN PINK)



1.4.20 CELL MAP WITH ASSIGNED MODULE TYPES (CLOSED WALLS SHOWN IN PINK)

ADJACENCIES & MODULE TYPES

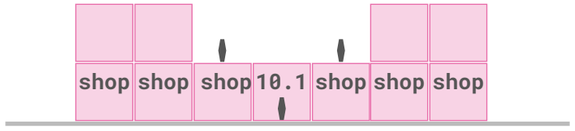
1.4 COVERING THE STREETS

Last step of this section is to assign modules to cover some portion of the streets. We decided to cover the streets starting from one specific node, the cornerstone as we like to call it. The cornerstone is defined by two shops on opposite sides of the streets facing each other. Since during the Allocation of Requirements, shops are prioritized and placed first, shops accumulate around the intersection as shown in figure 1.4.22. Therefore after one point there is no other shop facing shop condition possible, this point is the cornerstone. To locate the cornerstone, there are two conditions checked:

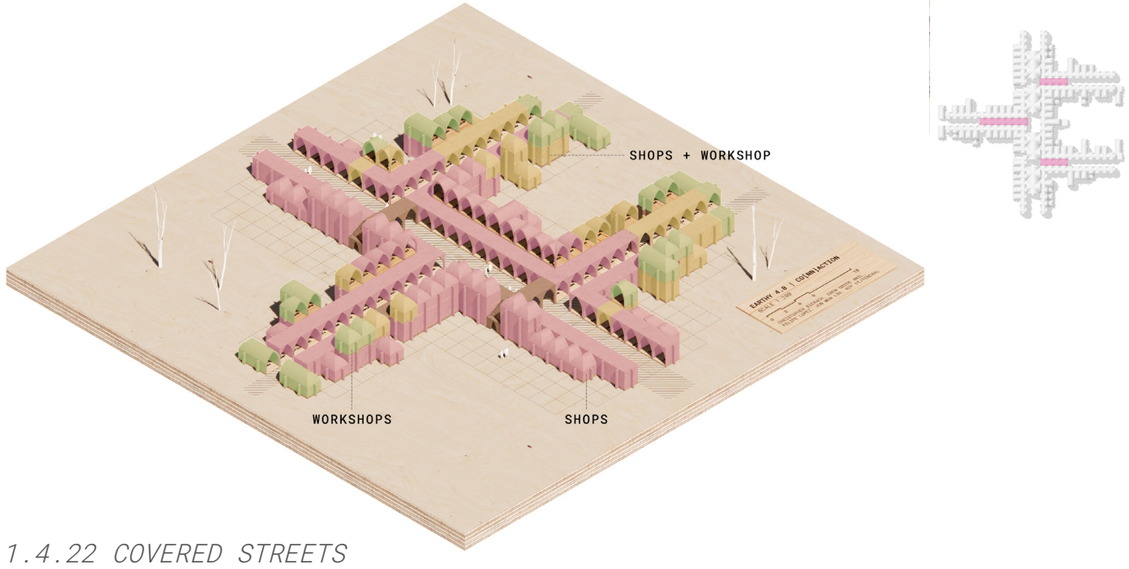
i. Is a shop facing another shop?

ii. Does any of these shops have a non-shop and non-street neighbor?

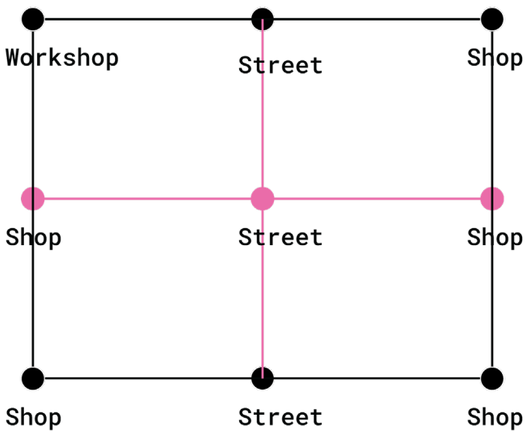
If the answer is yes to both of these questions, then that node is the cornerstone. After locating the cornerstone on the local streets(“tag”=0), all nodes on that street with shop neighbors are assigned module 10.1 and consequently covered.



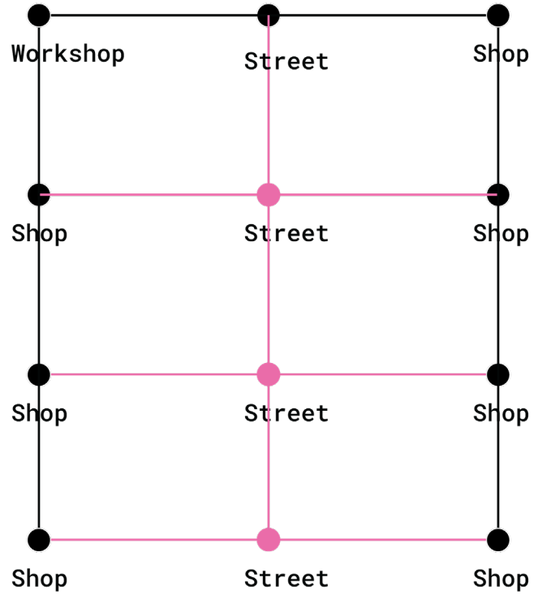
1.4.21 COVERED STREETS DIAGRAM IN SECTION



1.4.22 COVERED STREETS



1.4.23 FINDING THE CORNERSTONE



1.4.24 ASSIGNING MODULES TO STREET NODES

ADJACENCIES & MODULE TYPES

1.4 FINAL OUTPUT

After following all the steps explained before, all nodes which should have a module are assigned a module. Assigning modules is the main goal of this section, determining the “connection” attributes is only the approach we followed to be able to decide on module types. Along the way, nodes nested dictionary is updated to hold “connection” attribute and “module” attribute.

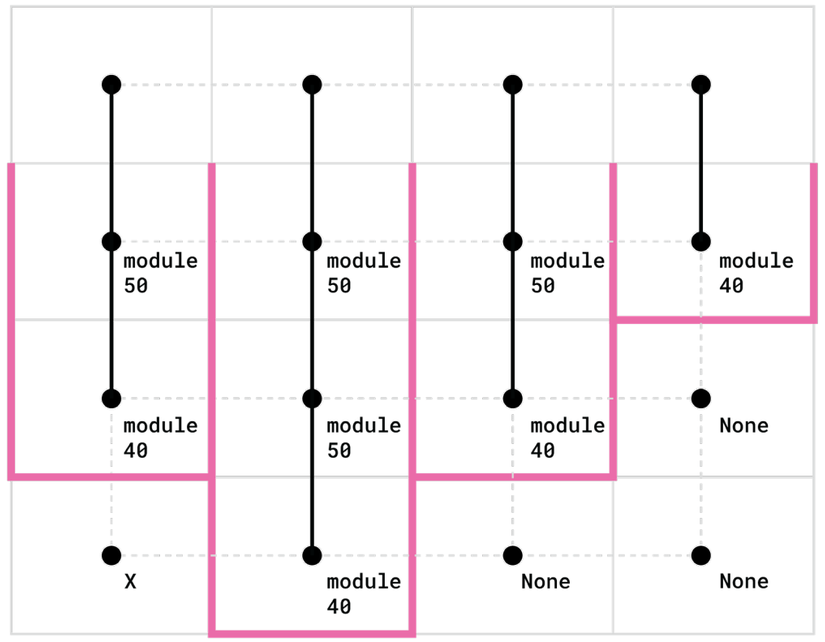
updated node dictionary:

{..., 5: {‘tag’: “Emilie”, ‘units’: 3,
‘district’: 7, ‘name’: 0, ‘use_frequency’: ‘M’,
‘type’: None, ‘floor’: 0, ‘connection’: [1,0,0,1]},...}

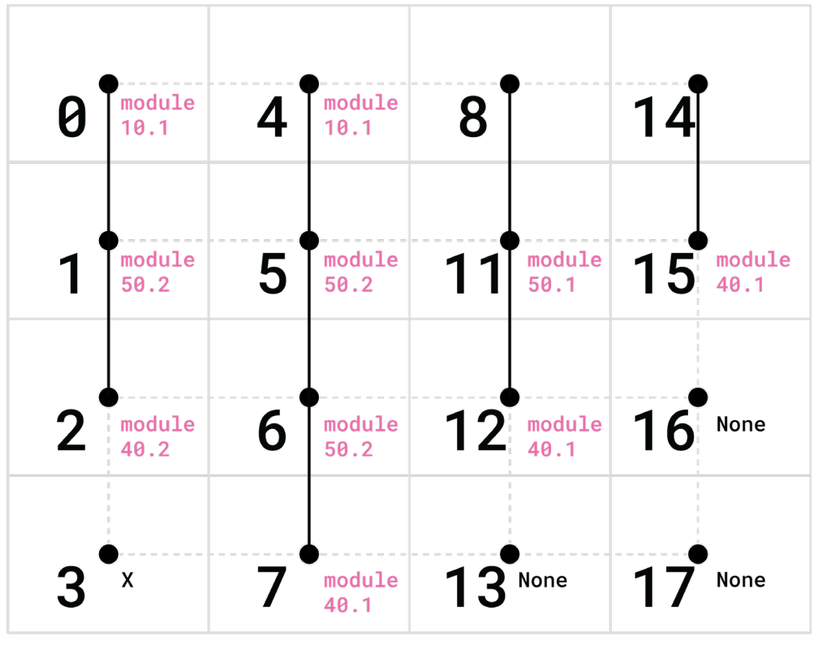
final node dictionary:

{..., 5: {‘tag’: “Emilie”, ‘units’: 3,
‘district’: 7, ‘name’: 0, ‘use_frequency’: ‘M’,
‘type’: None, ‘floor’: 0, ‘connection’: [1,0,0,1],
‘module’:50.2},...}

After this section is complete, only action left to take is orienting the modules on site, on the base grid. Nodes dictionary and adjacency(edge) dictionary are exported in text(.txt) format to Github in order to orient the modules. The reason behind exporting the adjacency dictionary is to make sure in this section and the following part we use the same data in the same order for accurate results. Data is transferred to the next stage: Orientation.



1.4.25 CELL MAP WITH ASSIGNED MODULE TYPES (CLOSED WALLS SHOWN IN PINK)



1.4.26 CELL MAP WITH ASSIGNED MODULE TYPES

ORIENT MODULES

1.5 OVERVIEW

Once the foundation of the base grid, the allocation of the requested programs and adjacency conditions are finalized, the given dictionary of nodes and the adjacency list from Python is injected in Grasshopper to establish the correct orientation for a given module type. The logics for each assigned orientation is first initiated by creating a mind map shown in FIG 1.5.1 to organize the nine different type of logics to finalize the correct orientation for each assigned module type.

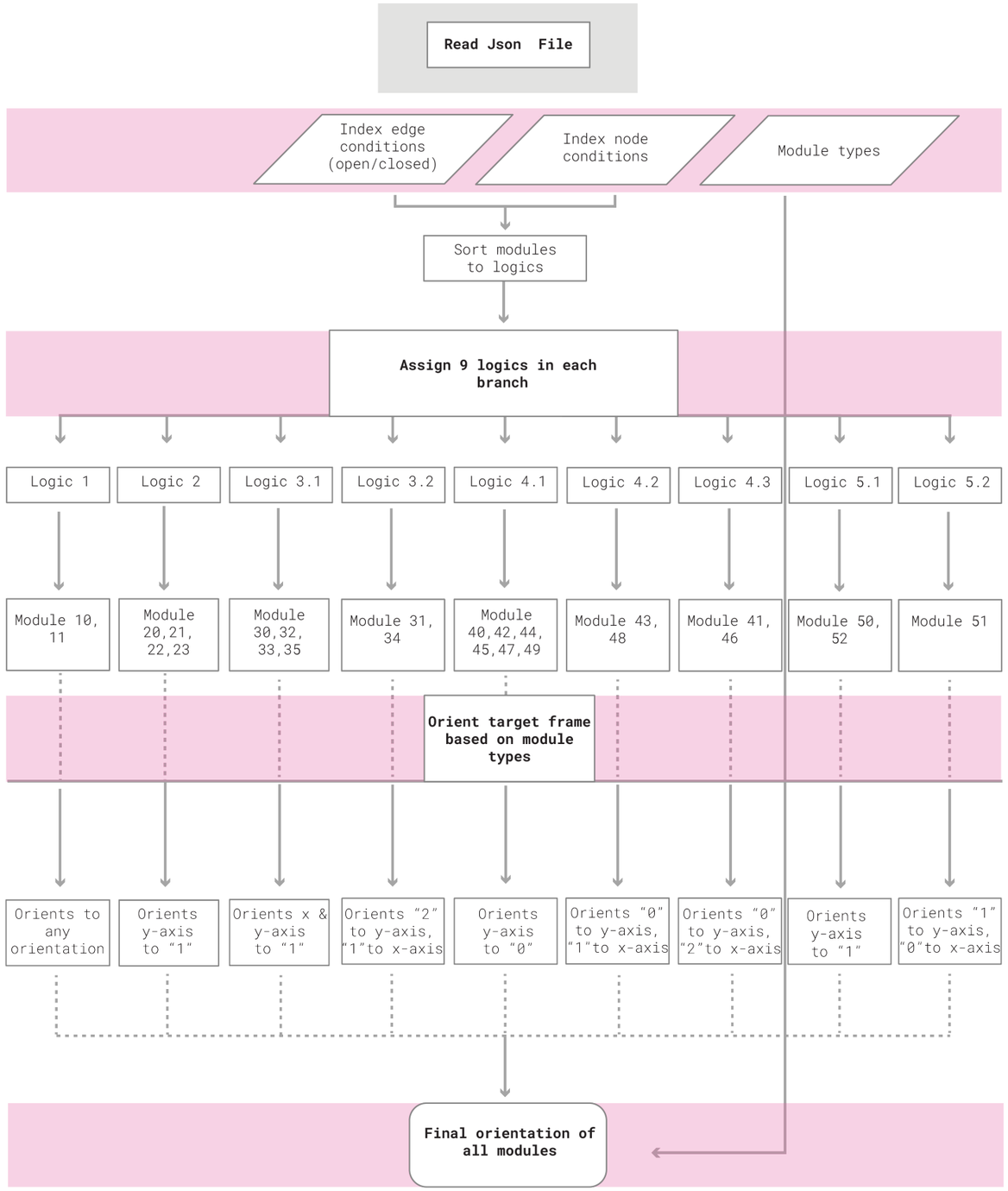


FIG 1.5.1 | MIND MAP OF ORIENTION

ORIENT MODULES

1.5 READ JSON FILE

The JSON output is first read into a GHPython script to receive the list of each nodes assigned to its module type, adjacency connection, and node to adjacency sequence seen in FIG 1.5.3. As a consequence,the definition takes in a logic_list, the type of module to evaluate and the current index of the node and sorts it to the logic it belongs to and to a nested index_list to be distributed to the 9 different logics required to orient the modules. Once the JSON file is understood, the evaluation of the module index will be used to identify the different connections from one module to another.

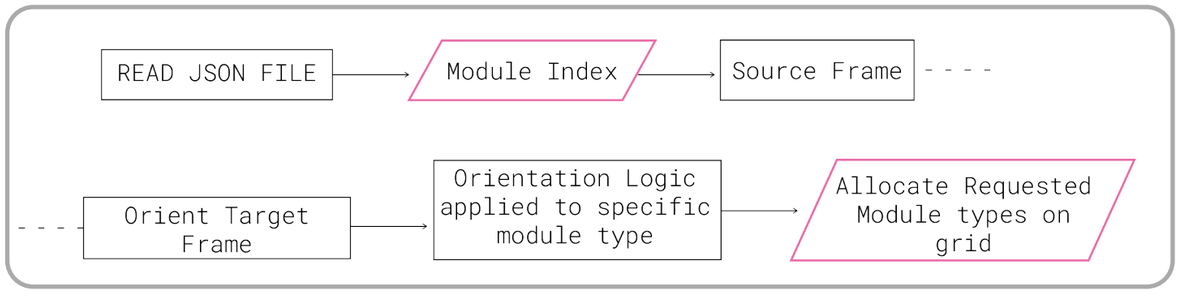


FIG 1.5.2 | FLOWCHART OF ORITATION SEQUENCE FROM PYTHON TO GH

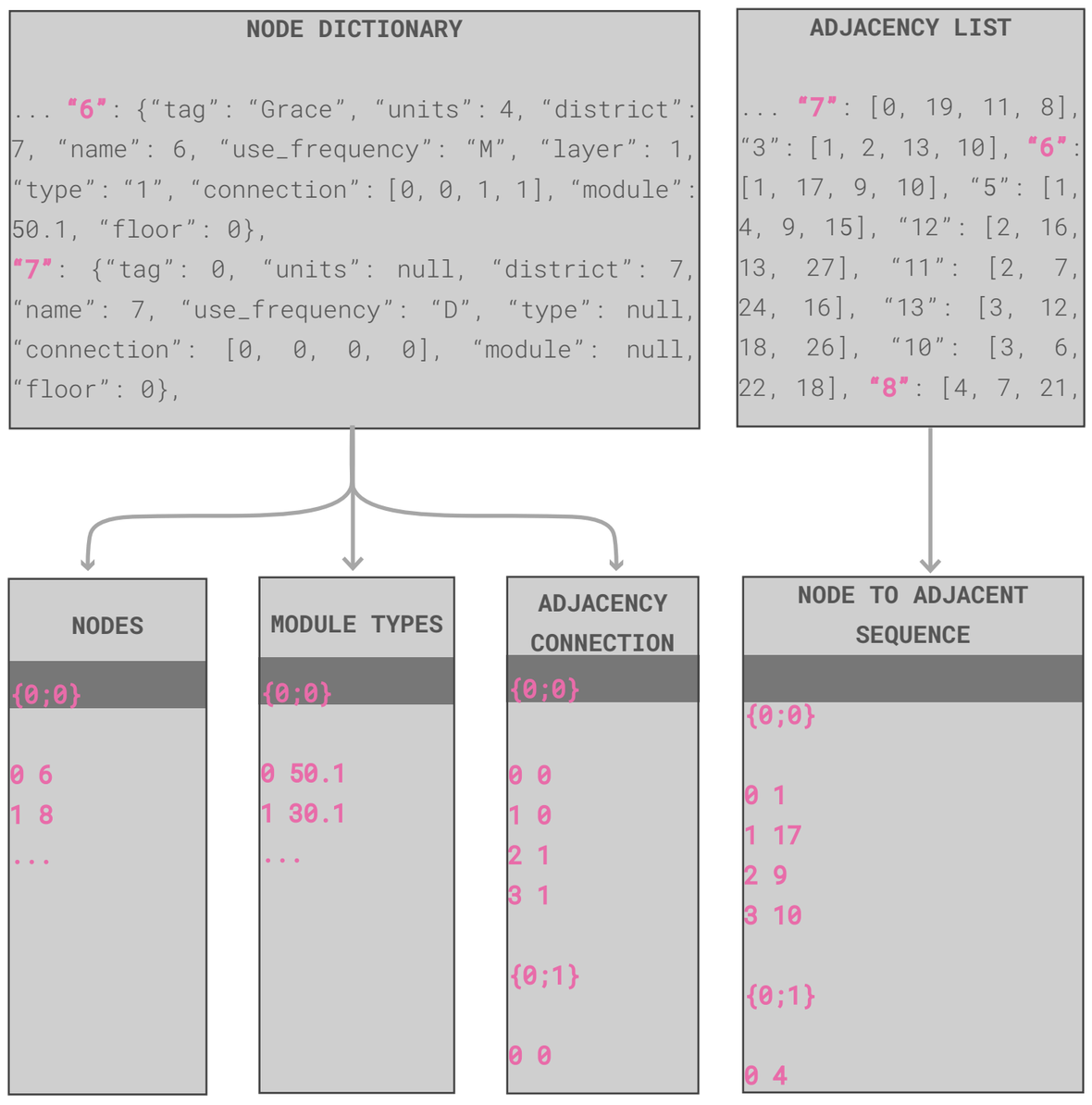


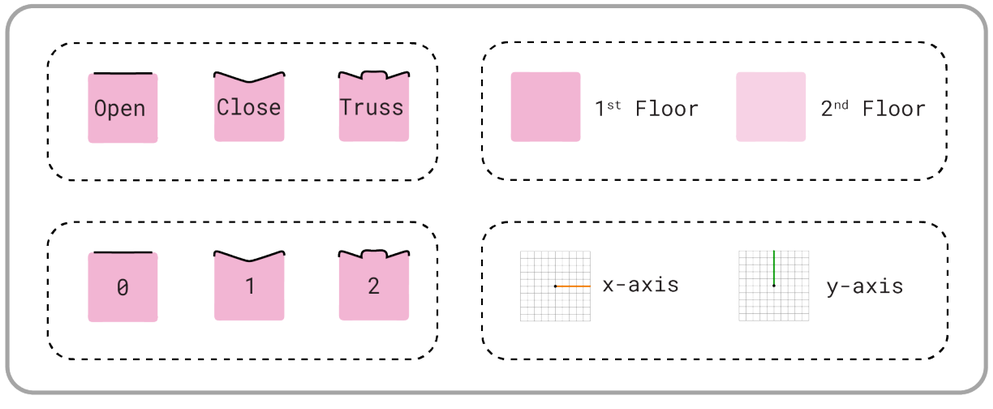
FIG 1.5.3 | EXAMPLE OUTPUT EXTRACTION JSON FILE

ORIENT MODULES

1.5 REPREESENTATION OF MODULE TYPES

Before assigning the final module types on the allocated base grid, a dummy or a representation of how is defined a closed, open or reinforced edge is first established. The exploration of its possibilities is shown in FIG 1.5.4. Afterwards, each module type is assigned to a type of connection by numbering them for 0 to be open, 1 to be closed and 2 for reinforcement or truss. The five different types is further elaborated in nine different logics according to the orientation of its constructed plane.

LEGEND



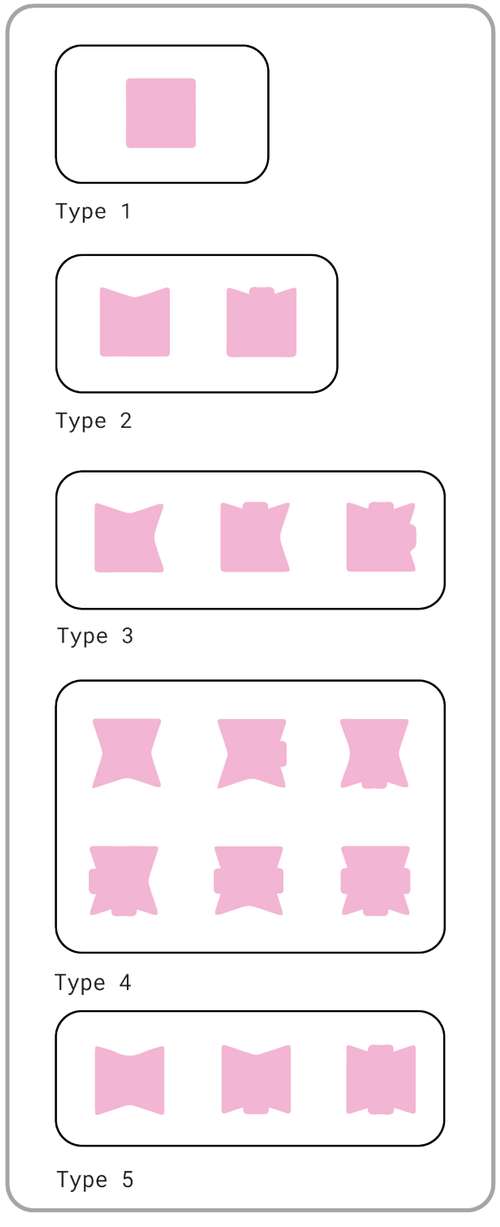


FIG 1.5.4 | MODULE TYPES

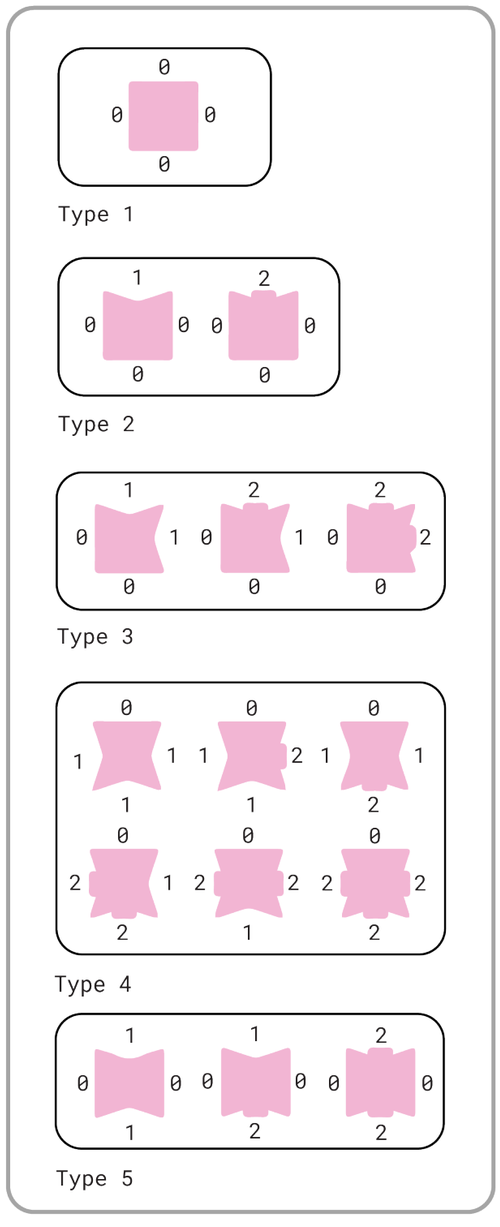


FIG 1.5.5 | SOURCE FRAME

ORIENT MODULES

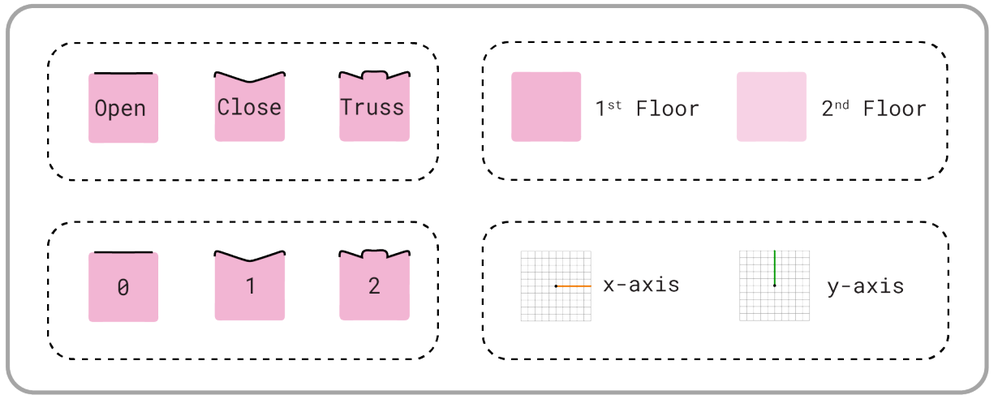
1.5 MODULES ORIENTATION LOGIC

As each logic represents a method of orientation, it applies to a specific module type only which is categorized in a **logic_list** = [logic_1, logic_2, logic_3.1, logic_3.2, logic_4.1, logic_4.2, logic_4.3, logic_5.1, logic_5.2]. Furthermore, the sequence of types which is defined as the index position corresponds to its index in GH and merges the component of all types, and is categorized once again from smallest to largest as a **type_sequence** = [10.1, 10.2, 20.1, 21.1, 20.1, 20.2, 30.1, 31.1, 32.1, 30.2, 31.2, 32.2, 40.1, 41.1, 42.1,43.1, 44.1, 45.1, 40.2, 41.2, 42.2, 43.2, 44.2, 45.2, 50.1, 51.1, 52.1, 50.2, 51.2, 52.2].

