Oasis

For everyone's well being

Date: 02-11-2021

Report: Earthy 2021

Students: Group 4
Marnix van den Assum 4594207
Jordy van Eijk 4566297
Doris van Uffelen 4587995
Tim Schouws 5244153
Sjoerd van Hedel 4546199

Tutors: Dr. ir. Pirouz Nourian
ir. Shervin Azadi
ir. Frank Schnater
ir. Hans Hoogenboom
Dr. ir. Fred Veer
Dr. Charamlampos Andriotis

Faculty of Architecture and the built environment
TU Delft 2021-2022

Content

0_Introduction

In this report the process of the Oasis will be explained. In history books we see and learn how an Oasis can give live to a place where we do not expect it to be possible. By storing and using water in the right way, life can appear in the middle of a dessert. Along the silk route, old Oases apeared and small economies arised. Exactly what the refugee camp Zaatari needs.

This project is part of the course EARTHY (AR3B011). This course is part of the Building Technology Master track at the TU Delft. Raw earth is a forgotten building material that has been used for ages and is everywhere around us. With the rising influence of computers on the design process and algorithms configuring floor plans, EARTHY is a course that focusses on creating earth architecture using a computational approach. The building site for this project is the refugee camp Zaatari in Jordan.

The refugee camp in Jordan has become a city with over 80.000 inhabitants, but without the facilities a city needs. This report proposes to create an oasis where flood water is stored and utilized by buildings build around it. This way, a local economy with lots of chances for employment are created. The main flowchart shows the overall process of the project. Starting with analysing the refugee camp and define the problem statement, based on this a design vision is created. This vision defines the program of requirements