Finally, the list of nodes requires a specific orientation logic and adding each nodes to the logic they apply to defined as an **index_list** = [logic_1_index, logic_2_index, logic_31_index, logic_32_index, logic_41_index, logic_42_index, logic_43_index, logic_51_index,logic_52_index].

Each module types has an assigned constructed plane orientation according to its facing nodes. The assigned target frames of the module types is then finalized to its orientation sequence, shown in FIG 1.5.6.

LEGEND



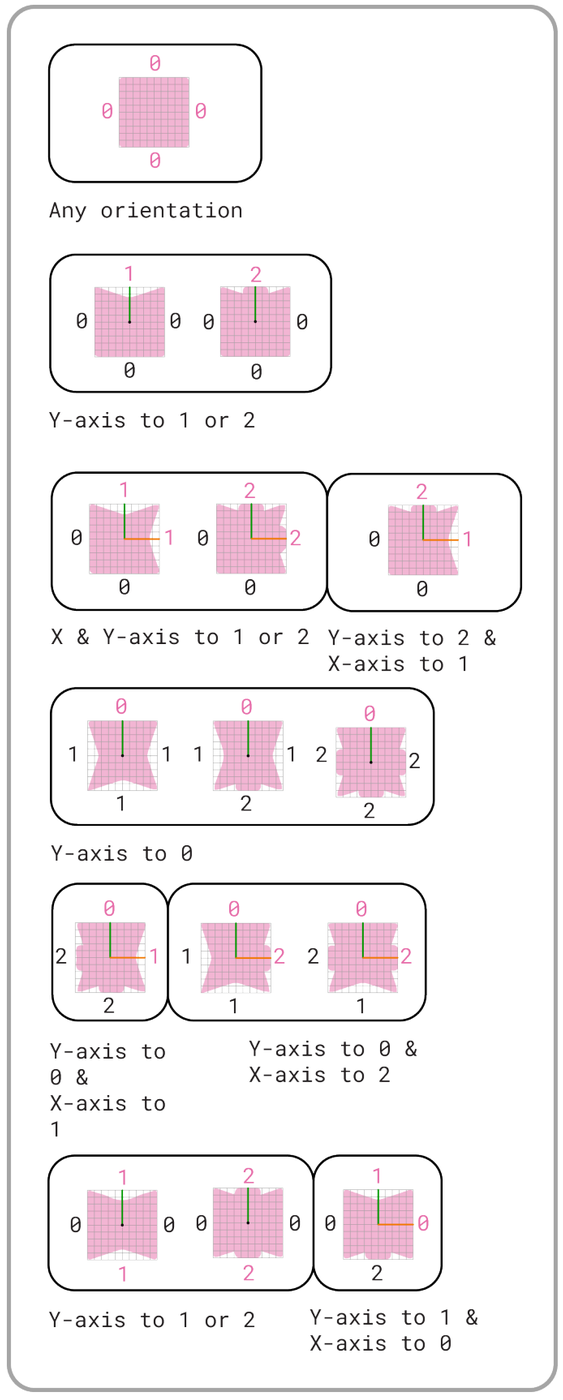


FIG 1.5.6 | ORIENT TARGET FRAME

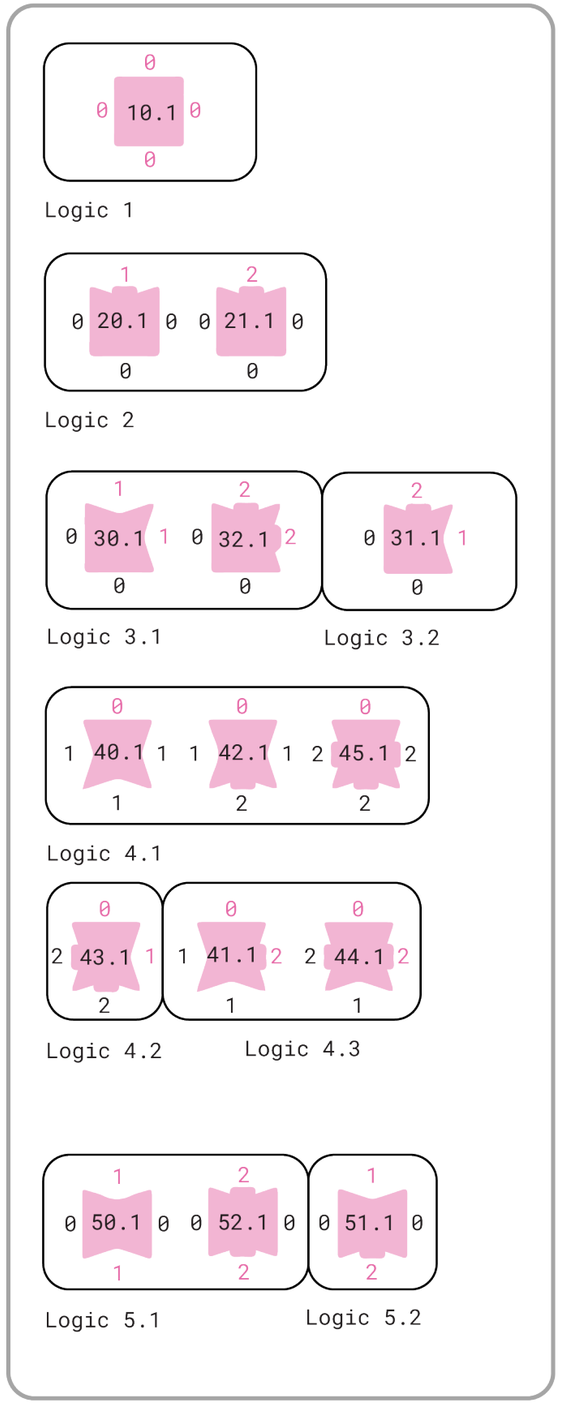


FIG 1.5.7 | FINAL ORIENTATION LOGIC

ORIENT MODULES

1.5 TABLE OF MODULE ORIENTATION LOGIC

Logics module n	Source Frame/Plane	Orient to target	Module types
Logic 1	0 0 ● 0 0	Any orientation	10.1 10.2
Logic 2	1 0 ● 0 0	Y-axis to 1	20.1 21.1 20.2 21.2
Logic 3.1	1 0 ● 1 0	X & Y-axis to 1	30.1 32.1 30.2 32.2
Logic 3.2	2 0 ● 1 0	Y-axis to 2 & X-axis to 1	31.1 31.2
Logic 4.1	0 1 ● 1 1	Y-axis to 0	40.1 42.1 45.1 40.2 42.2 45.2
Logic 4.2	0 2 ● 1 2	Y-axis to 0 & X-axis to 1	43.1 43.2
Logic 4.3	0 1 ● 2 1	Y-axis to 0 & X-axis to 2	41.1 44.1 41.2 44.2
Logic 5.1	1 0 ● 0 1	Y-axis to 1	50.1 52.1 50.2 52.2
Logic 5.2	1 0 ● 0 2	Y-axis to 1 & X-axis to 0	51.1 51.2

FIG 1.5.8 | 9 LOGICS ORIENTATION

ORIENT MODULES

1.5 MODULAR METHOD OF PLACING MODULE TYPES

The final allocation of the dummies are further placed onto the base grid to visualize the multitude of constructed planes orientations in relation to their neighbors. Once the finalize orientation sequence is established, the modules types are assigned to a program: shop/workshop/shop+workshop. There are also fixed conditions defined as the street covers, bridges and staircases.

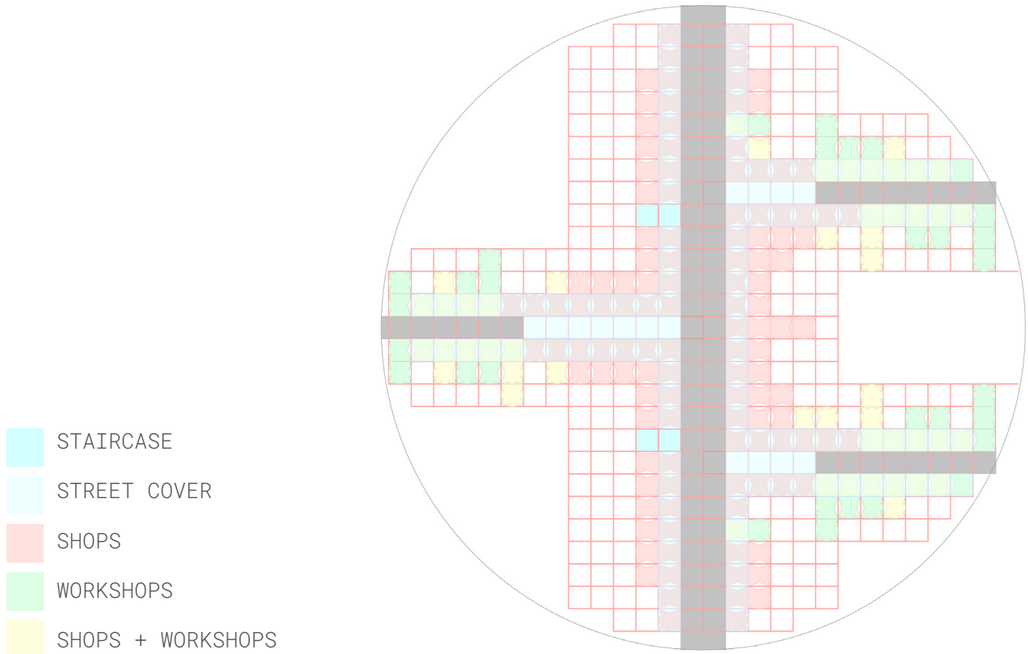


FIG 1.5.13 | ALLOCATE REQUEST (1ST + 2ND

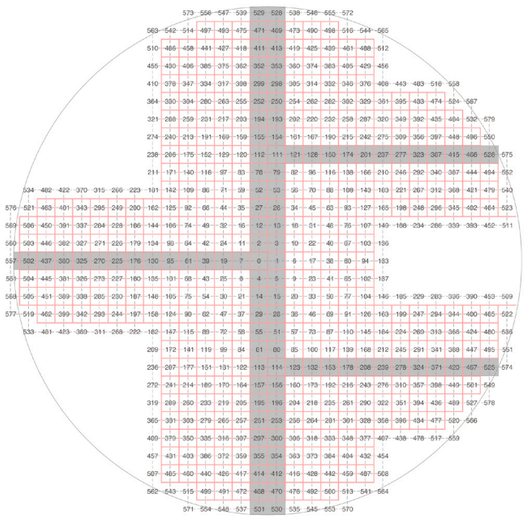


FIG 1.5.9 | BOUNDARY AT INTERSECTION

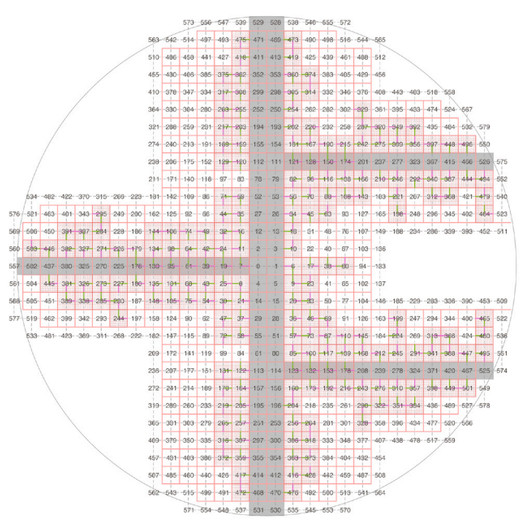


FIG 1.5.10 | ORIENT TARGET FRAME (1ST

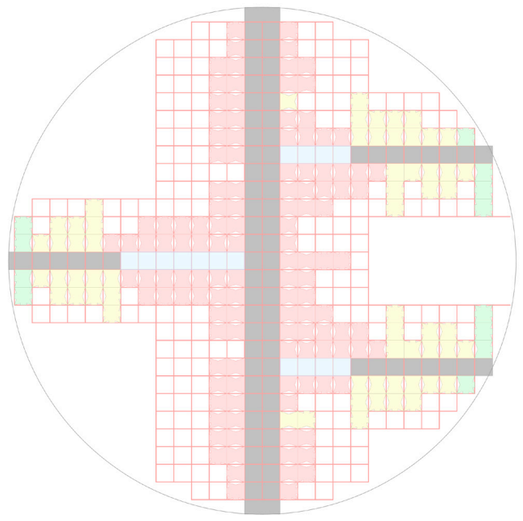


FIG 1.5.11 | ALLOCATE REQUEST (1ST

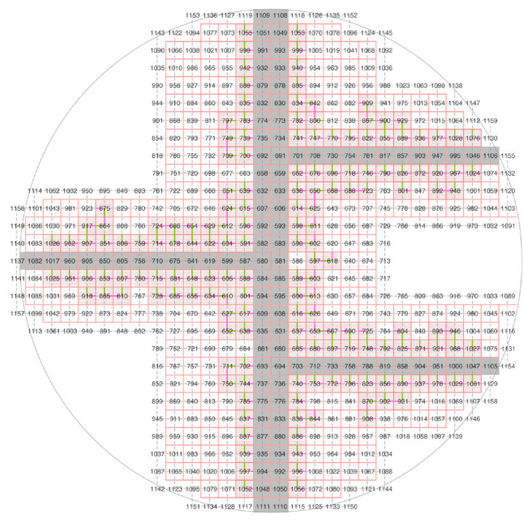


FIG 1.5.12 | ORIENT TARGET FRAME (2ND

ORIENT MODULES

1.5 MESH MODULE TYPES PLACED ON GRID

The urban configuration of the final module types are then replaced by their representational model to express the importance of modularity to allow its application in different scenarios, and therefore allow each module designs to be replaced if desired.

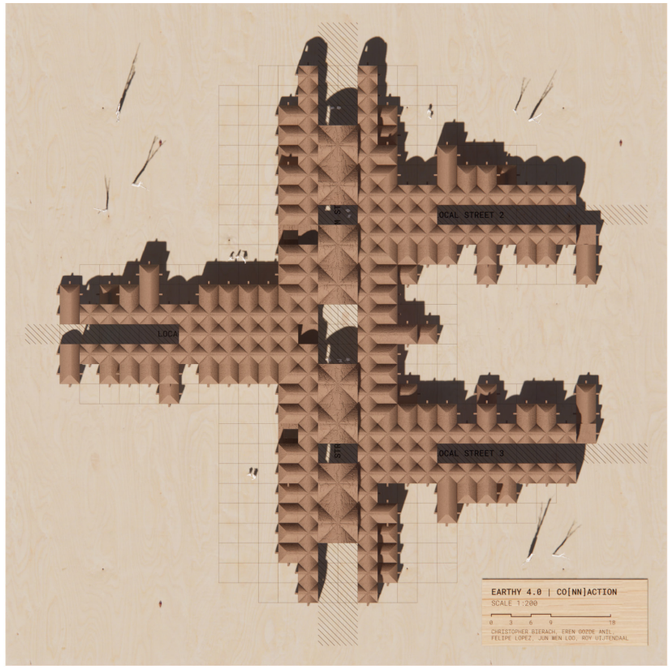


FIG 1.5.18 | ALLOCATE MODULE TYPES + FIXED TYPES

FIG 1.5.14 | INTERSECTION

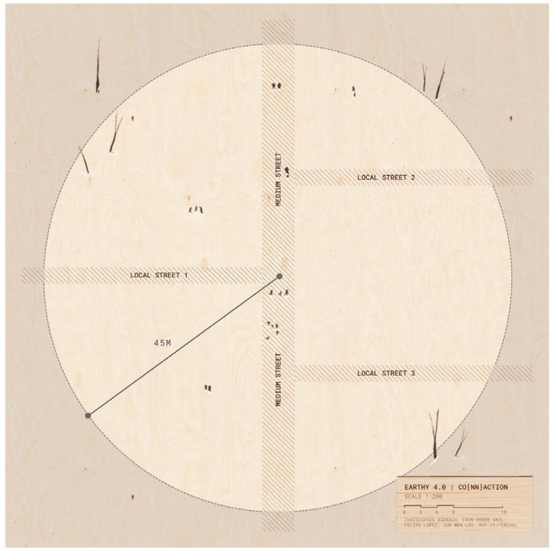


FIG 1.5.15 | BOUNDARY

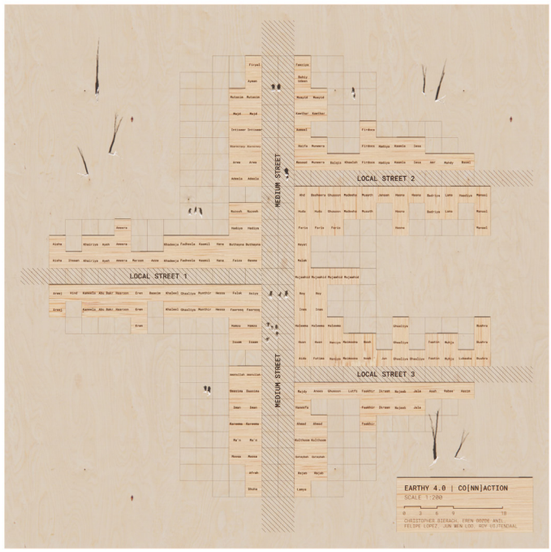


FIG 1.5.16 | ALLOCATE REQUEST (1ST

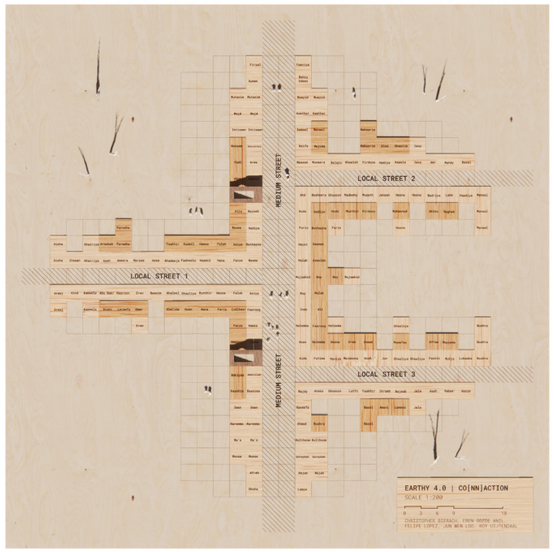


FIG 1.5.17 | ALLOCATE REQUEST (2ND

ORIENT MODULES

1.5 MESH MODULE TYPES PLACED ON GRID

The boundary conditions of 3 local streets intersecting with a medium street is an example of what can be furthered assigned throughout the entire camp. The elaboration of the modular bazaar allows a positive prospective for the local inhabitants to request their desires and allocate them accordingly.

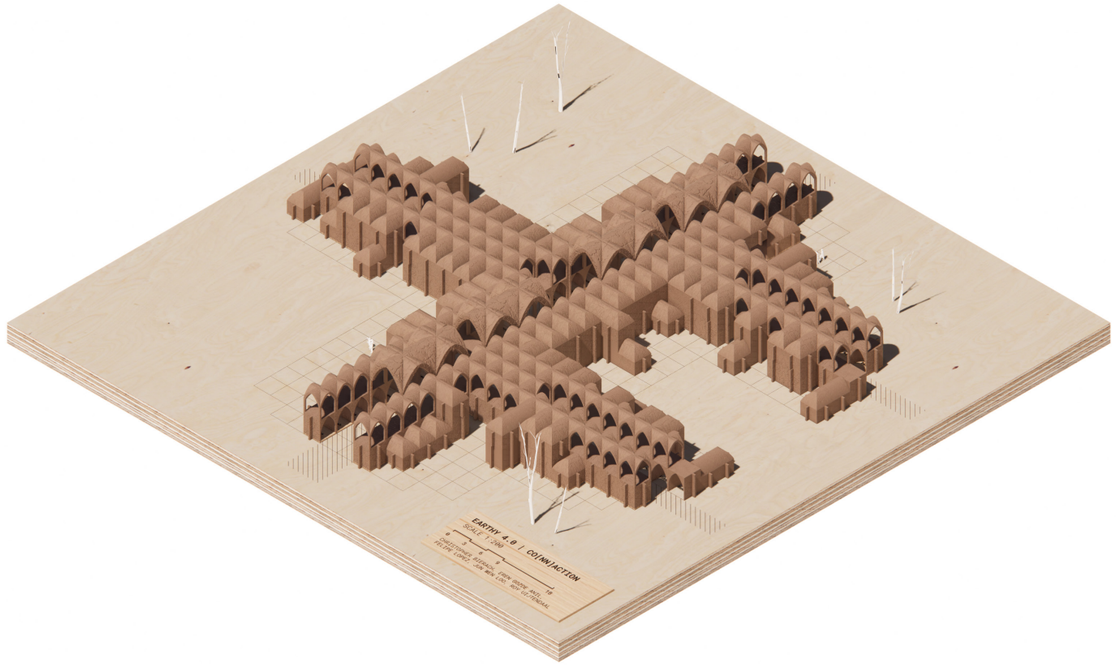


FIG 1.5.22 | ALLOCATE MODULE TYPES + FIXED TYPES

FIG 1.5.18 | INTERSECTION

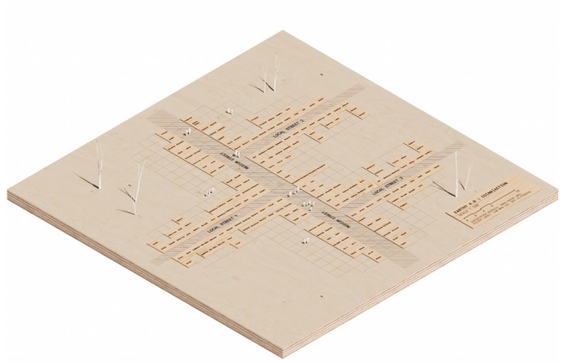


FIG 1.5.19 | 1ST FLOOR ALLOCATION

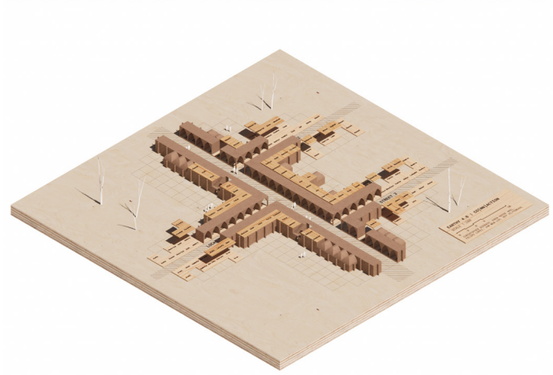


FIG 1.5.20 | 2ND FLOOR ALLOCATION

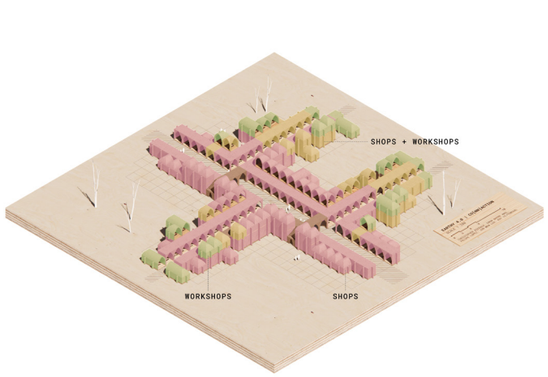


FIG 1.5.21 | OVERVIEW OF PROGRAMS

Shaping 2.0

FORMING

2.0 OVERVIEW

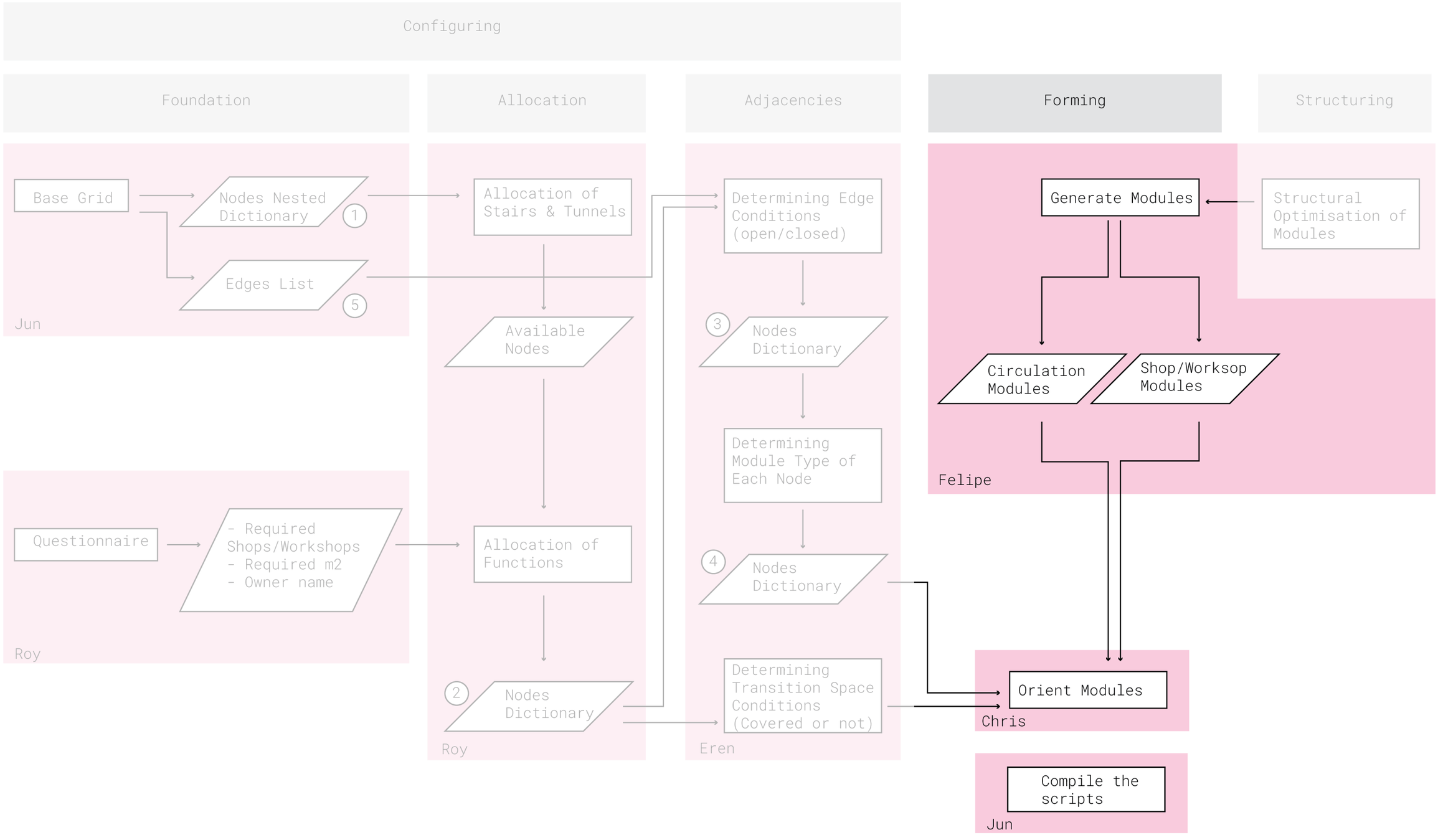


FIGURE 2.0.1. COMPLETE PROCCES FLOWCHART

EXPLORATION APPROACH

2.0 CLOSE IS NOT THE SAME THAT A WALL

The first explored idea consists of optimizing an entire open module and then introducing walls depending on the opening condition. However, early on in the evaluation, this approach shows several difficulties that give the base to further exploration.

First, this solution creates a high repetitive space without the local condition recognition. Furthermore, all the ceilings will be the same, independent of the modules position.

Second, the structure is also done not recognizing the local condition. As shown in figure 2.0.2, each closed relation will become a not structurally necessary wall because the module in figure 2.0.1 already solves all the structural requirements.

In conclusion, we need to explore options where the ceiling and structure recognize the condition of the local module.

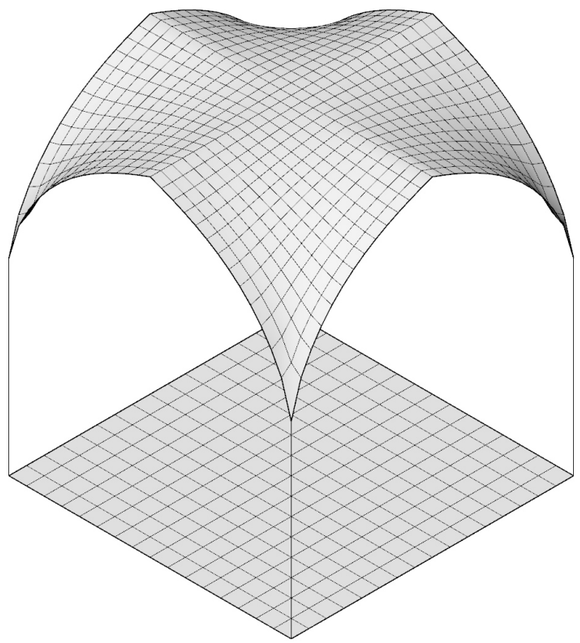


FIGURE 2.0.1: RELAXED MESH WITH FOR OPENINGS

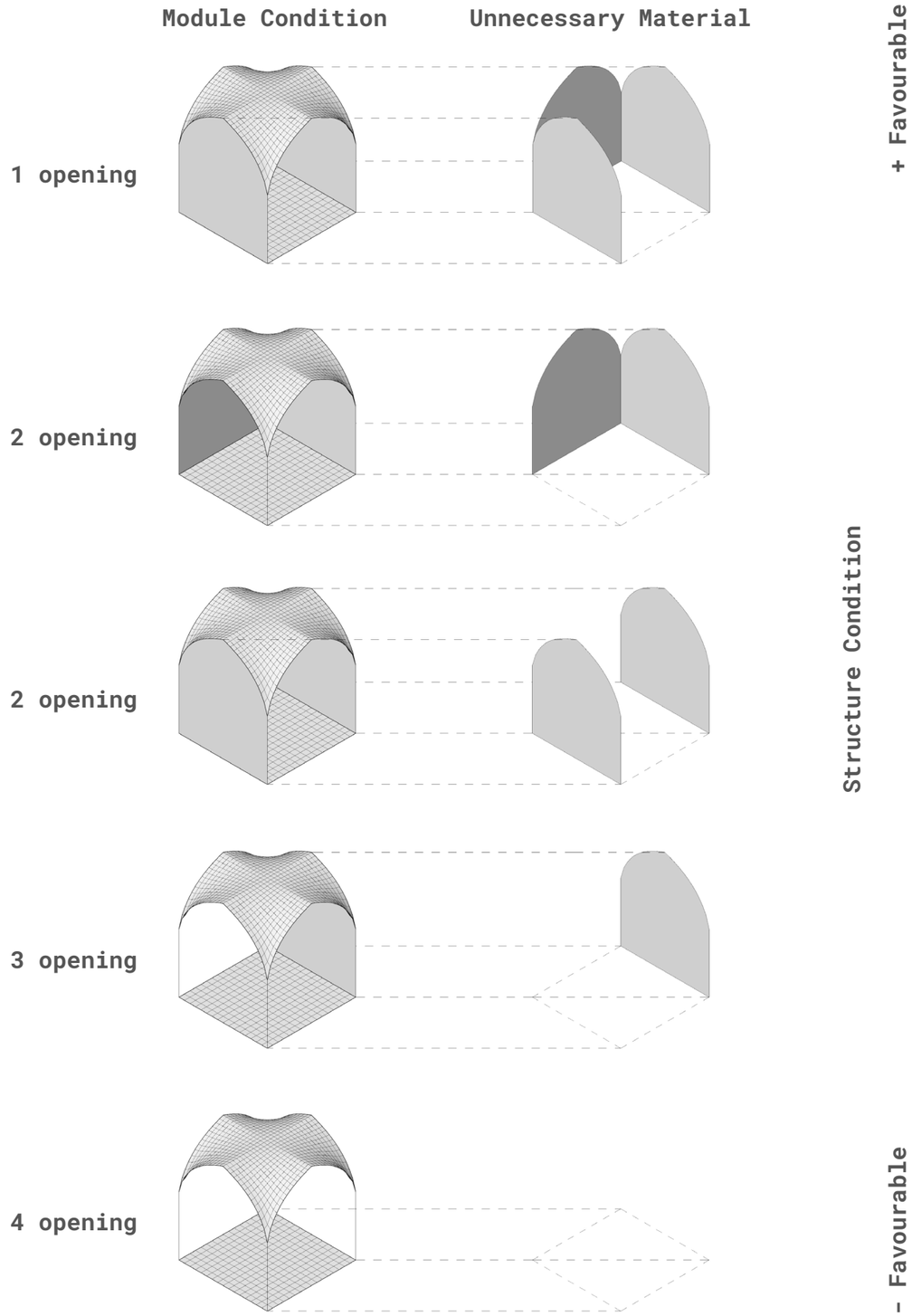


FIGURE 2.0.2: DIFFERENT MODULES AND THE UNNECESSARY STRUCTURE

EXPLORATION APPROACH

2.1 MESH RELAXATION WITH VARIABLE ANCHOR CONDITION

The first idea established to have a space recognition through the ceiling was to use vertical walls and columns to support the ceilings. Thus, the roof must be supported in continuous lines when it is in a close condition and at a point when it is in an open state. Consequently, different vaults were generated.

In this approach, the vaults were generated by the dynamic relaxation of a mesh. To obtain the different conditions, the anchor is related to the opening condition. Where a close condition is required, all the edge is anchored. On the other hand, if an open condition is needed, only the vertices are anchored. Consequently, the ceiling recognizes the space condition, as figure 2.1.2 shows.

After this exploration was discovered that the openings of all modules must match; if not, unwished openings occur, as is shown in figure 2.1.1

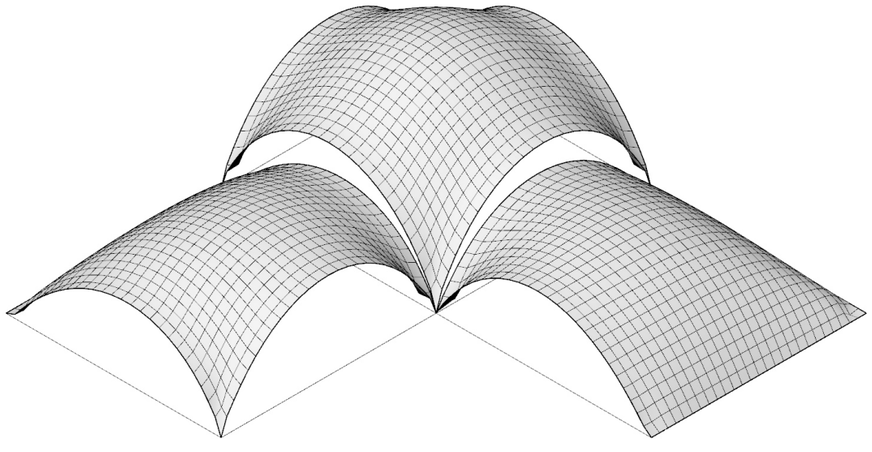


FIGURE 2.1.1: COMBINATION OF DEFERENTS MODULES GENERATED WITHOUT RESTRICTIONS

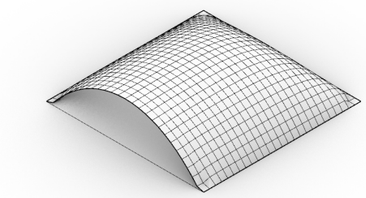
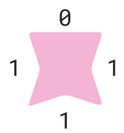
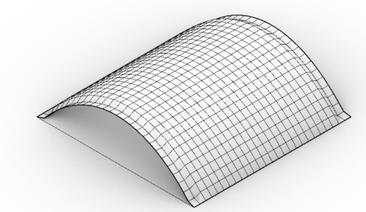
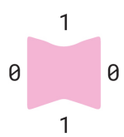
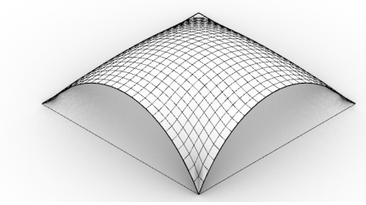
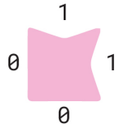
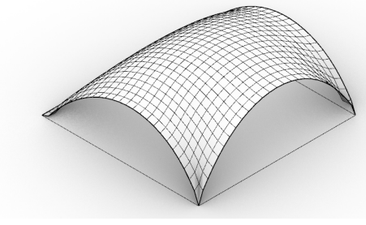
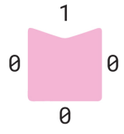
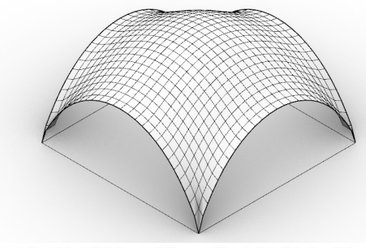
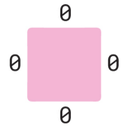
	Modules base on a relaxed mesh and anchor condition.	Opening Condition and previous representation
1 opening		 Type 4
2 opening		 Type 5
2 opening		 Type 3
3 opening		 Type 2
4 opening		 Type 1

FIGURE 2.1.2: MODULES GENERATED WITHOUT RESTRICTIONS

EXPLORATION APPROACH

2.1 ADDING A RESTRICTION TO THE RELAXATION HEIGHT

Taking into account that the modules openings must be the same, an additional restriction was added. The hypothesis was that if all openings have the same height and are generated from the same mesh will end on similar arches.

As is shown in figure 2.1.3, this was not the case, even though several ways to achieve this condition were tested.

First, in the relaxation, a target height was introduced to all the points in the ridge.

Second, a collinear restriction to all the ridge points was added.

Third, a coplanar restriction to all the ridge points was added.

In all the cases, the problem described in figure 2.1.4 was present.

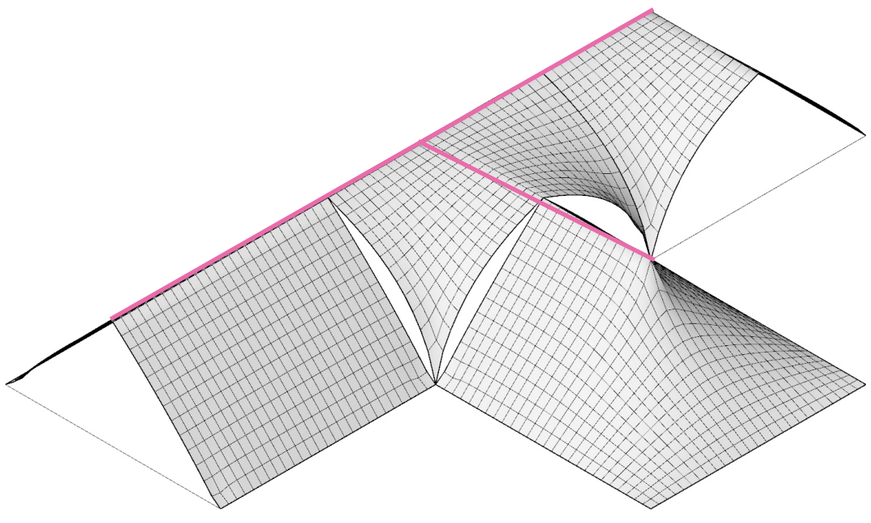


FIGURE 2.1.3: COMBINATION OF DEFERENTS MODULES GENERATED WITH TOP COLLINEAR RESTRICTION

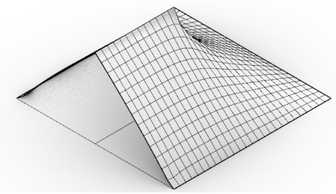
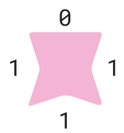
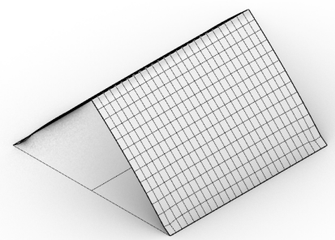
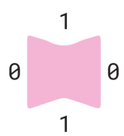
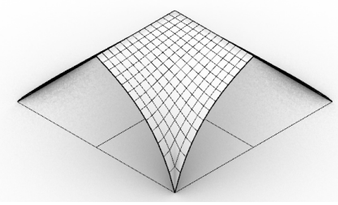
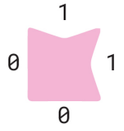
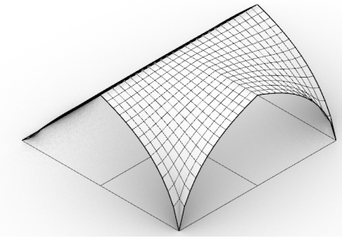
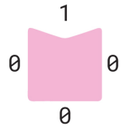
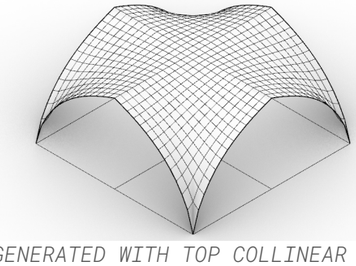
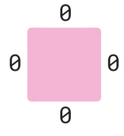
	Modules base on a relaxed mesh and anchor condition.	Opening Condition and previous representation
1 opening		 Type 4
2 opening		 Type 5
2 opening		 Type 3
3 opening		 Type 2
4 opening		 Type 1

FIGURE 2.1.4: MODULES GENERATED WITH TOP COLLINEAR RESTRICTION

EXPLORATION APPROACH

2.1 TESSELLATION DIVISION

In this case, a different approach was taken. As is shown in figure 2.1.6, different tessellation was created for each expected condition, only anchoring the vertices and taking into account the following principles:

First, the closed edges will not have division.
Second, all the tessellation start from the diagonal division.
Third, the division from the edge to the center is the same for all the sides.
Fourth, the number of divisions of the open edges is the same that at the third point.

How is shown in figure 2.1.5, the result is better in terms of openings similarities. Nevertheless, the structure behavior worsens with respect to the previous cases due to the absence of arches in the close faces.

From this approach, an important conclusion was taken, the number of steps to raise the ridge and arches must be the same.

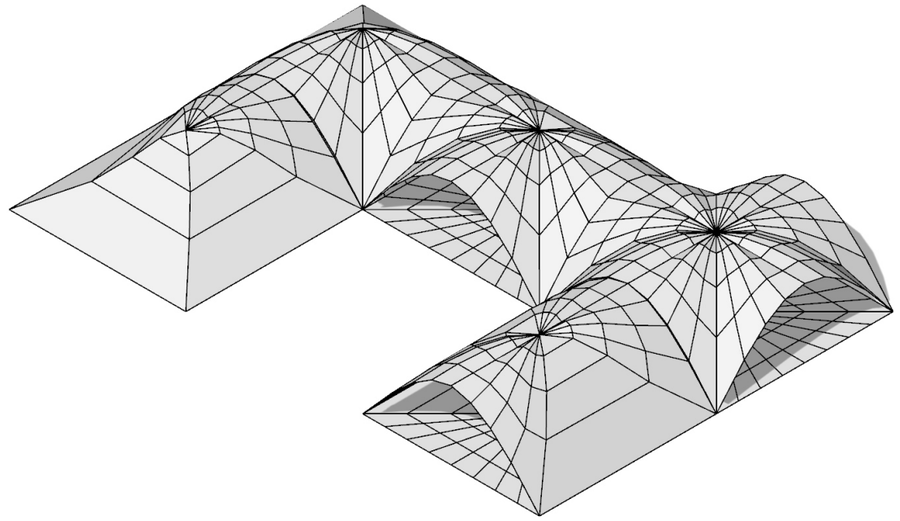


FIGURE 2.1.5: COMBINATION OF ALL TYPE OF MODULES

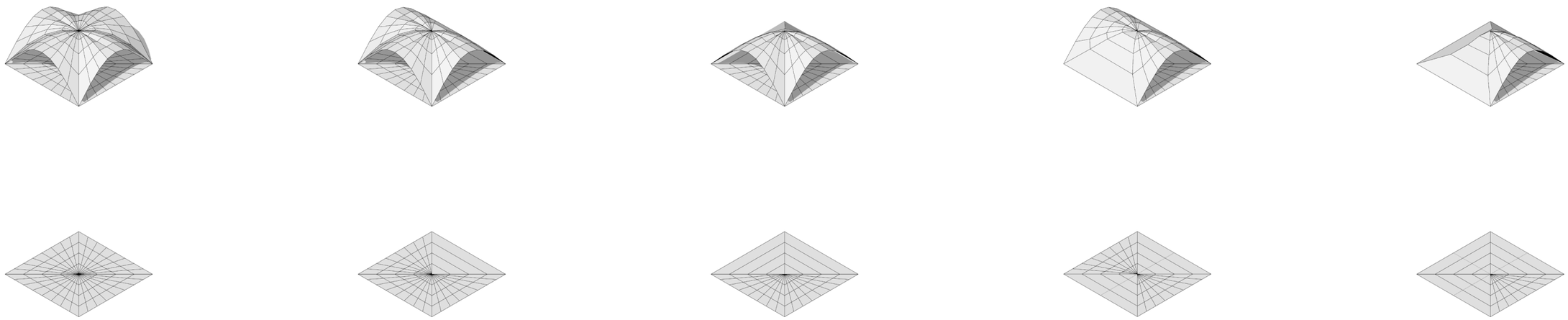


FIGURE 2.1.6: TESSELLATION AND THE CORRESPONDING MODULES GENERATED BY DYNAMIC RELAXATION

EXPLORATION APPROACH

2.1 UNDERSTANDING MODULARITY

After the previous exploration was done, a supplementary statement was established.

“The problem is not a match in the arches curvature; the problem is in the spacial structure of the ceilings.”

With this statement, we discovered-established three different pieces that can create all the necessary modules. The corners relation defines these pieces, depending on if it is open-open, open-closed or closed-closed.

As is shown in figure 2.1.8, all the opening conditions can be created with these modules. On the other hand, as is shown in figure 2.1.7, these pieces are already in the previous approaches. Nevertheless, each piece generated a different curvature opening, as is shown in figure 2.1.7.

How to get similar curvatures in the arches for all the figures is explored in three different approaches in the following pages.

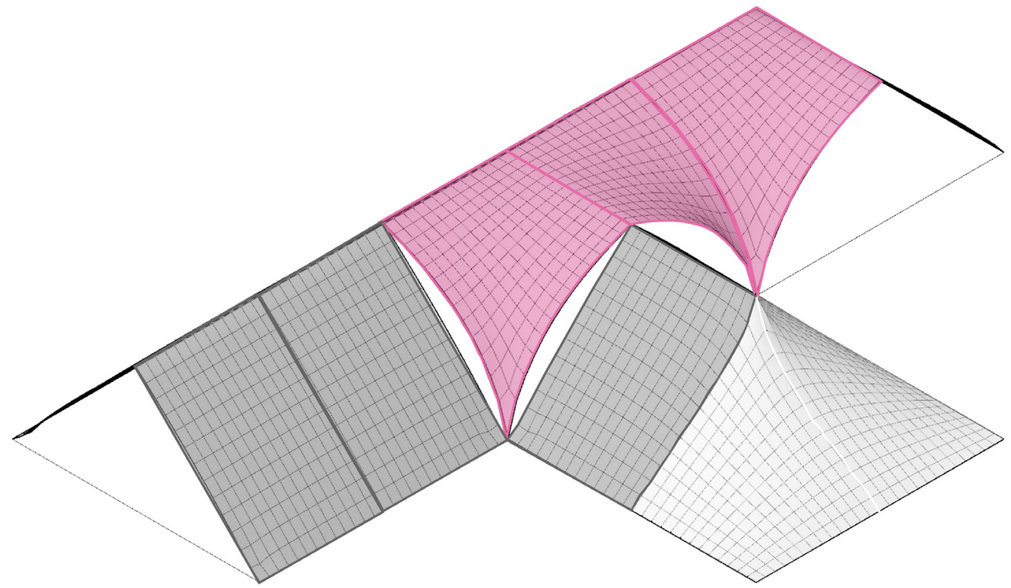


FIGURE 2.1.7: CEILING TESSELLATIONS PART ON DEFERENTS MODULES

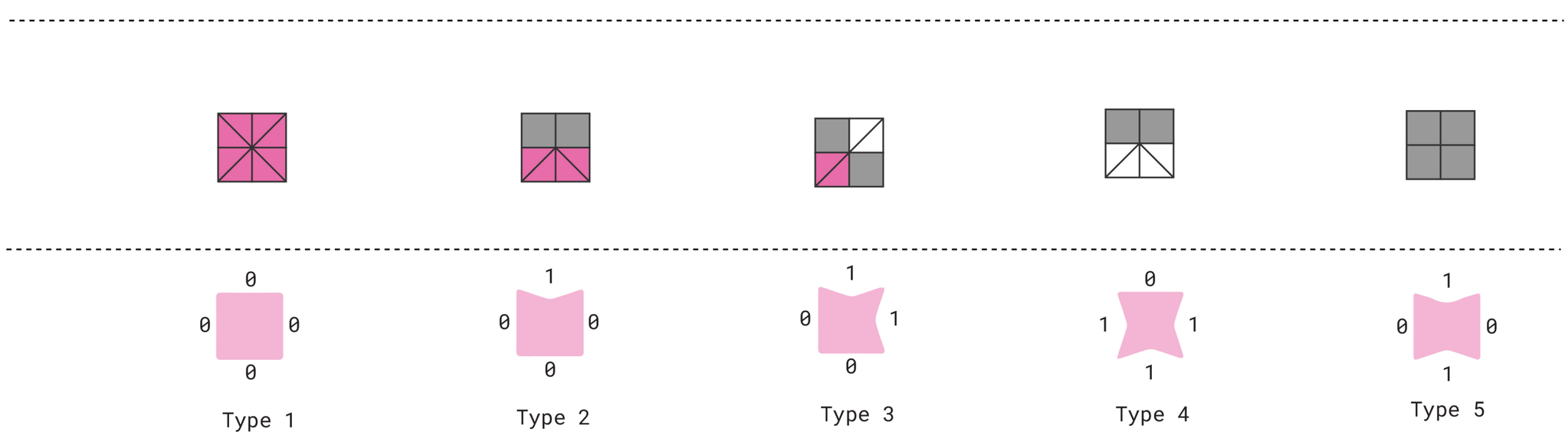


FIGURE 2.1.8: MODULE TYPE AND CEILING TESSELLATION AND MODULARITY

EXPLORATION APPROACH

2.1 GEOMETRIC APPROACH

In this case, a geometric approach was taken.

The following steps were developed from a relaxed mesh with ridge colinearity, as figure 2.14 shows.

First, identify the arch curvature. (Figure 2.1.10) Piece 1 is already created
Second, extrude the curvature in the half of the edge length (Figure 2.1.11)
Third, intersect two extrusions starting in the corner (figure 2.1.12)
Fourth, take the top part of the intersection (Figure 2.1.13). Piece B is created
Fifth, take the bottom part of the intersection (Figure 2.1.14). Piece C is created
Sixth, Combine the pieces depending on the module requirement.

As figure 2.1.9 shows, all the modules can be created and have perfect space continuity.

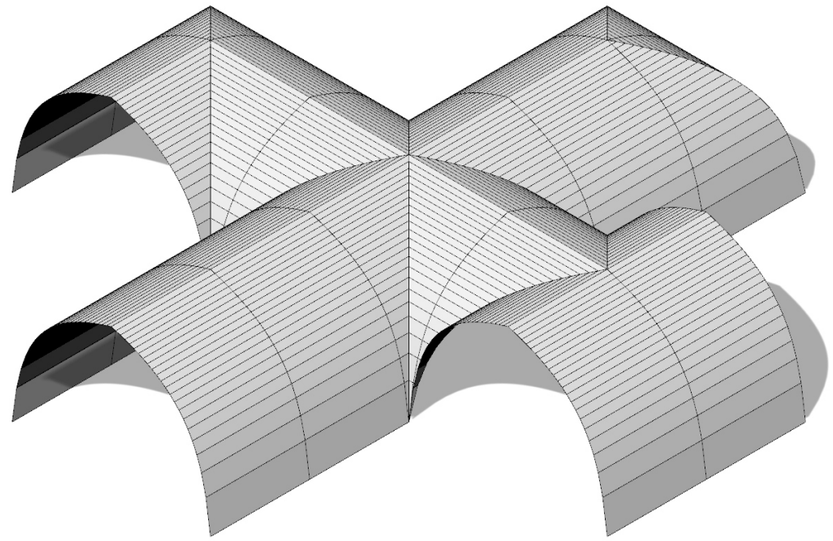


FIGURE 2.1.9: COMBINATION OF ALL THE MODULES GENERATED BY CURVE EXTRUSION APPROACH

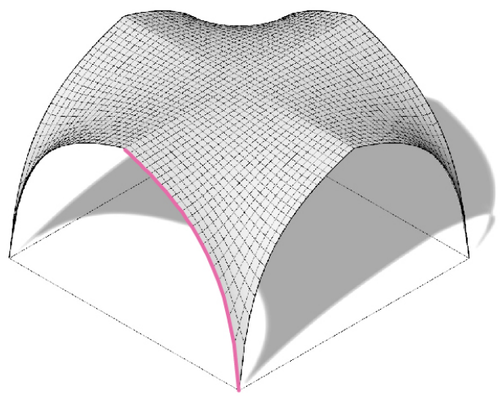


FIGURE 2.1.10: RELAXED MESH TO GET CURVE

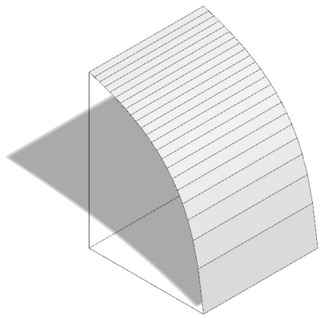


FIGURE 2.1.11: CURVE EXTRUSION

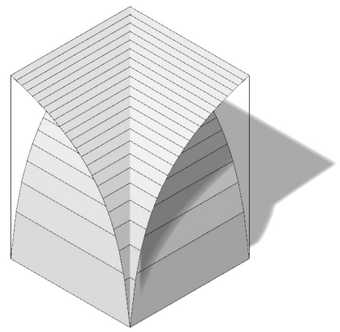


FIGURE 2.1.12: EXTRUSIONS INTERSECTION

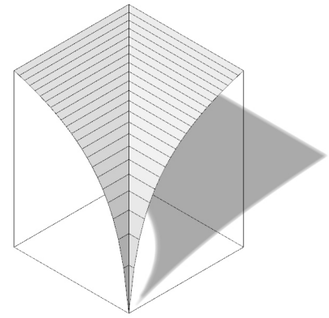


FIGURE 2.1.13: CORNER 11

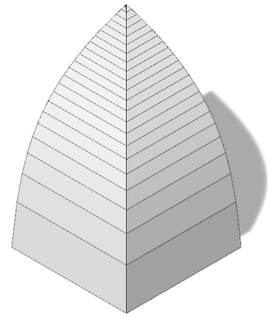


FIGURE 2.1.14: CORNER 00

EXPLORATION APPROACH

2.1 ONE TESSELLATION APPROACH

In this case, the idea of optimizing the structure by the use of non-regular tessellation was explored. Furthermore, to have perfect space continuity, the idea of starting from a continuous mesh was explored.

To obtain all the modules, the followings steps were developed:

first, Create a tessellation that recognizes all the opening conditions. (Figure 2.1.16)
Second, relax the mesh anchoring all the borders and 2 points in the middle of the tessellation (figure 2.1.15)
Third, determine the boxes that contain the different modules openings condition (figure 2.1.17)
Fourth, intersect the boxes with the relaxed mesh (figure 2.1.18)

The different modules were obtained; for example, a module with three openings is shown in figure 2.2.1.19. Consequently, space continuity is not an issue.

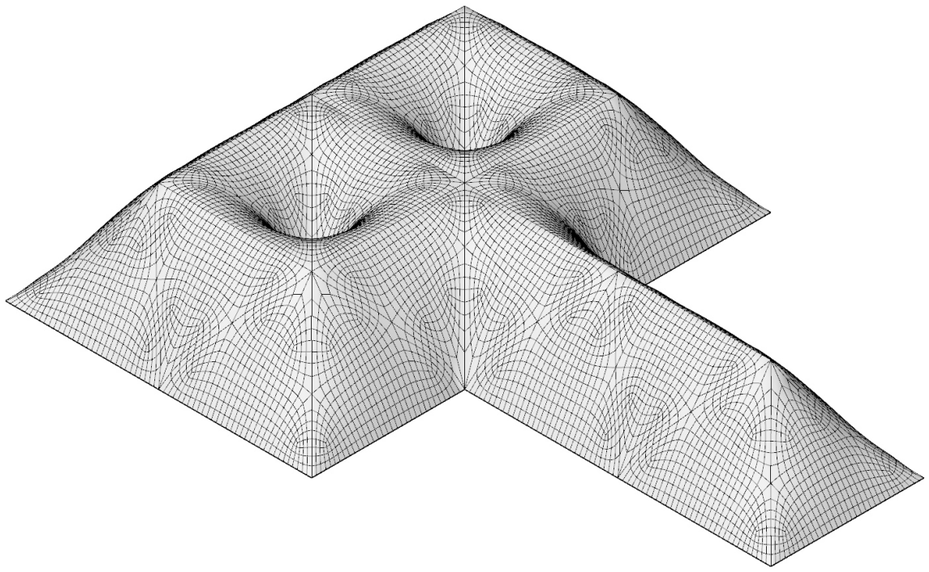


FIGURE 2.1.15: RELAXED MESH WITH ALL MODULES TYPE POSSIBILITIES

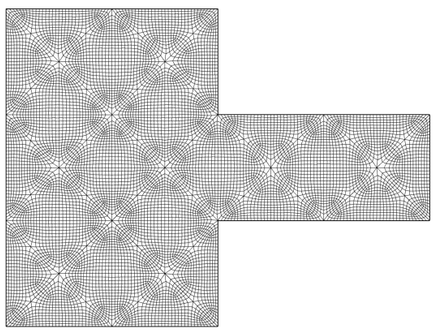


FIGURE 2.1.16: INITIAL TESSELLATION

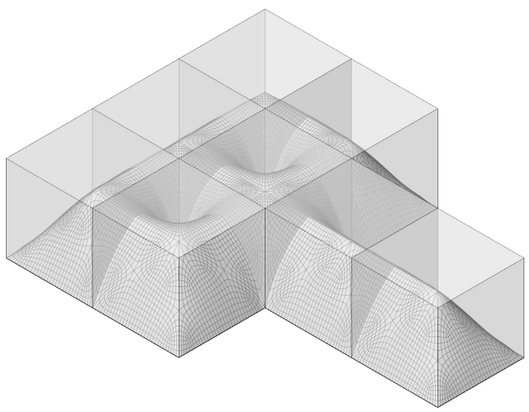


FIGURE 2.1.17: RELAXED MESH AND MODULES BOXES

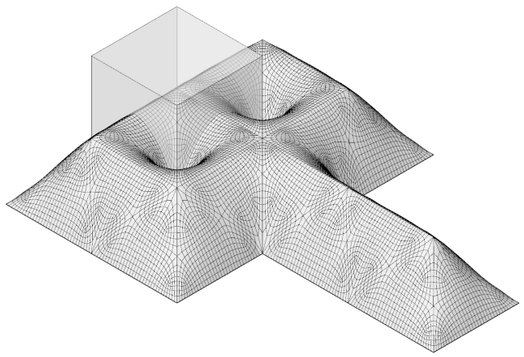


FIGURE 2.1.18: LOCATIONS OF MODULE WITH 3 OPENINGS

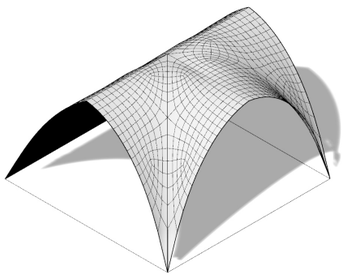


FIGURE 2.1.19: MODULE WITH 3 OPENINGS

EXPLORATION APPROACH

2.1 TOPOLOGICAL APPROACH

In this case, all the previous knowledge was taken to come up with a topological solution. As is already established in previous steps, the following conditions must be satisfied to solve the problem:

First, one piece must be created for each corner opening relation, open-open, open-closed or closed-closed.
Second, the steps to raise the maximum height must always be the same.
Third, steps sizes must always be the same.

With the previous restrictions, the boundary conditions were established for each piece of the module (figure 2.1.21)

To comply with all restrictions, a unique piece with all the conditions was established (Figure 2.1.20)

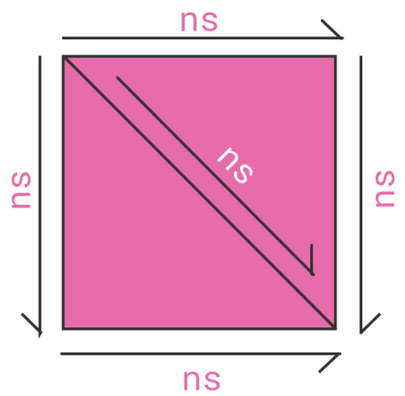


FIGURE 2.1.20: MODULE WITH ALL THE TOPOLOGICAL RELATION REQUIREMENTS

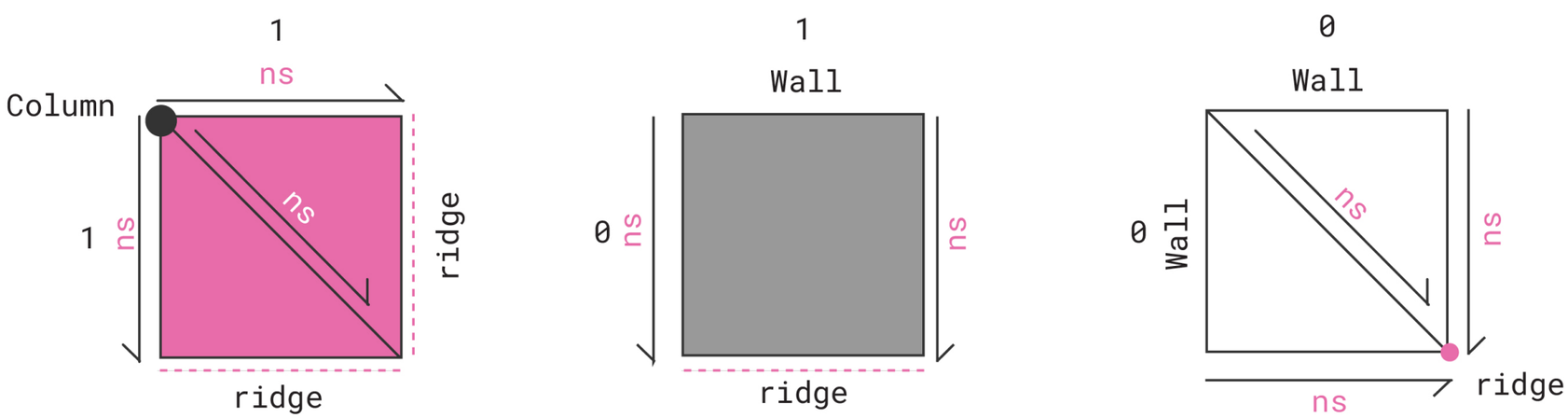


FIGURE 2.1.21: TOPOLOGICAL RELATION REQUIREMENTS PER MODULE

TOPOLOGICAL APPROACH

2.2 FROM PLAN TO SPACE RELATIONS

To comply with all the established conditions, a regular tessellation based on quadrilaterals has to be used. The distance conditions between the vertices are always the same amount of steps. (Figure 2.2.1)

To transform this 2d relation into a spatial consideration, the condition of the brick was taken into account with the followings steps (cording to figure 2.2.2):

1. Establish the grid.
2. Extrude the grid; bricks are created
3. Displace each brick vertically to the top of the previous step.
4. Extend each bricks to cover the previous step.
5. Extend each brick to cover the neighbor step
6. Displace laterally half of the bricks.
7. Create a half brick to have straight edges.

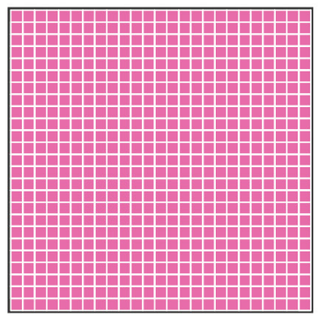


FIGURE 2.2.1: REGULAR TESSELLATION

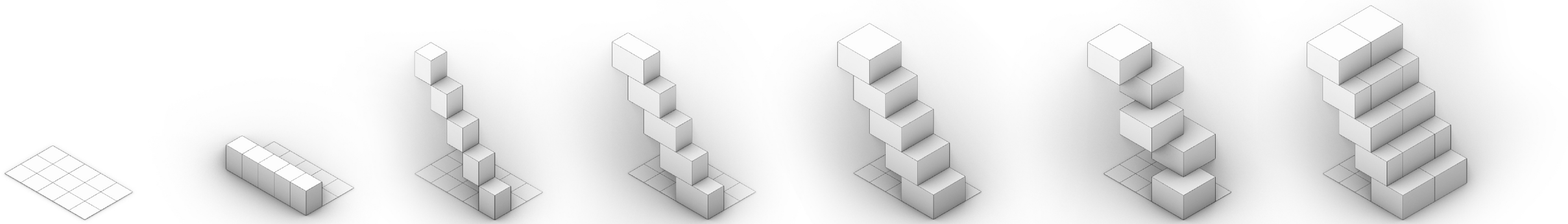


FIGURE 2.2.2: FROM A REGULAR TESSELLATION TO BRICKS TOPOLOGICAL BASIC APPROXIMATION

TOPOLOGICAL APPROACH

2.2 TESTING THE PIECES

To test the approach, the three pieces were created, as figure 2.2.4 shows. Because they were based on the restrictions, all the request conditions have complied.

To test in a module, type 3 was chosen because it needs the three pieces to be created. (figure 2.2.3)

This test demonstrates that the approach is valid. However, it is necessary to combine with a structural approach.

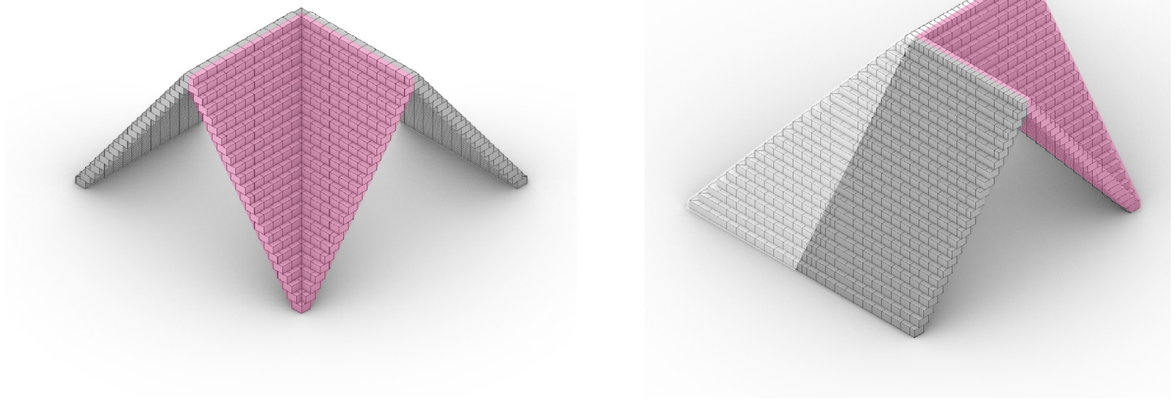


FIGURE 2.2.3: MODULE TYPE 3 BUILT BY TOPOLOGICAL APPROACH

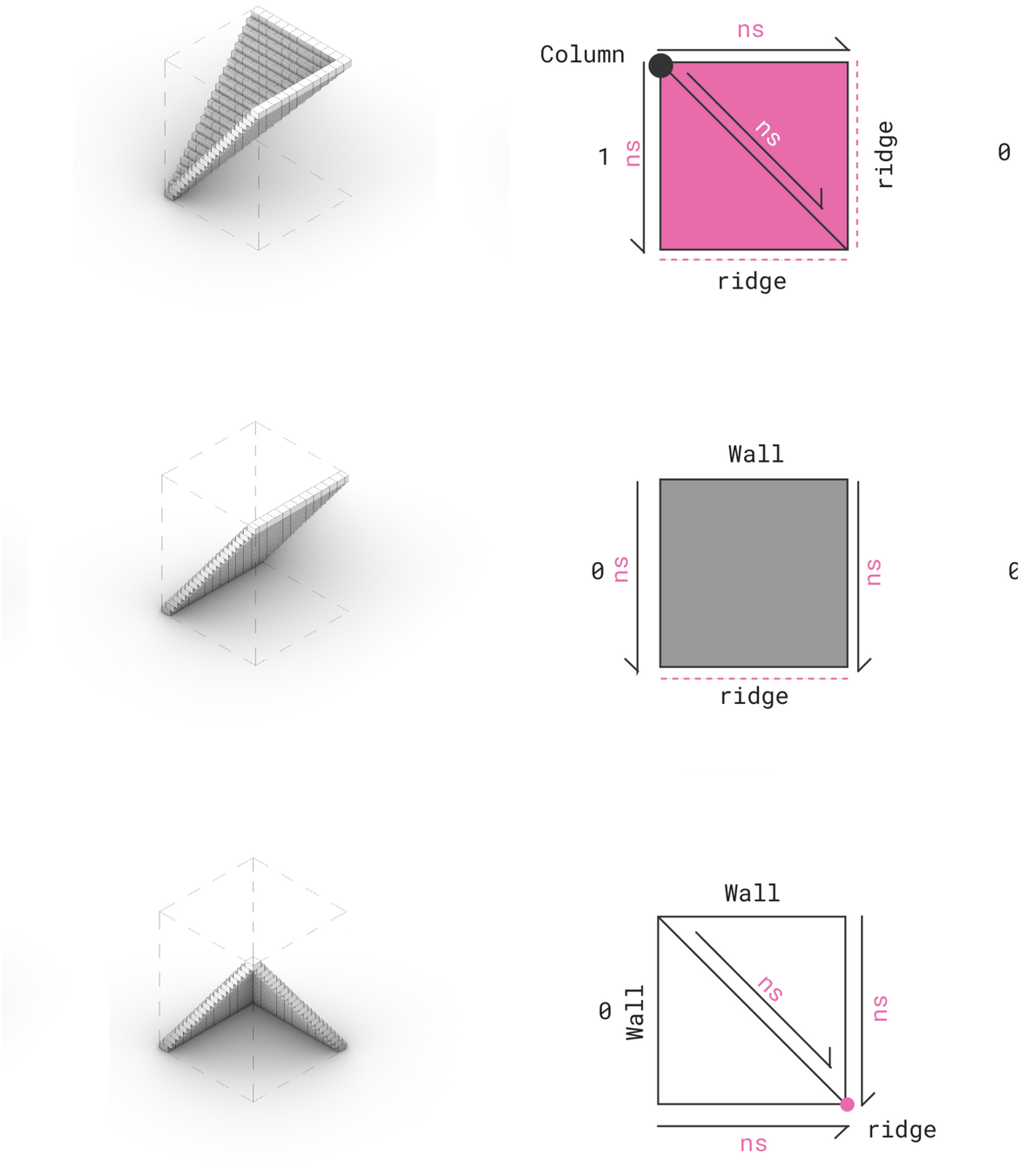


FIGURE 2.2.4: MODULES PARTS OF TOPOLOGICAL APPROACH

TOPOLOGICAL APPROACH

2.2 COMBINING WITH DYNAMIC RELAXATION

Given the almost zero tensile strength of adobe structures, dynamic relaxation was explored to obtain a relaxed mesh that can be approximated. Several restrictions were added to the process:

First, start with a regular tessellation (figure 2.2.5)
Second, relax the mesh with distributed loads in all the vertices, anchoring the corners (Figure 2.2.6)
Third, Keep the borders in vertical planes. Consequently, spatial continuity will be achieved. (Figure 2.2.7)
Fourth, keep coplanarity per step. Consequently, a similar height per step will be achieved but different between steps. (Figure 2..2.8)

With all these restrictions, an acceptable relaxed mesh to be approximate is archived. Nevertheless, as is shown in figure 2.2.4, the current topological approach is not a valid approximation.

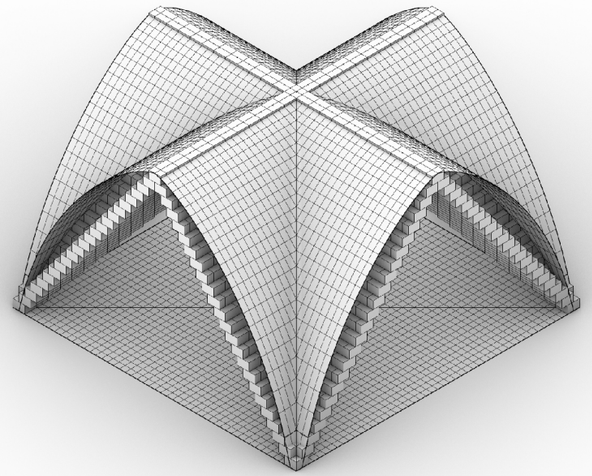


FIGURE 2.2.4: RELAXED MESH AND TOPOLOGICAL APPROACH WITHOUT FORM APPROXIMATION

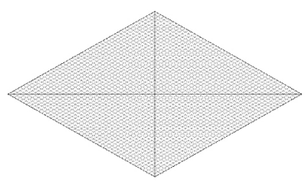


FIGURE 2.2.5: INITIAL MESH

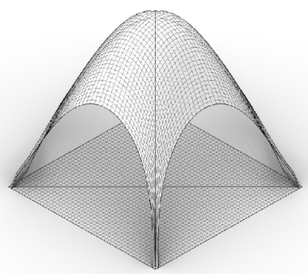


FIGURE 2.2.6: RELAXED MESH WITHOUT RESTRICTIONS

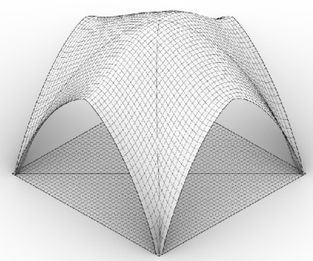


FIGURE 2.2.7: RELAXED MESH KEEPING OPENINGS IN VERTICALS PLANES

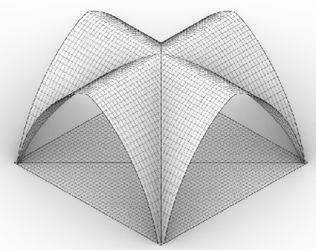


FIGURE 2.2.8: RELAXED MESH KEEPING OPENINGS IN VERTICALS PLANES AND LEVEL CO-PLANARITY

TOPOLOGICAL APPROACH

2.2 A VALID FORM APPROXIMATION

To achieve a valid form approximation, the step height was modified. Considering that the number of each step and their relations keep the same, we assume that the height change per level is a topological modification. Thus all the conditions and properties of the structure will be maintained.

To determine each level height, each step starts at the intersection of the external vertices with the mesh and end when the centroid intersection. Consequently, the thrust line is always inside the bricks. (figure 2.2.9)

This form is easy to achieve in terms of constructability because it is the same extrusion with different lengths. Thus, it is easy to accomplish in a compress brick system without changing the mode.

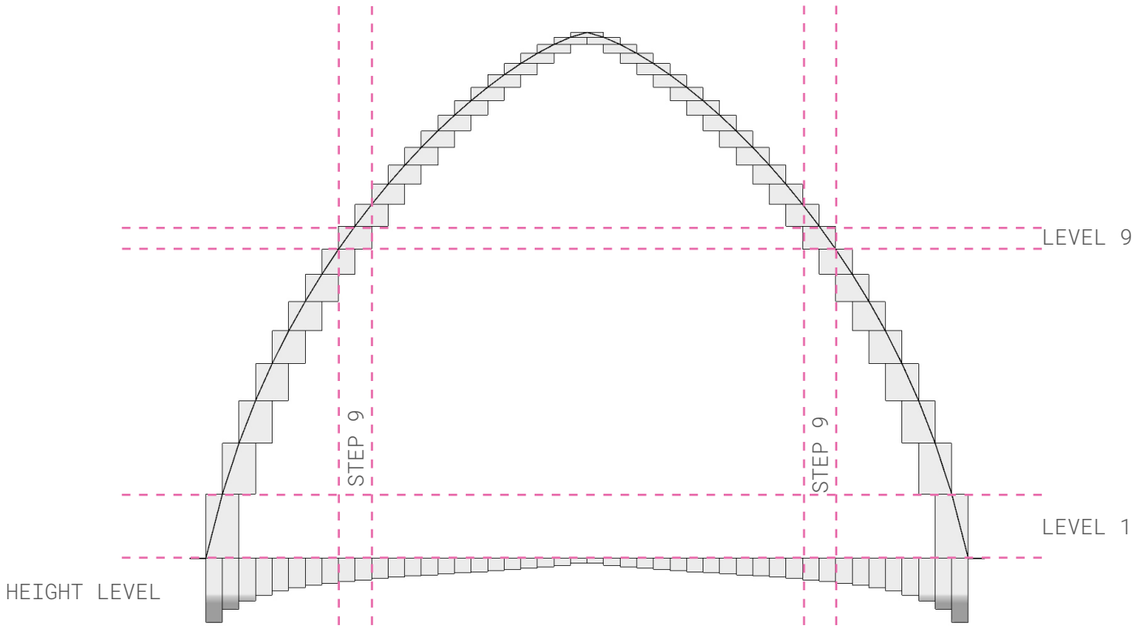


FIGURE 2.2.9: MESH FORM APPROXIMATED BY BRICKS WITH VARIABLE LEVEL HEIGHT

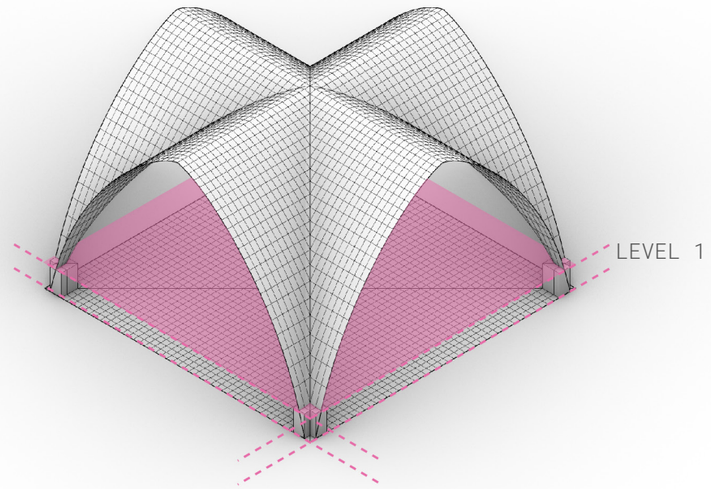


FIGURE 2.2.9: LEVEL 1 APPROXIMATION

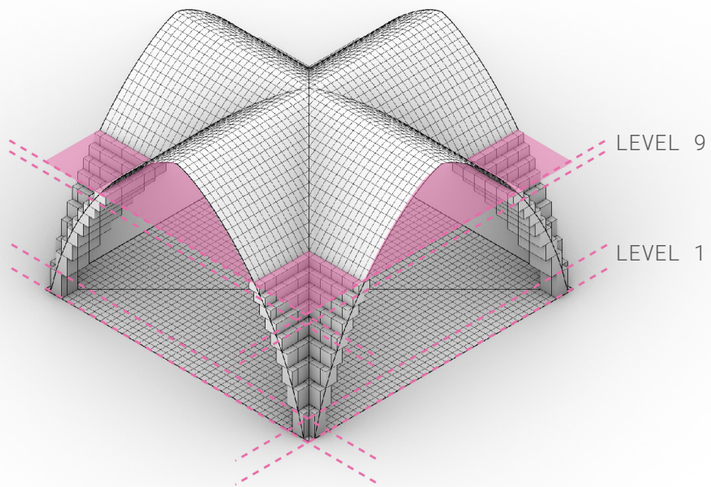


FIGURE 2.2.10: LEVEL 9 APPROXIMATION

TOPOLOGICAL APPROACH

2.2 A VALID FORM APPROXIMATION

With the described process, a valid approximation of the relaxed mesh was achieved with a brick structure.

Because the thrust line is coming to the border in each step (figure 2.2.11), a second brick-level was added (figure 2..2.13), keeping the thrust line always in the middle center of the structure.

A particular case is built for the modules that have the floor on top, shown in figure 2.2.12. In this case, all the bricks are modeled to test the structure considering all of them.

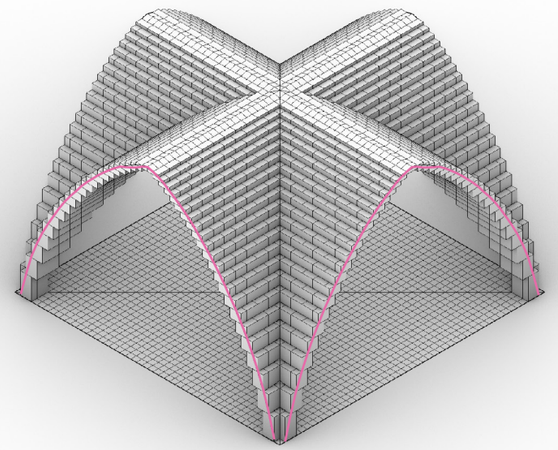


FIGURE 2.2.11: VAULT WITH SINGLE BRICK LAYER

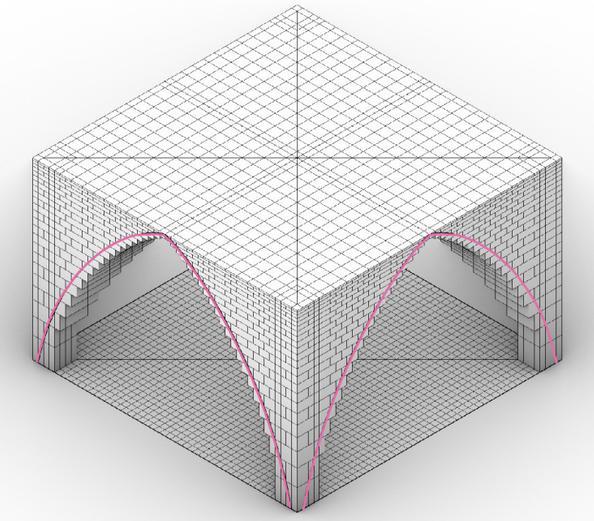


FIGURE 2.2.12: VAULT WITH FULL BRICK LAYER TO RECEIVE SECOND LEVEL

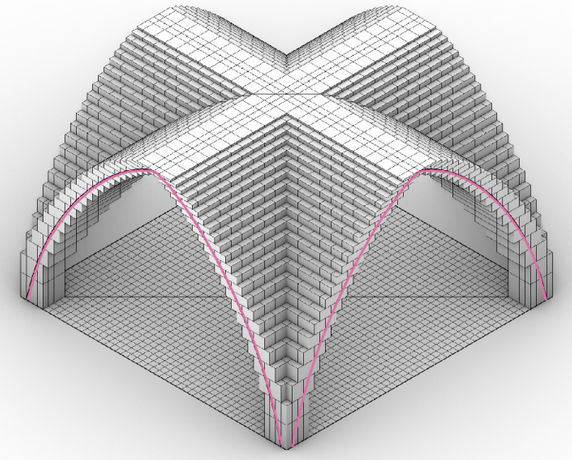


FIGURE 2.2.13: VAULT WITH DOUBLE BRICK LAYER

FINAL SHAPE

2.3 MODULES LIBRARY

Using the same topological approach, all the ground and second-floor modules were created (figure 2.3.2). Due to the established restrictions, all the requirements are achieved:

First, having spatial continuity.
Second, ceiling recognition of the local space.
Third, structural optimization for compression
Fourth, non-filling waste material.

Figure 2.3.1 shows all the modules and a combination where all the conditions can be appreciated.

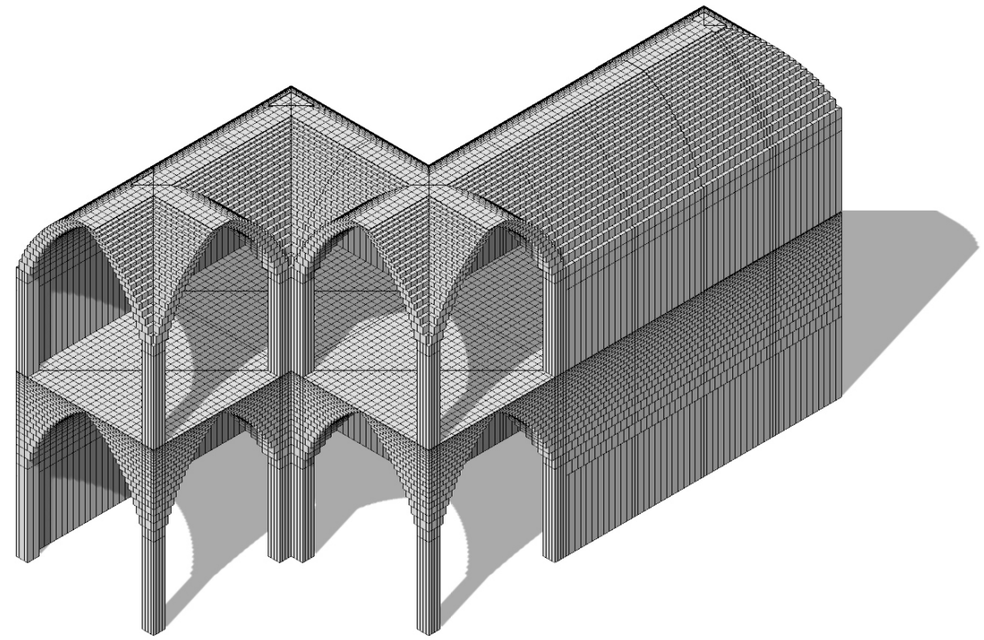


FIGURE 2.3.1: BASIC MODULES INTEGRATION

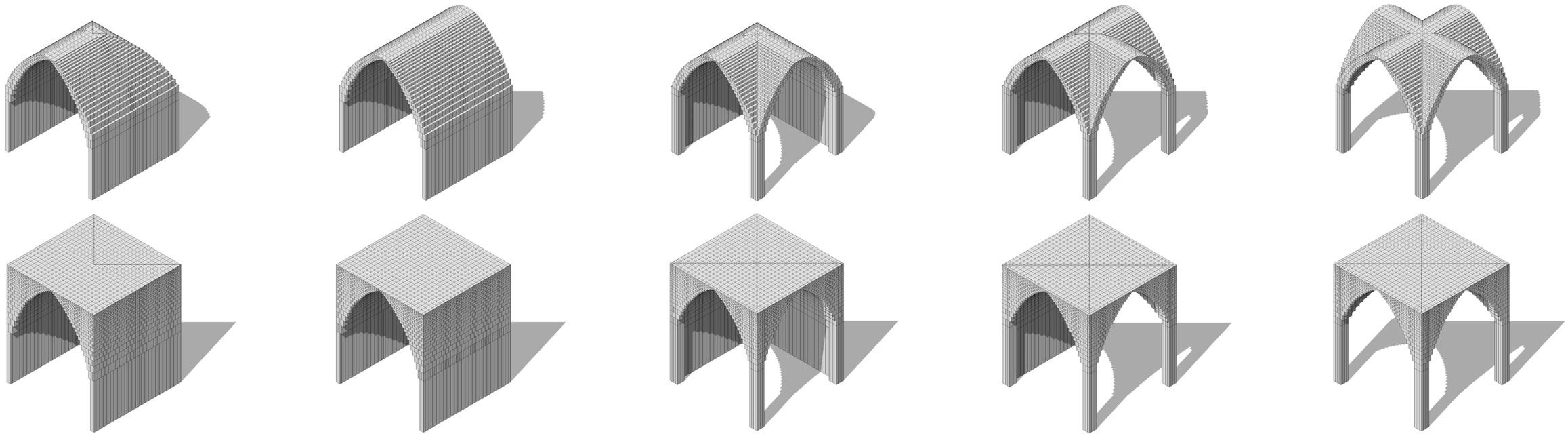


FIGURE 2.3.2: BASIC MODULES LIBRARY (EXCLUDING VARIANTS DUE TO BUTTRESSES)

FINAL SHAPE

2.3 SPECIAL MODULES

To complete the spatial configuration, some extra modules were needed, these modules are:

The stairs module (figure 2.3.4) connect the ground with the second floor allowing pedestrian connection to the centre of the block.

Local street roof (Figure 2.3.5). Because we want to have some of the streets covered, a double-height module was needed. Then the spatial connection from the ground and second-floor street don’t disappear.

Middle street roof (Figure 2.3.6). This particular module covers the intersection in the middle street, providing spatial continuity and allowing ventilation and light from the ceiling.

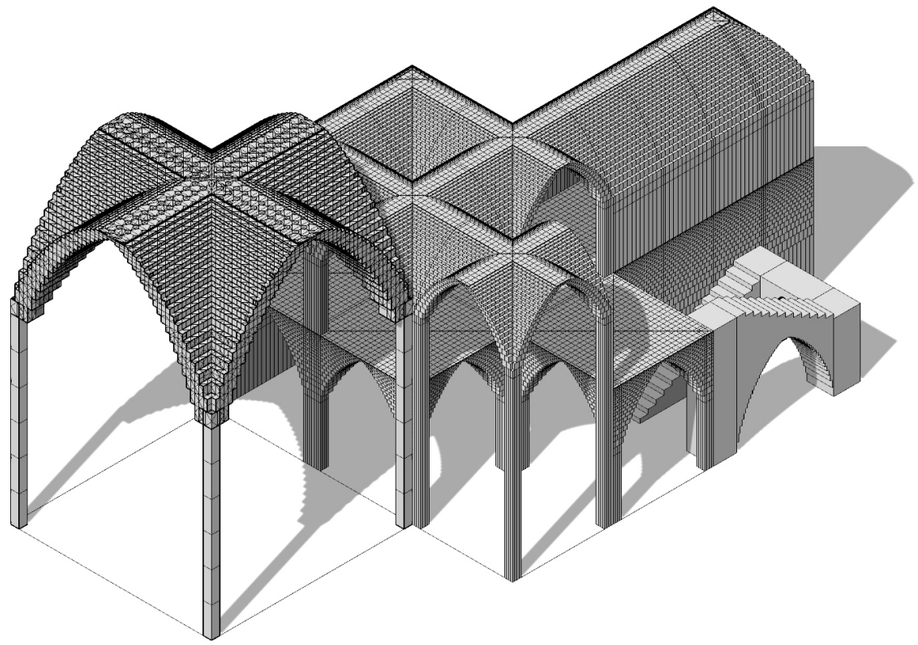


FIGURE 2.3.3: MODULES LIBRARY INTEGRATION

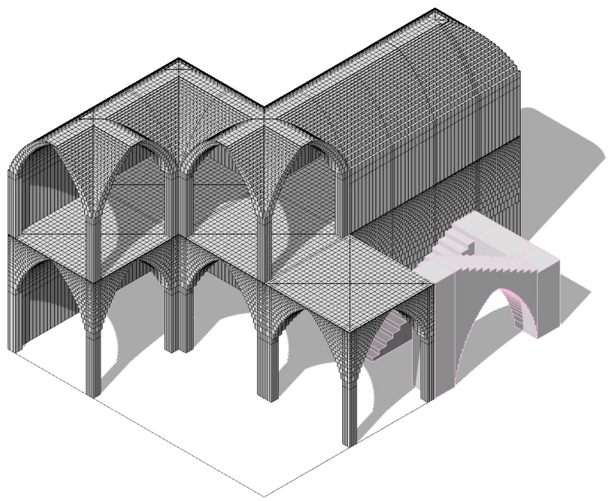


FIGURE 2.3.4: STAIRS MODULE INTEGRATION

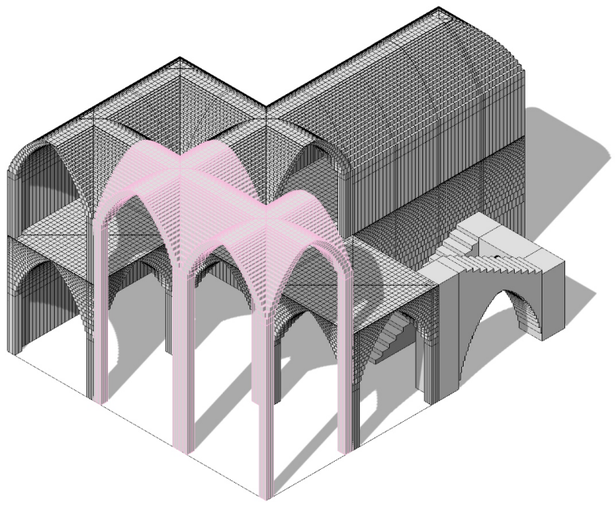


FIGURE 2.3.5: LOCAL STREET ROOF MODULE INTEGRATION

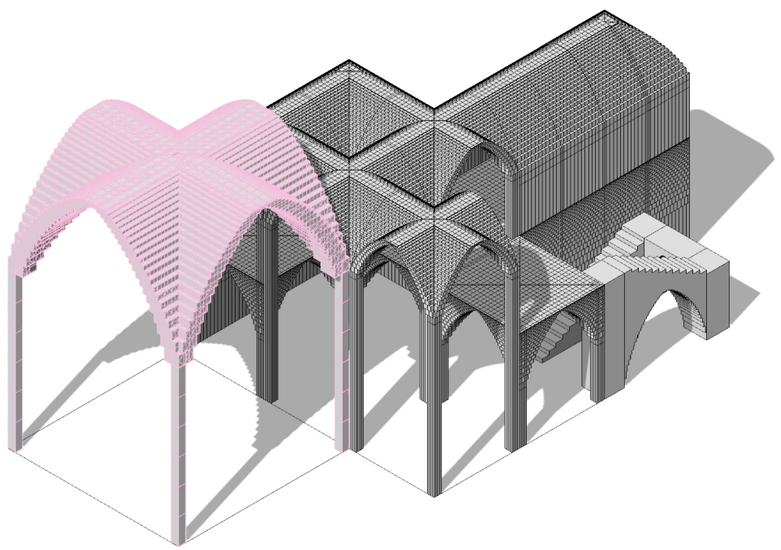


FIGURE 2.3.6: MIDDLE STREET ROOF MODULE INTEGRATION

Structuring 3.0

STRUCTURING

3.0 OVERVIEW

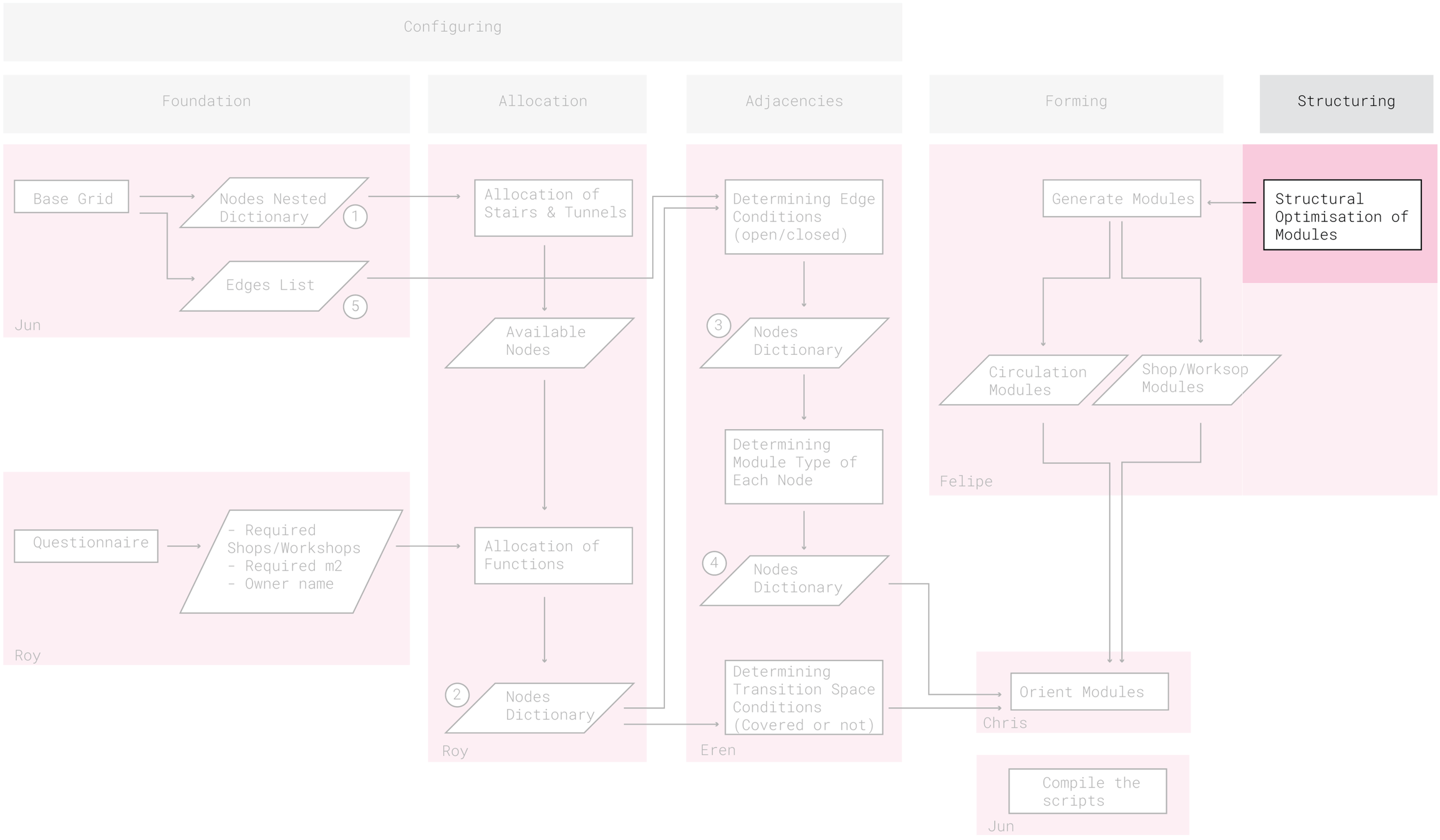


FIGURE 3.0.1. COMPLETE PROCCES FLOWCHART

STRUCTURAL ANALYSIS

3.0 DETERMINING THE CASE

To run the structural simulation and abstraction from the actual situation must be done.

This abstraction start chooses the worst-case scenario shown in figure 3.0.3, where the fewer material module (four openings) support, the more material module (1 opening).

To run the simulation, a simplification was developed with the following steps)(figure 3.0.3):

First, replace the lateral modules for reaction forces, assuming that the lateral loads of all the modules will be the same.
Second, Replace the second floor with vertical forces.
Third, replace the columns with reaction forces.

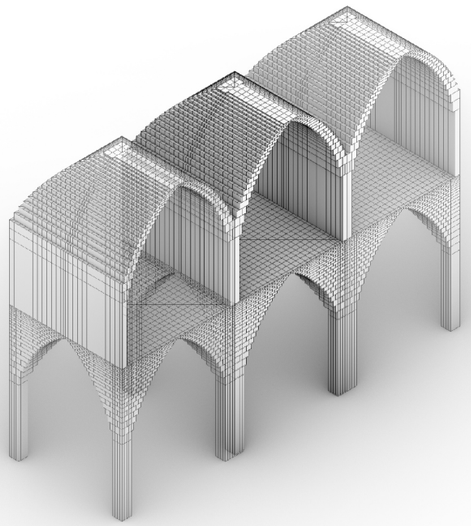
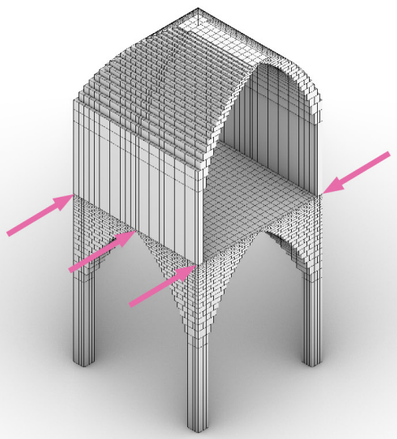
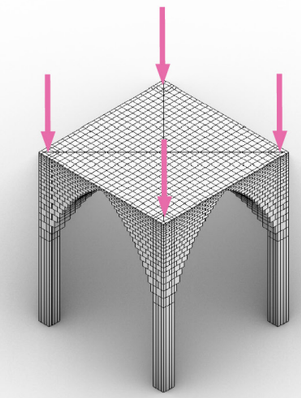


FIGURE 3.0.2: WORST CASE SCENARIO TO BE EVALUATED





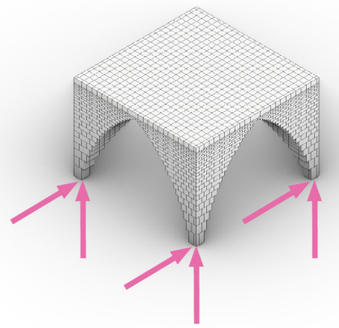


FIGURE 3.0.3: STRUCTURAL CASE ABSTRACTION

STRUCTURAL ANALYSIS

3.1 SHELL ANALYSIS

With the abstraction already done, a simulation-based on shell structure was done.

All the loads and support were configured and then obtained the maximum and minimum stresses shown in the table.

In this case, the bricks are only considered weight, and the shell thickness abstracted the bricks structure capacity.

The problem with this approach is the filling is missed as a structure. In this case, is considered a dead load, but in reality, are part of the structure

Variable	Value
Self Weight + Filling (kN)	89
Live Loads	45
Second Floor Weight	17
Thickness (m)	0.1
Max. Axial Compressive stress (kN/cm2)	1.9
Max. Axial Tensile stress (kN/cm2)	0.02
Maximum Displacement (m)	0.01 (0.23%)

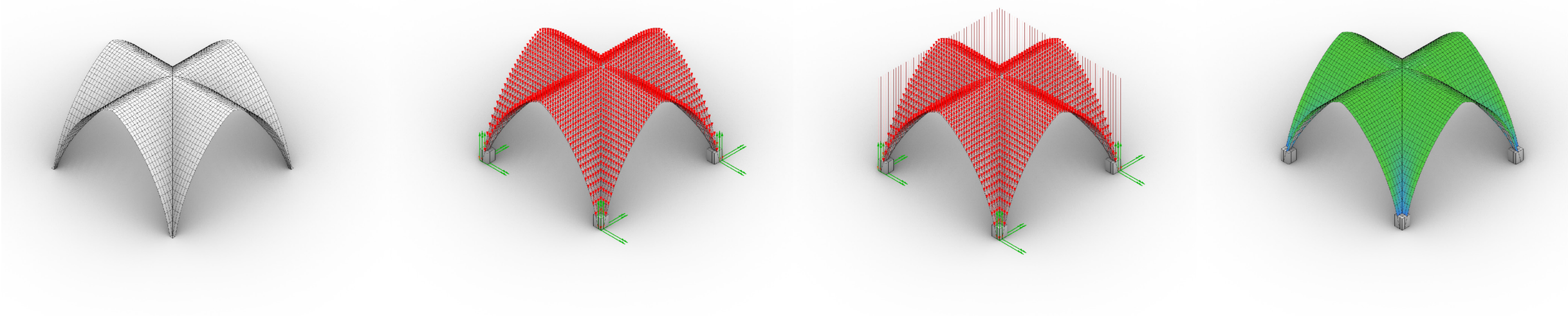


FIGURE 3.1.1: STRUCTURAL ANALYSIS BASE ON THE MESH APROXIMATION. TEST MESH / EQUAL DISTRIBUTED LOADS / UNIQUAL DISTRIBUTED LOADS / RESULTS

STRUCTURAL ANALYSIS

3.1 BRICKS STRUCTURAL ABSTRACTION

To find a valid brick structural approximation where all the bricks are an active part of the structure and the waste material does not exist, a domi model was developed.

This domi model consists of a slight arch, built in a shell structure (figure 3.1.2) and a brick structure (figure 3.1.3)

To be able to compare the same loads of 100 kN per point were appliedadded in both cases (figures 3.1.5 and 3.1.6)

The simulation runs in a known method for the shell model, and the table results are obtained.

For the bricks model, is no known method to abstract the bricks. Thus, we further explore the bricks abstraction

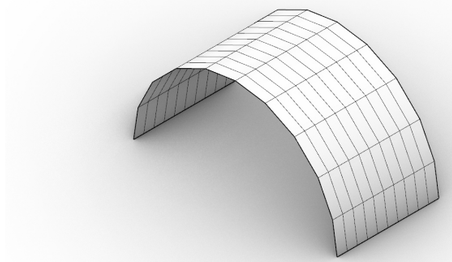


FIGURE 3.1.2: "DOMI" MESH ARCH

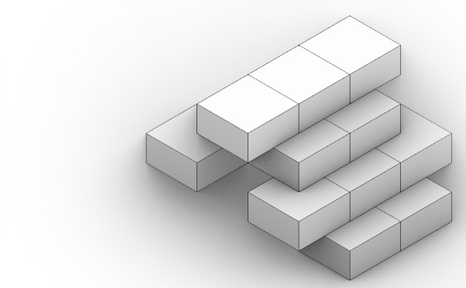


FIGURE 3.1.3: "DOMI" BRICKS ARCH

Variable	Value
Max. Axial Compressive Stress (kN/cm2)	1.9
Max. Axial Tensile Stress (kN/cm2)	0.03
Maximum Displacement (m)	0.0003

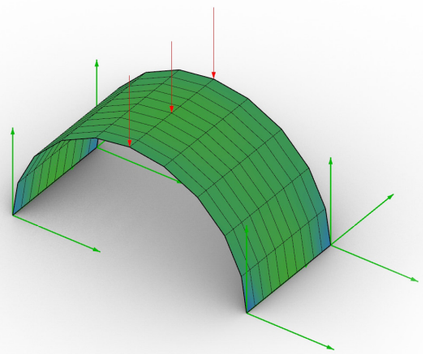


FIGURE 3.1.4: "DOMI" MESH ARCH RESULTS

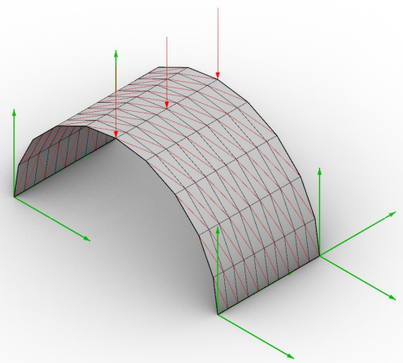


FIGURE 3.1.5: "DOMI" MESH ARCH LOAD CONDITION

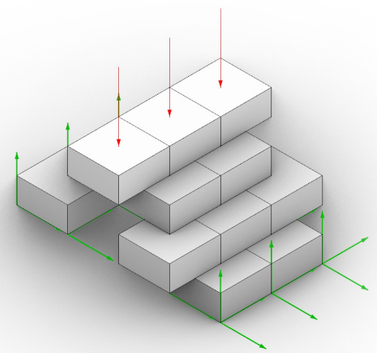


FIGURE 3.1.6: "DOMI" BRICKS ARCH LOAD CONDITION

STRUCTURAL ANALYSIS

3.1 BRICKS STRUCTURAL ABSTRACTION

To model the bricks based on linear elements linked by nodes, four types of abstraction were developed.

Type I consists of 4 linear elements connecting the centroids of the top face with the fourth vertices of the bottom front.

Type II: Same that type I with extra elements connecting the top face's centroids with the middle point of the edge of the top fece. Formin an spatial frame module

Type III: Same that type I with extra elements place on the edges of the bottom face. Forming a pyramid structure

Type IV: Consist of vertical elements in all the vertical edges and one that connects the centroids of the bottom and top face. Diagonal of the top and bottom faces and diagonals between opposite vertices in the top and bottom faces.

Type IV was created to have an option with vertical elements, where the load can flow directly from one brick to another when they are placed on top of each other.

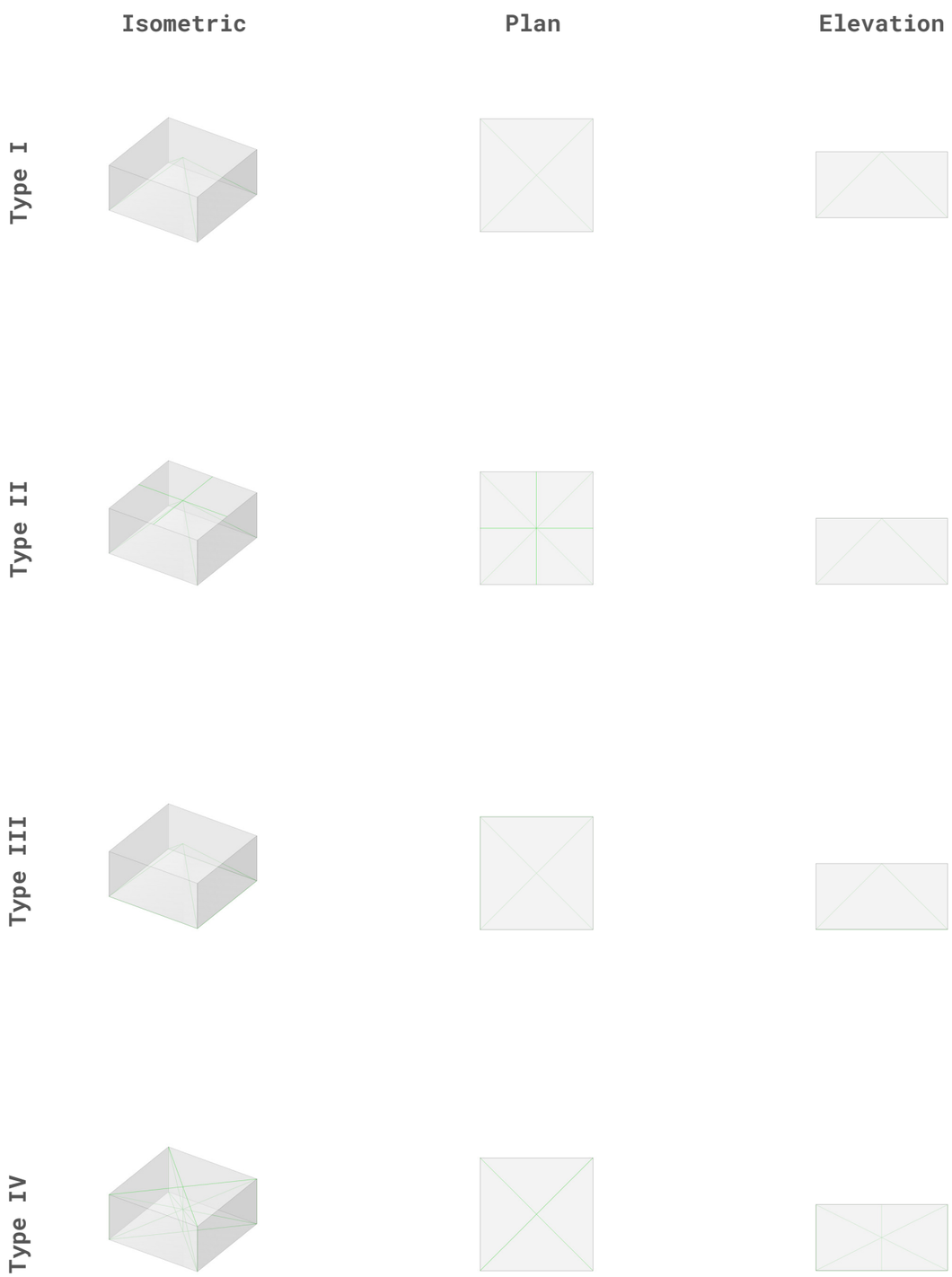


FIGURE 3.1.7: BRICKS ABSTRACTION INTO SPACIAL FRAME COMPONENTS

STRUCTURAL ANALYSIS

3.1 DOMI MODEL

To test the domi bricks model determining the cross-section of each element was fundamental.

In this case, the cross-section is determined by the following formula:

Cros Section area = Brick Volume / Total elements length.

Figure 3.1.8 shows the result of the tensions efforts present on the domi model.

The tension stresses are analyzed due to the abstraction amplifying existing tension stresses (mainly due to cantilever situations). The goal is to find the abstraction that had the smaller amplification of these stresses.

Variable	Type I	Type II	Type III	Type IV
Max. Axial Compressive stress (kN/cm2)	9.70	4.88	12.37	7.32
Max. Axial Tensile stress (kN/cm2)	7.87	3.50	11.88	6.85
Maximum Displacement (m)	0.17	0.29	0.17	0.04
Diameter (cm) (Circular cross section)	8.50	6.85	6.10	4.50

0 kN/cm2 — 11.88 kN/cm2

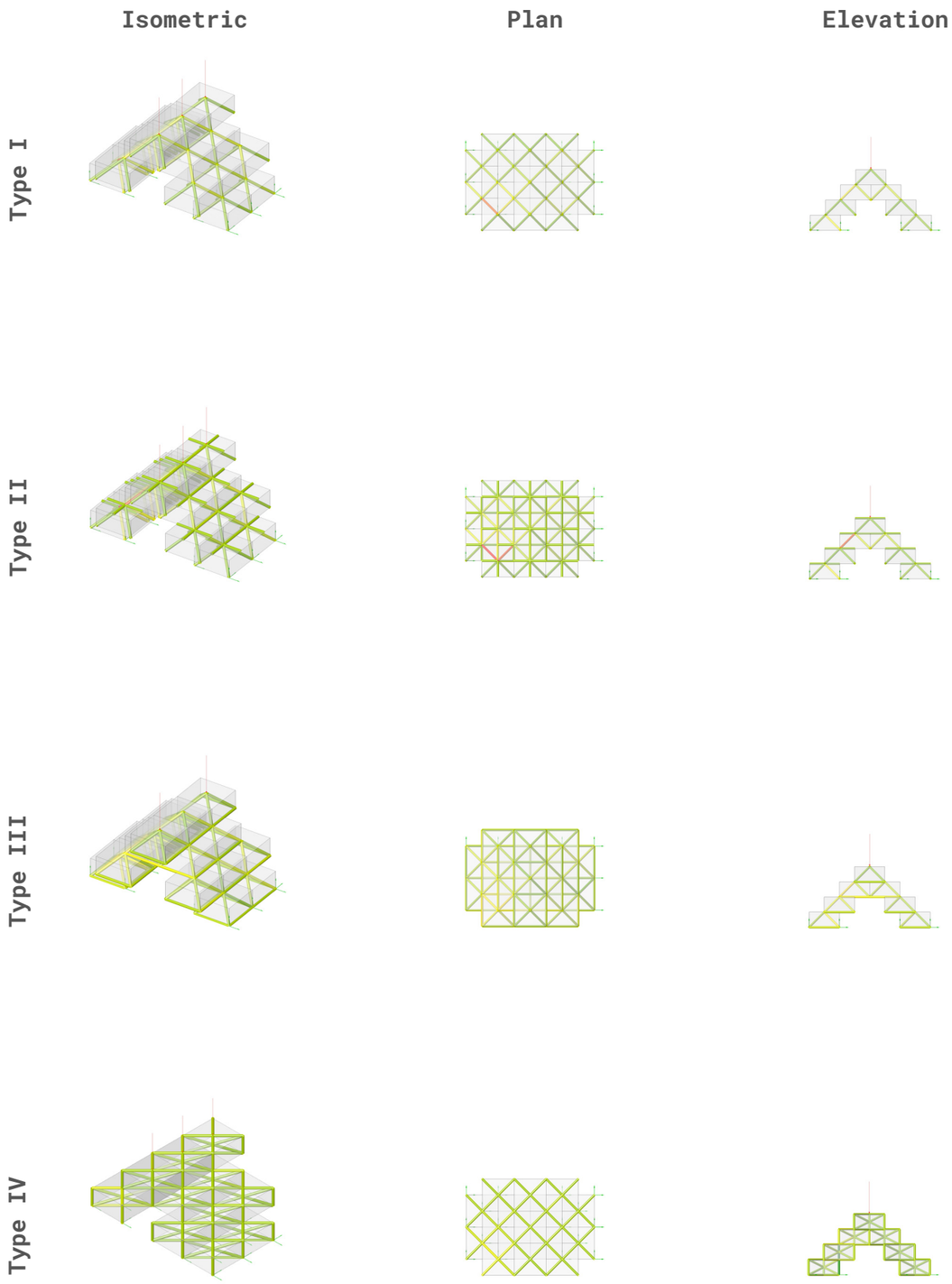


FIGURE 3.1.8: TENSILE STRESSES IN DIFFERENT BRICKS ABSTRACTION WITH VARIABLE CROSS SECTION

STRUCTURAL ANALYSIS

3.1 DOMI MODEL

Due to the strong influence on the cross-section into the results, a second simulation where the cross-section of all the abstractions is normalized to the higher one is shown in figure 3.1.9.

This analysis demonstrates that the better abstraction in a similar cross-section is TYPE IV due to the lower stresses and lower displacement.

This abstraction was taken into account to run the final simulation, with a factor of 1.8 in the cross-section area.

Variable	Type I	Type II	Type III	Type IV
Max. Axial Compressive stress (kN/cm2)	9.70	3.14	4.70	1.65
Max. Axial Tensile stress (kN/cm2)	7.87	2.24	4.37	1.15
Maximum Displacement (m)	0.17	0.13	0.065	0.01
Diameter (cm) (Circular cross section)	8.50	8.50	8.50	8.50

0 kN/cm2 7.87 kN/cm2

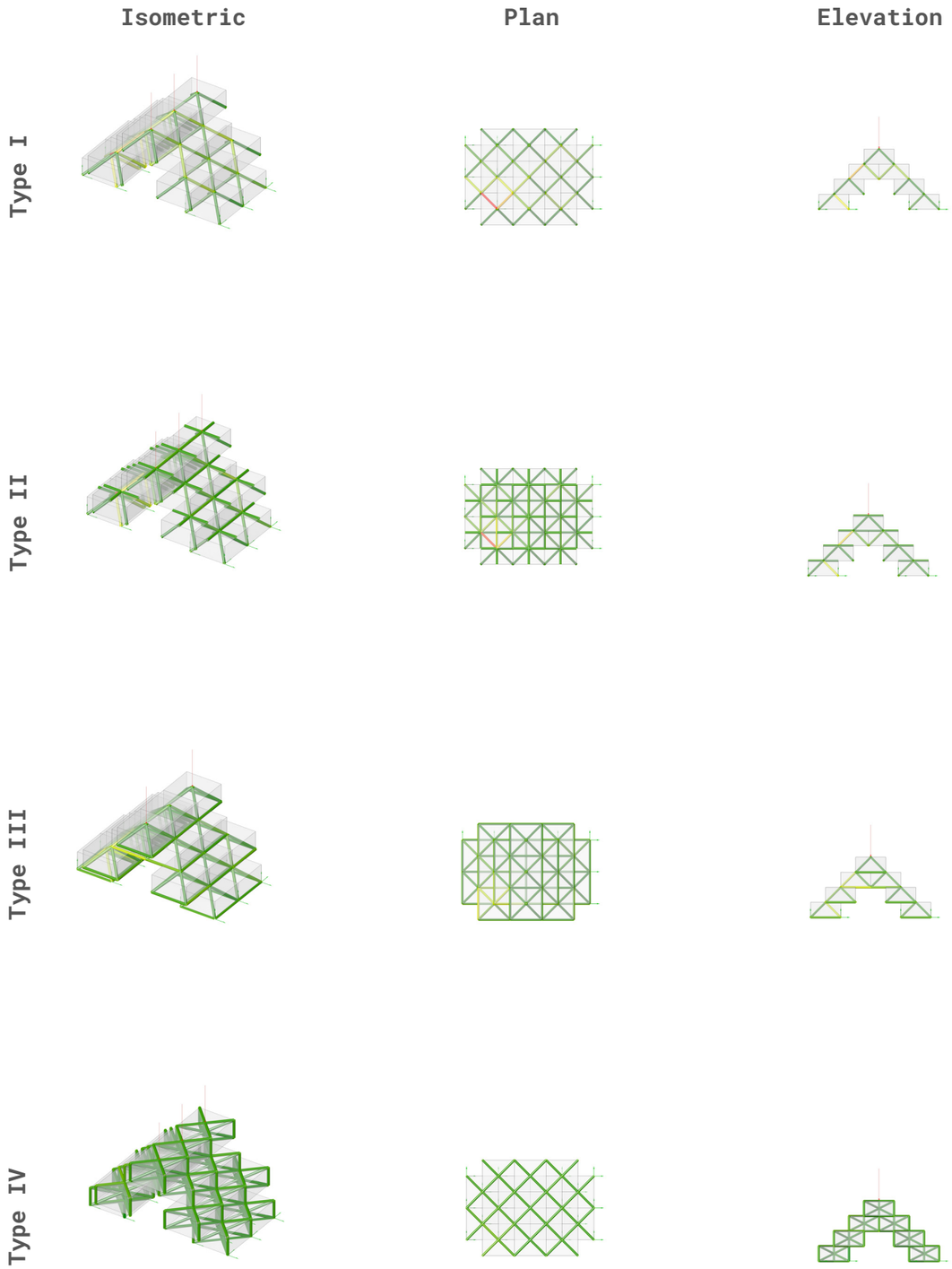


FIGURE 3.1.9 TENSILE STRESSES IN DIFFERENT BRICKS ABSTRACTION WITH FIX CROSS SECTION

STRUCTURAL ANALYSIS

3.1 SITE ANALYSIS

To run the final analysis case, figure 3.1.10 was taken into account, adding distributed load en each node of the structure (figure 3.1.11), live load on the top surface applied to the centroid of each brick (figure 3.1.12) and second-floor weight applied to each contact brick centroid (figure 3.1.13)

For all the cases, the total load were divided by the number of nodes to apply, and then a punctual load was applied in the corresponding nodes.

A deviation of this approach is that the weight of the bricks is distributed in all nodes with the same values disrupting the reality where the bottom ones are bigger.

Load Case	Load Type	kN
0	Self Weight	89
1	Live Loads	45
2	Second Floor Weight	17
Total	All Loads	151

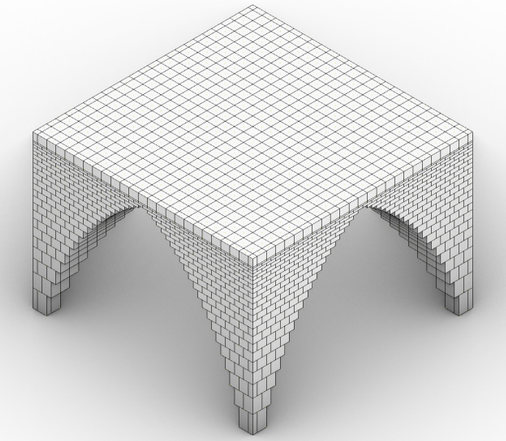
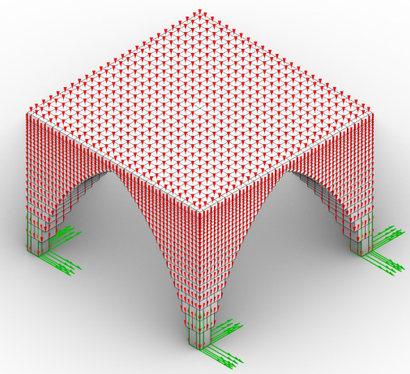
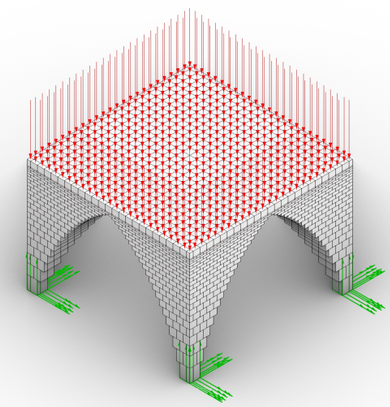


FIGURE 3.1.10: STRUCTURE TO BE TESTED WITH ALL THE BRICKS RECOGNIZABLE



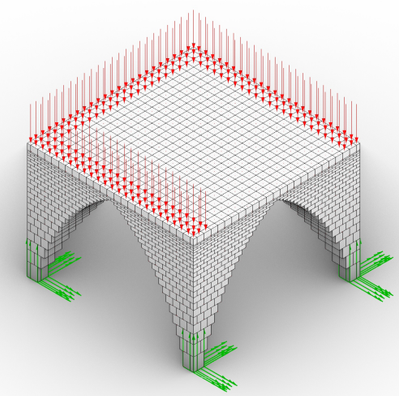
89 kN (6.08m3X14.71 kN/m3)

FIGURE 3.1.11: LOAD CASE 0



17 kN (1.2m3X14.71 kN/m3)

FIGURE 3.1.12: LOAD CASE 1



45 kN (9m2x5 kN/m2)

FIGURE 3.1.13: LOAD CASE 2

STRUCTURAL ANALYSIS

3.1 FINAL SIMULATION

After running the final simulations, a particular visualization was configured.

Because linear elements are used, tension and compression will be present in the majority of the elements. Thus, each element is taken as a line and coloured in the tension scale. (figure 3.1.14)

Because each element is recognizable was possible to isolate the ones that are out of the limits of 0.02 kN/cm2 (figure 3.1.15)

After the distribution of the elements out of the limits was studied (Figure 3.1.16), demonstrating that only a few components raise the maximum stresses and the majority is close to the limits.

Also, the maximum tensions are in the base and not in the point where the load was applied, indicating that the model is working properly.

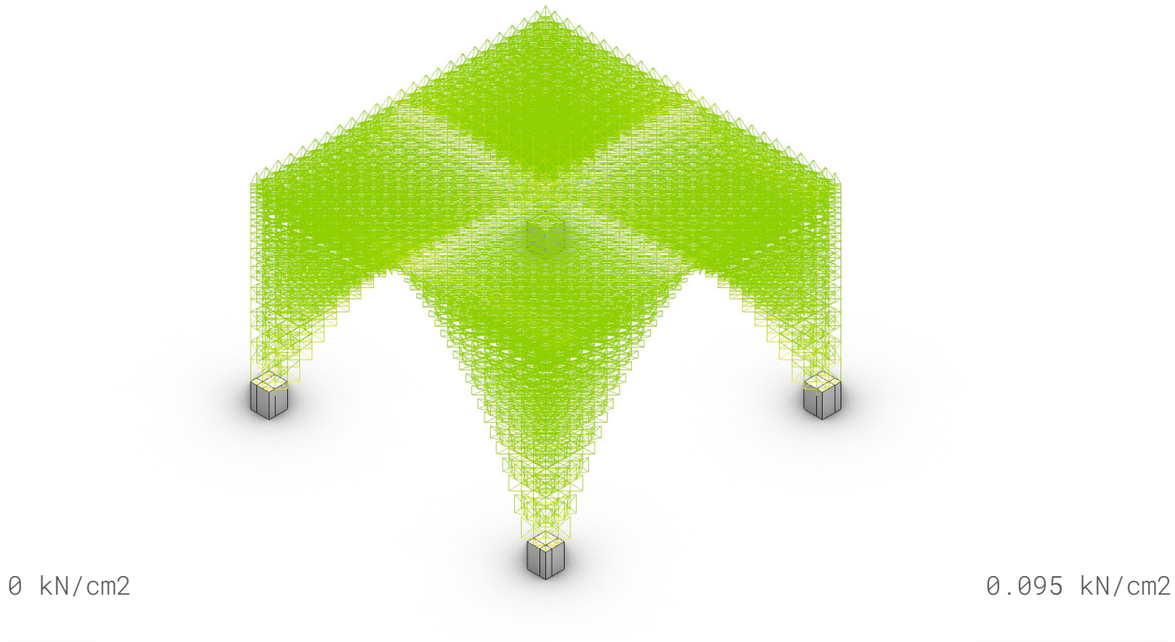


FIGURE 3.1.14: TENSILE STRESSES PER ELEMENT

Variable	Value
Max. Axial Compressive stress (kN/cm2)	0.128
Max. Axial Tensile stress (kN/cm2)	0.095
Maximum Displacement (m)	0.31 (0.7%)

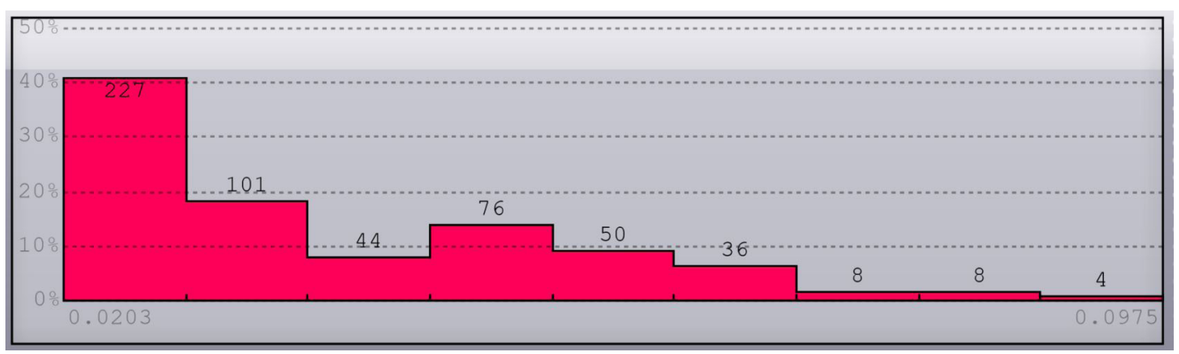


FIGURE 3.1.16: DISTRIBUTION OF ELEMENTS OUT OF RANGE OF TENSILE STRESS (MORE THAN 0.02 KN/CM2)

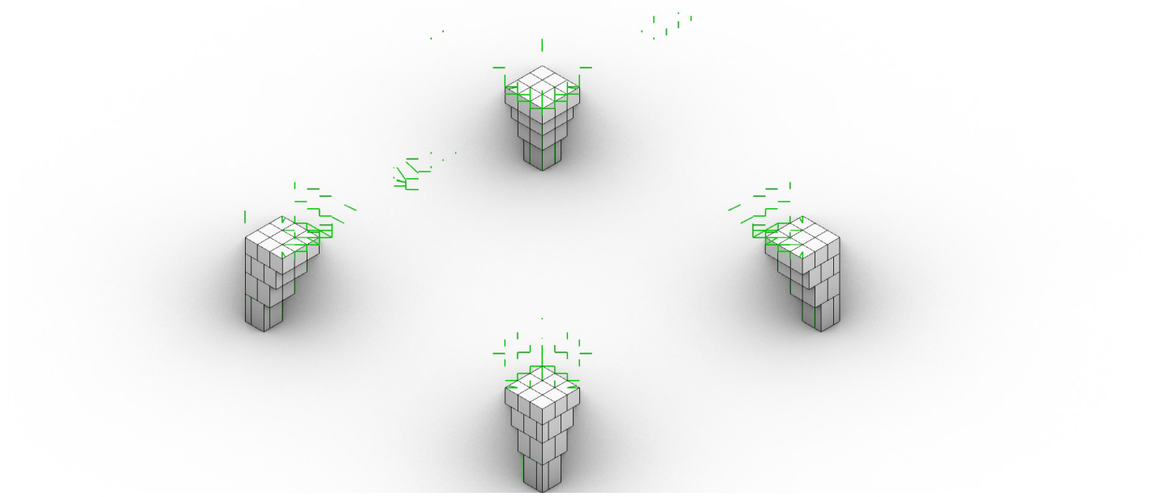


FIGURE 3.1.15: ELEMENTS OUT OF RANGE OF TENSILE STRESS (MORE THAN 0.02 KN/CM2)

STRUCTURAL ANALYSIS

3.1 FINAL SIMULATION

In this case, the axial compression stresses were visualized for all the elements. Thus, for the compression, all the elements are inside the limits of -0.2 kN/cm2, as figure 3.1.17 shows.

To understand the behaviour of the structure, the elements with compression higher than 1MP were isolated (Figure 3.1.18). This analysis shows the evident influence of the second-floor load transmitted by the walls. Also, it demonstrated that the shell analysis misses out on most of these elements in the “Filling waste” and are those with higher requirements.

Figure 3.1.19 shows the compression distributions demonstrating that are few elements with higher stresses.

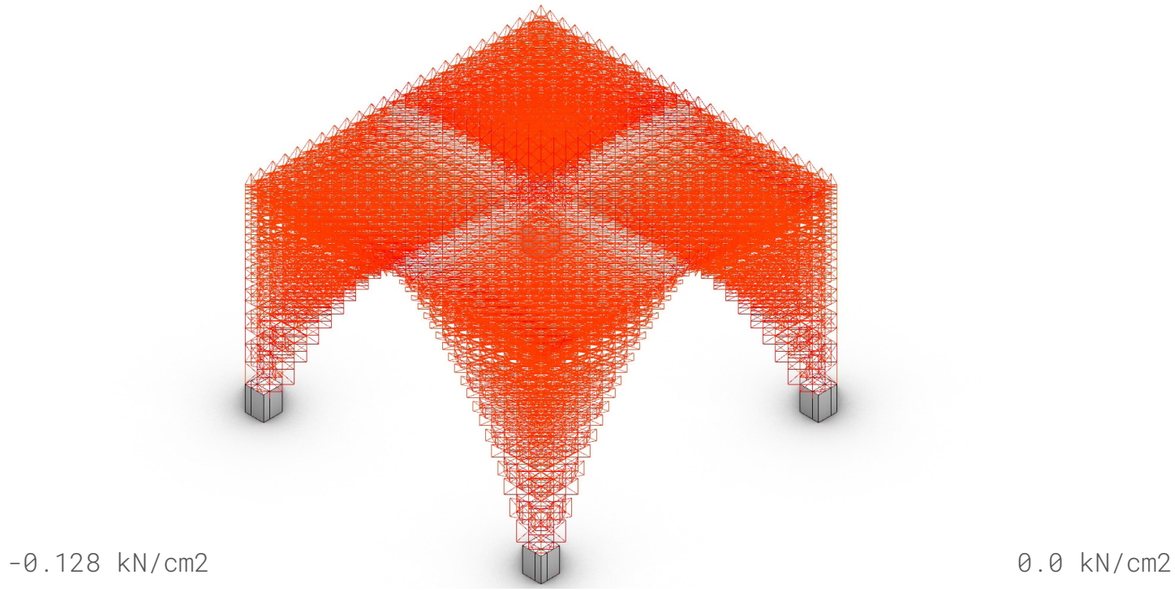


FIGURE 3.1.17: TENSILE STRESSES PER ELEMENT

Variable	Value
Max. Axial Compressive stress (kN/cm2)	0.128
Max. Axial Tensile stress (kN/cm2)	0.095
Maximum Displacement (m)	0.31 (0.7%)

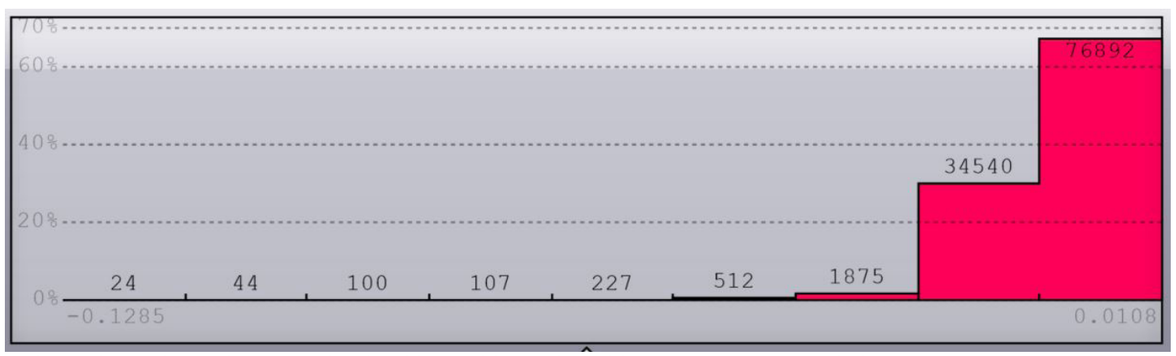


FIGURE 3.1.19: DISTRIBUTION OF COMPRESSIVE STRESS PER ELEMENT KN/CM2

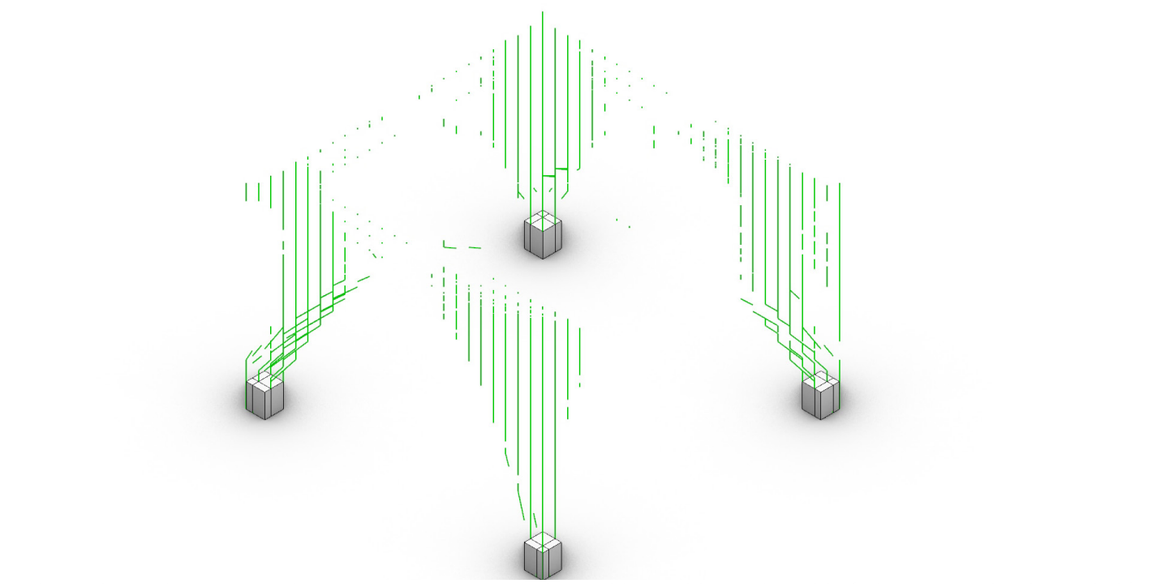


FIGURE 3.1.18: ELEMENTS THAT EXPERIMENT HIGHER COMPRESSIVE STRESS (MORE THAN 0.10 KN/CM2)

CONSTRUCTION PROCESS

2.3 BRICK BY BRICK

The constructed model is a visualization of its own to understand the different steps that are necessary to build its form at a 1:50 scale on situ. A few supports are required to allow the voxelated flat bricks to become a Gothic vault. The 4 different model types were chosen because of their open and closed edge condition to prove its tessellation with one to another and its structural stability by placing the heaviest module type on the second floor.

FIG 2.3. | FINAL MAQUETTE 1:50

FIG 2.3.1 | FOUNDATION

FIG 2.3.2 | ASSEMBLY SUPPORT

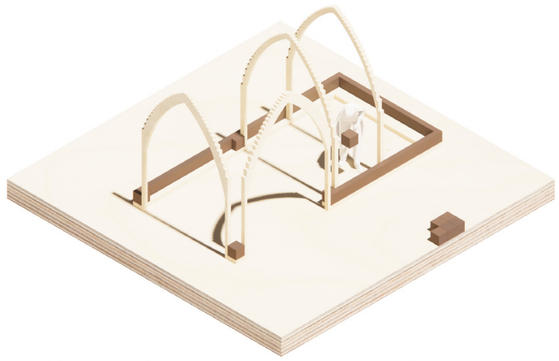


FIG 2.3.3 | PLACING BRICK

FIG 2.3.4 | STACKING EFFECT

VISUALIZATION

VISUALIZATION

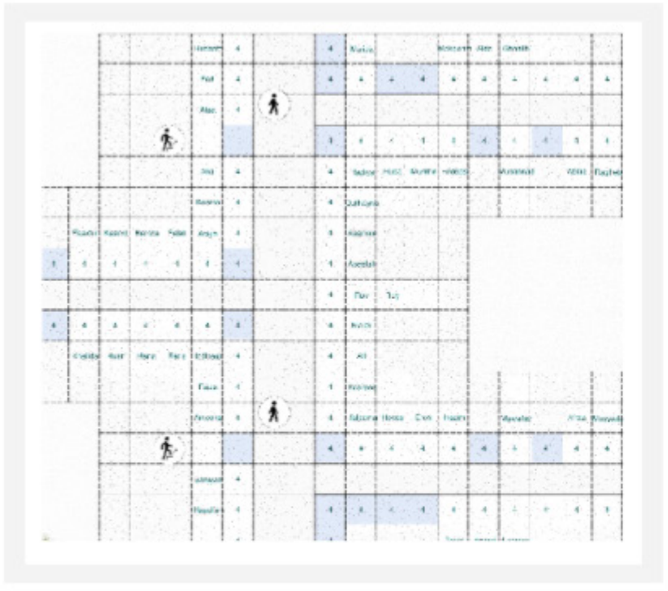
Reflection 4.0

GENERAL REFLECTION

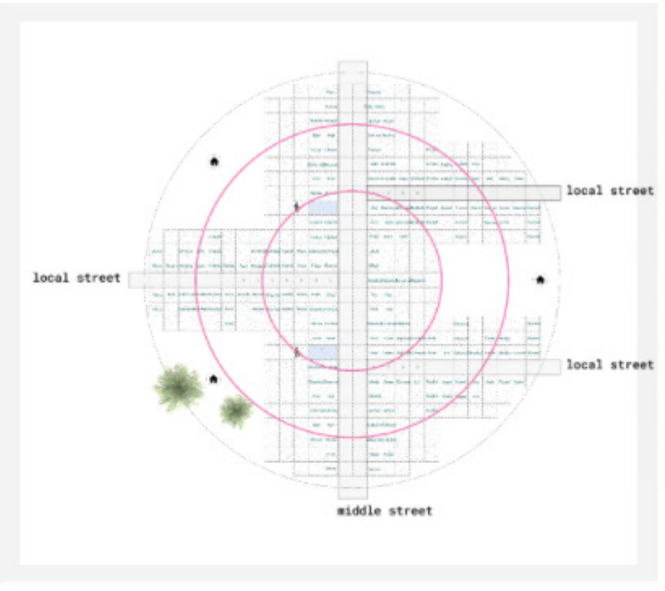
Overall, as a group we managed to realise our design decisions and follow the methodology we decided that is the most suitable. Even though at the end we designed a bazaar with the requirements and set of rules we determined, there are aspects to improve. Firstly, there are some minor issues about the module types on the corners as those cannot simply be determined in relation to their neighbors. We needed to change some modules manually to make every shop/workshop/both accessible. Secondly, we aimed to place the staircase modules depending on the distance from the intersection and repeat the staircases every 30 meters (or another fixed value) . However, since the location of the stairs is not the main goal of Co[nn]Action, we decided to locate them manually to save some time. In addition, one of the goals of this project is to provide communal spaces such as courtyards. Since we were mainly focused on the bazaar afterwards, methodologically locating the courtyards or other communal spaces became an afterthought.

If we further developed this project, communal spaces would need to be designed. Another point is, as a group we envisioned an organic bazaar in terms of variety of shops and their sizes however due to some technical problems the bazaar we designed is not as diverse as we imagined. Further developments should be done to make the dwellings more diverse in shape and size either by incorporating more questionnaire data or changing some parts of the script. One of the decisions we took at the beginning is to make our method as modular as possible and applicable to multiple scenarios, therefore module designs can be changed if desired. For instance, intersection modules can be designed differently to allow light from above. From a more technical point of view, construction technique can be studied more in order to make it easier to construct as well as to make the framework more realistic. Last but not the least, on the structural side the method we followed to do the abstraction of the element and their cross sections should be validated by further research and investigation. In conclusion, naturally there are topics to be further studied given the time limit and technical limitations we had. However, we believe that Co[nn]action became a project fitting to our problem statements and suggested solutions by utilising a methodological workflow.

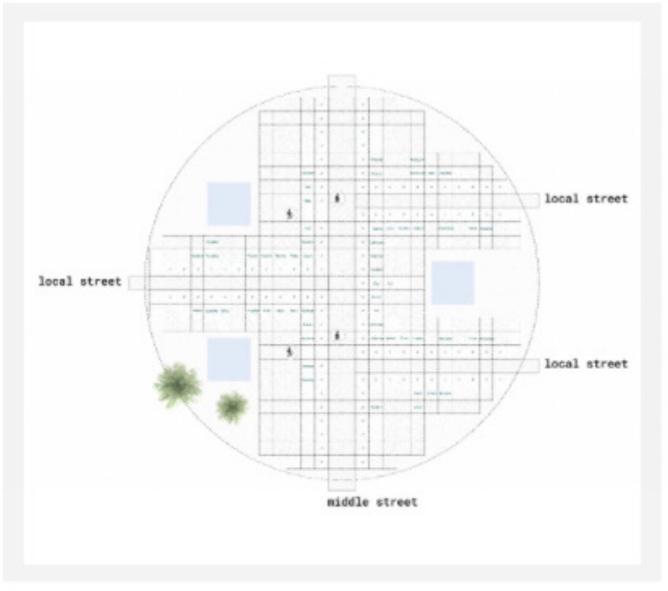
GENERAL REFLECTION



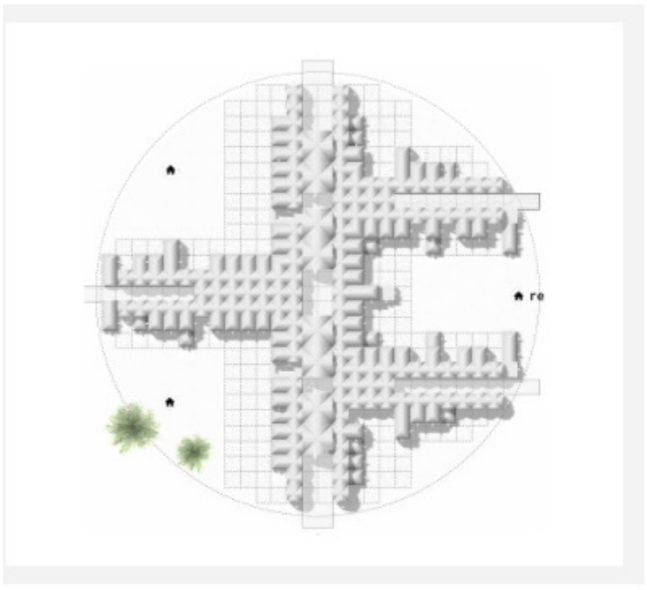
EXCEPTIONS : WALKABLE SHOP MODULES ON THE CORNERS



REPEATING STAIRCASES EVERY #... NODES



GENERATION OF COURTYARDS OUT OF THE BAZAAR



DIVERSITY IN CONFIGURATION OF SPACES

MORE REALISTIC STUDY ABOUT THE USE OF FRAMEWORKS.

VALIDATE THE BRICK ABSTRACTION AND ELEMENTS CROSS SECTIONS

INDIVIDUAL REFLECTIONS

JUN WEN - BLENDER

This course has been enriching for me as we had a deep dive into the application of python across many different platforms and libraries. Particularly I've learnt tremendously from helping to debug the scripts within the team as it is exciting to see that my group members equally share the enthusiasm to push the project computationally. Furthermore, the study into topological relations and abstracting our spatial allocation problems into graphs and adjacencies has opened my eyes to the myriad possibilities that could arise from this approach. It certainly helped our team achieve the result we had intended, although much more improvements are still required. I would like to thank Pirouz and Shervin, together with Frank, Charalampos and Hans for your feedback. It has been an intense but enriching journey.

ROY - PONDERER

After the design approach was decided, we divided the tasks. My design task was to cover the input from the questionnaire and allocate the requests on the site. The data that was filled in in the questionnaire was directly converted into a excel sheet. From here on I only used Python scripts to retrieve, manipulate, and send data. This was completely new to me and became one big learning process. The first few weeks I already spent a lot of evenings with learning the python basics (I didn't even know the difference between a string and an integer). I really liked the process and the new way of thinking, using logical approaches and rules to develop a script. As I developed more and increased my coding knowledge, I kept improving the codes I already written to make them shorter and faster, this way I spend a lot of time on the code. However, I also encountered many difficulties. The occupation of the requests, following all the rules was one massive puzzle. It took a lot of time and a lot of help from Jun, but in the end I managed to complete the allocation. In conclusion, our methodology and logic worked out well, the allocation of requests worked, but there are some aspects to improve and change slightly for the sake of a cleaner result.

EREN - MAGICIAN

After deciding on how to proceed and what to design, my task was to assess adjacencies and assign modules to the nodes. For the Adjacencies & Modules Stage, only Python is used with some libraries such as networkx, itertools. Even though this was the first time I used Python and it was a learning process, at the end the code worked as we wanted it to do. Creating rules and logical approaches to work with instead of drawing lines freely was definitely a new experience. When it comes to the code, it changed many times during the developing stage. After understanding how to use networkx and graphs to their full potential the script itself got shorter significantly. There are some aspects to improve or some parts which do not work perfectly of course, given this was a learning experience. First aspect is the naming of the modules, we could have used integers to name the modules instead of floats which we decided to use for the ease of application from our side.

Secondly, some modules on the ground floor are switched to walkable modules on top on the local street. Method for this was if the upper floor walk path has shop/workshop/shop&workshop neighbor than the module below the walk path should be filled. However, after bringing the nodes data to grasshopper in Orientation stage, we realized that there were some islands created due to the fact that corners only have walk-path neighbors and sometimes there are empty nodes between shops on the upper level. Lastly, assigning modules to middle streets and assigning intersection modules were done manually with a logic behind due to the necessity of coming up with too many exceptions. However, if we had more time, a streamline methodology could be applied. In conclusion, indeed our methodology and logic worked well to assign modules but there are some aspects to improve and change slightly for the sake of a cleaner work flow.

INDIVIDUAL REFLECTIONS

CHRIS - SCULPTOR/MAGICIAN

My main task for the group was to orient the modules to their assigned location onto a base grid in grasshopper. I first had to understand the python script to gather the important information necessary to be transferred to a visual platform. I was fascinated by how a programming language can be a way to help the design process. It has shown how an open source language such as python can be manipulated by the users logic to design rules and establish a modular design. The grasshopper script was first used as a dummy to interpret and evaluate the output of the module types placed and oriented in the correct logic. The visualization of the module types assigned to their allocation on the base grid was successful. However, the fixed conditions such as the bridges and staircases were placed and oriented manually as we did not have enough time to implement them in our code. In conclusion, our workflow and logic was well structured. Each student had an individual task which was a success once put all together. The studio's logic on modularity could become the story of a great success by helping communities to use safer paths in the camp and offer the inhabitants an active and connected neighbourhood designed by themselves.

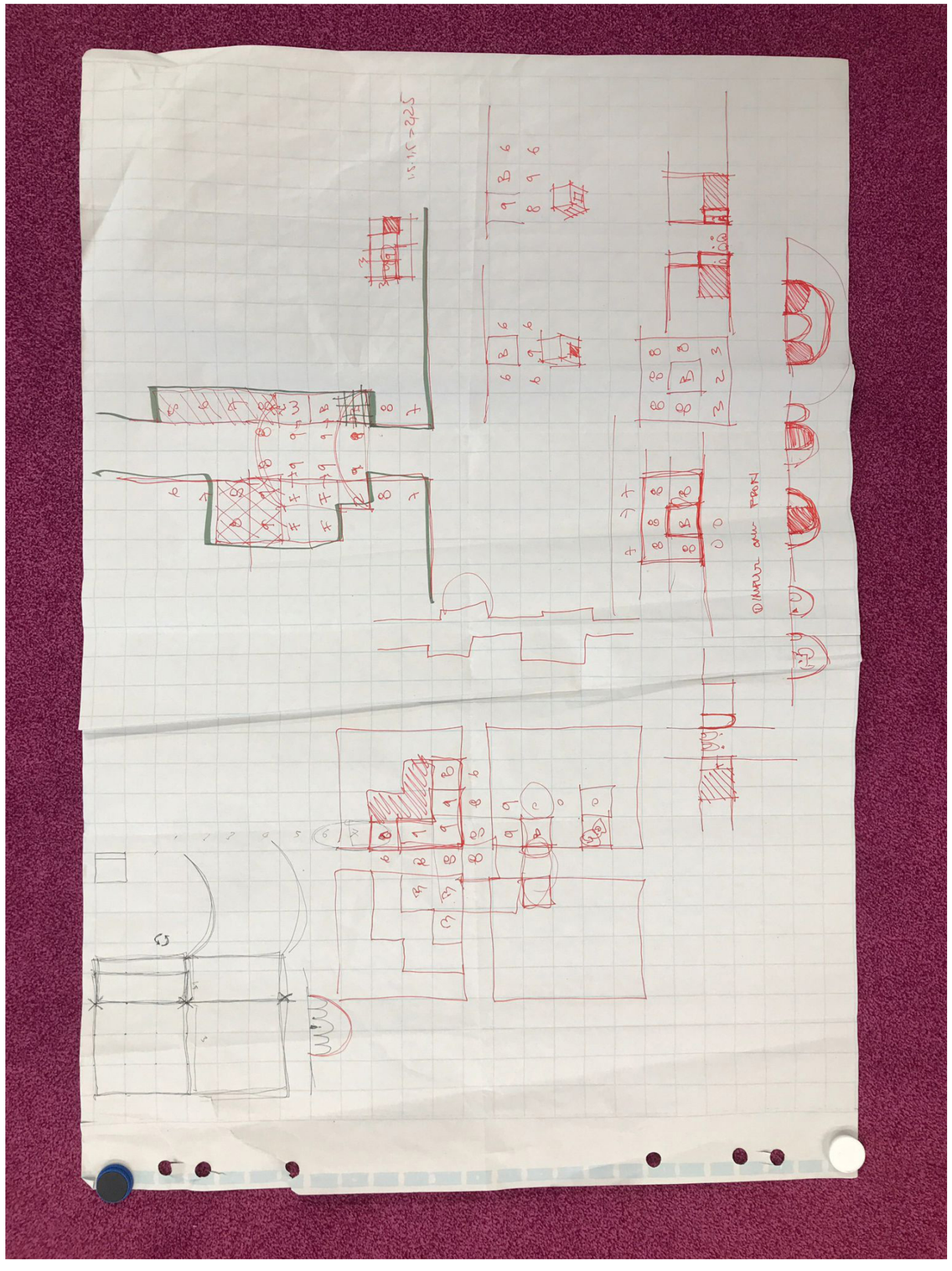
FELIPE - BINDER/SCULPTOR

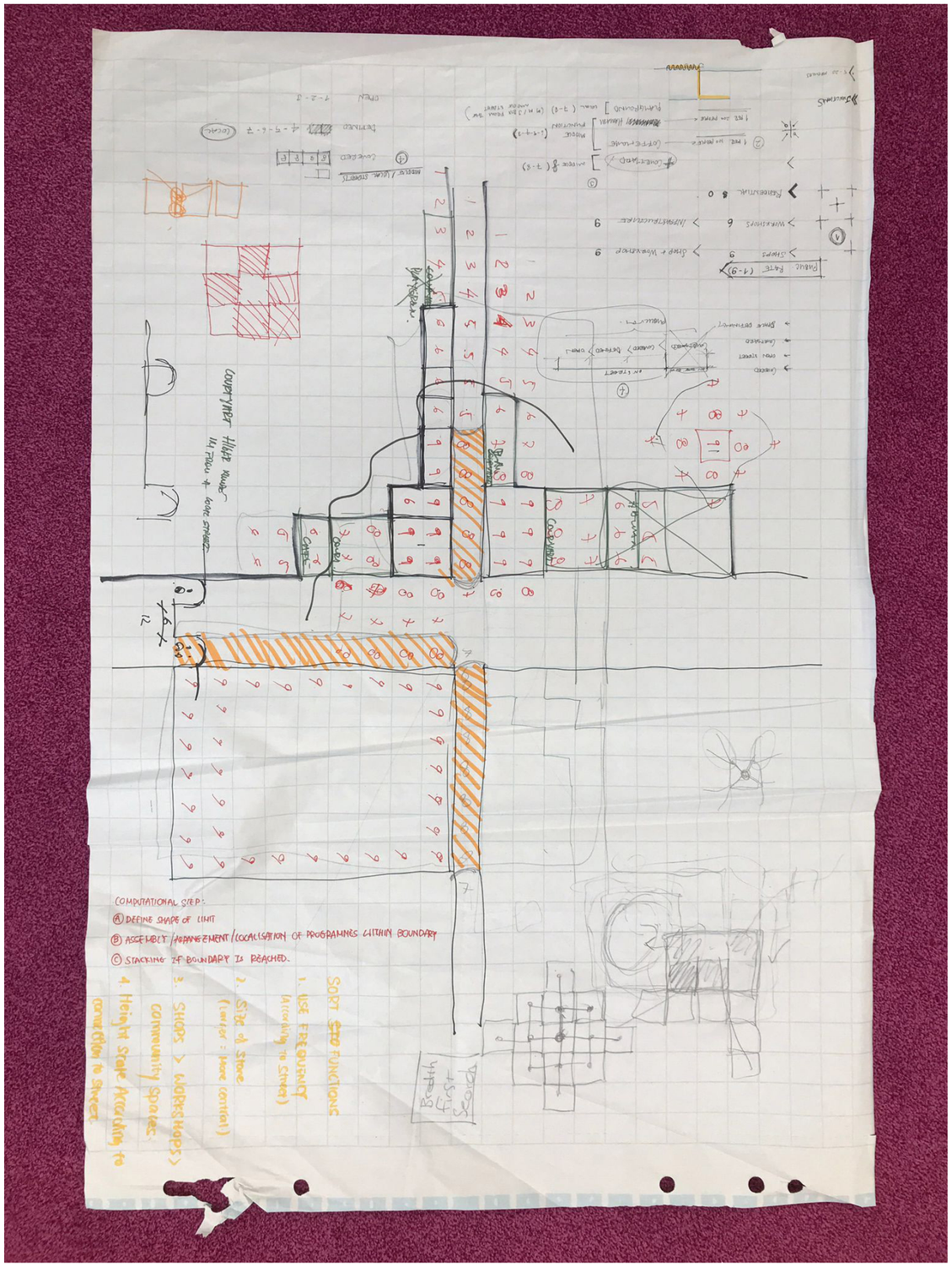
As a personal task, I developed the forming and structuring part of the project. Related to the forming face in general, the exploration process was not fluence. Nevertheless, always in the same direction. If I don't show them in the report, several approaches were developed because they are not relevant, but all of them contribute to a better understanding of the process to get the final shape.

On the other hand, the structuring process was mainly a waste of time that did not contribute to the final approach. Several weeks was dedicated to strategies that later on was not acceptable for the course, and I lost all my job. Afterwards, the second approach of bricks abstraction was complicated because it is not a known method to follow or use as a guide, transforming into an arduous task and consuming more than expected time. In the main conclusion, due to the task that I needed to develop, I didn't get the skills and lessons I expected from the course. Furthermore, due to the complication of the structural calculation was not possible to integrate with the coding process losing the opportunity to learn how to code in the architectural environment that I was expecting.

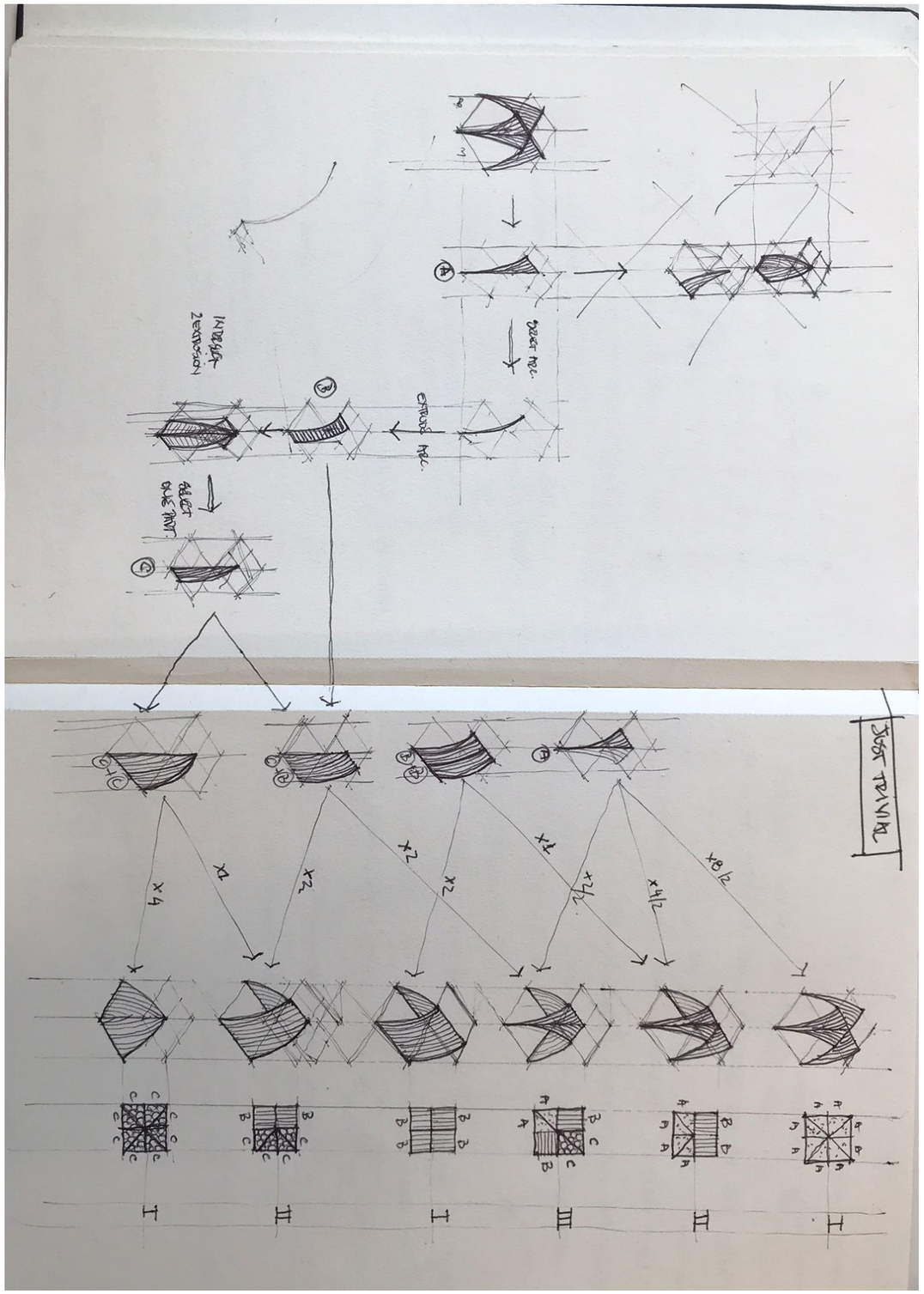
Appendix 5.0

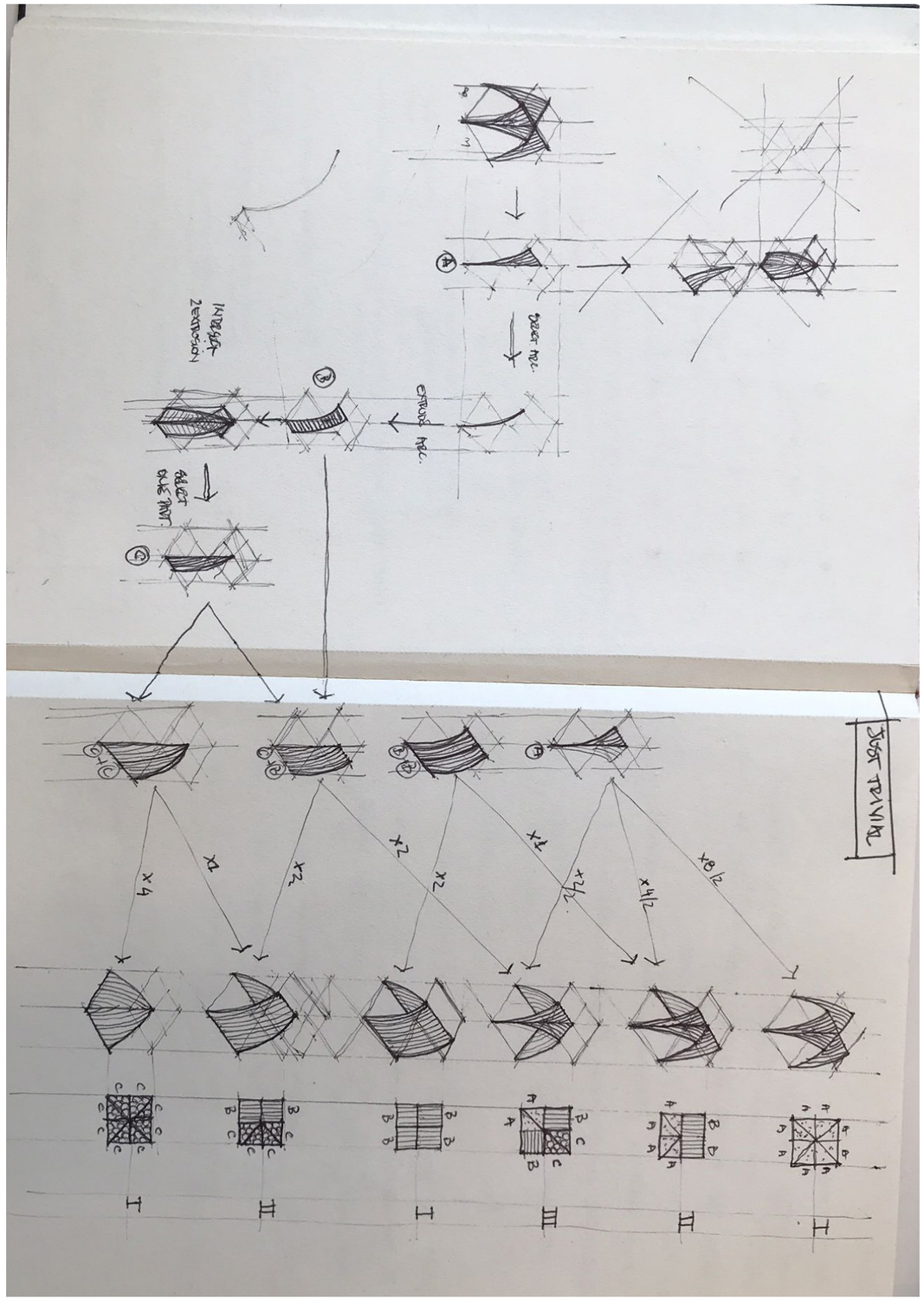
SKETCHES AND DRAWINGS

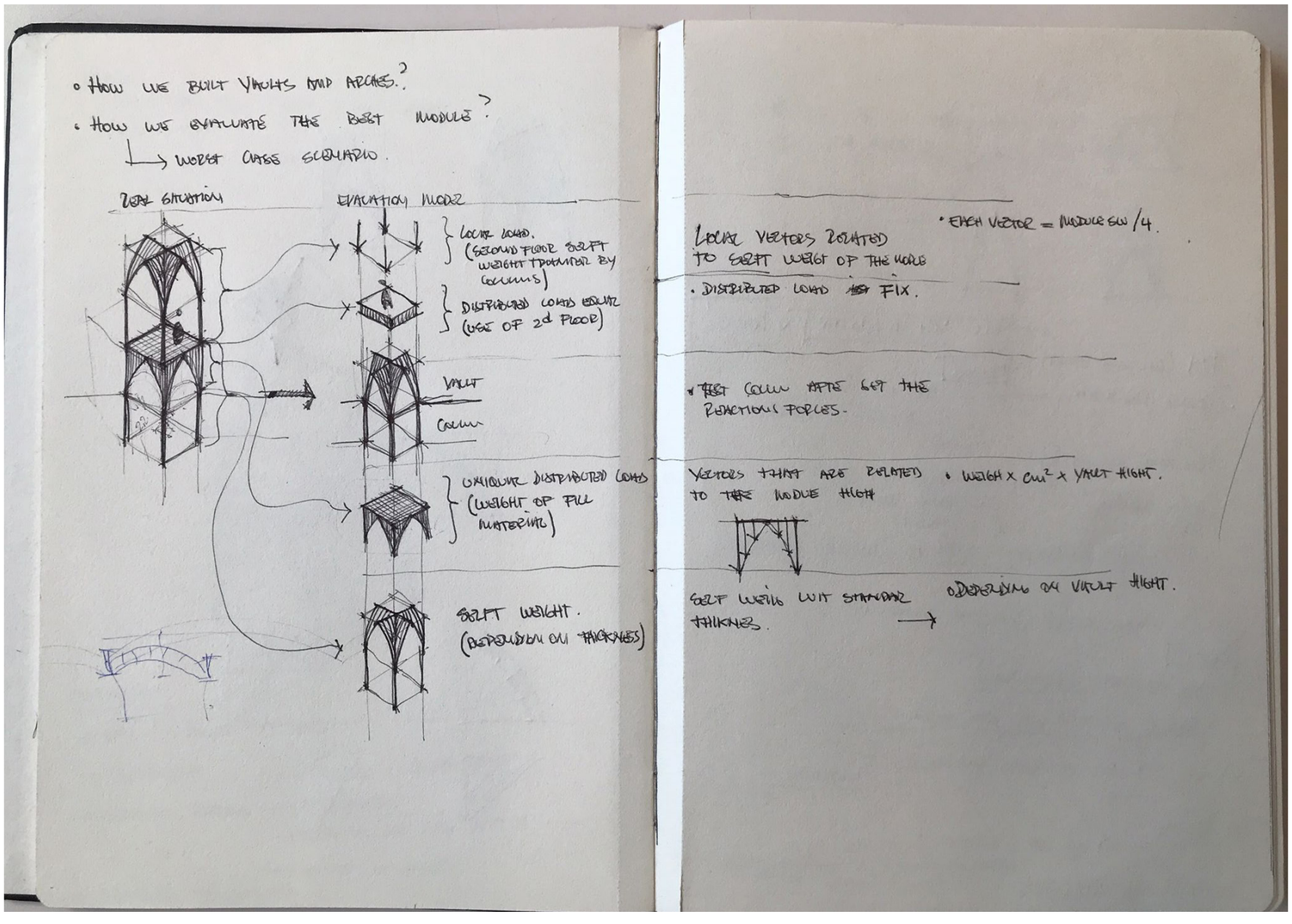




SKETCHES AND DRAWINGS







• HOW WE BUILT VAULTS AND ARCHES.?
• HOW WE EVALUATE THE BEST MODULE ?
↳ WORST CASES SCENARIO.
REAL SITUATION
EVACUATION MODEL
LOCAL LOAD. (SECOND FLOOR SELFT WEIGHT TRANSFER BY COLUMNS)
DISTRIBUTED LOADS EQUAL (USE OF 2nd FLOOR)
VAULT
COLUM
UNIFORM DISTRIBUTED LOADS (WEIGHT OF FILL MATERIAL)
SELFT WEIGHT. (DEPENDING ON THICKNESS)
LOCAL VECTORS ROTATED TO SELFT WEIGH OF THE MODE
• EACH VECTOR = MODULE SW /4.
• DISTRIBUTED LOAD FIX.
• TEST COLUM ARTE GET THE REACTIONS FORCES.
VECTORS THAT ARE RELATED TO THE MODUE HIGH
• WEIGH x CM² x VAULT HIGHT.
SELF WEIN WIT STANDAR THIKNES.
DEPENDING ON VAULT HIGHT.

SKETCHES AND DRAWINGS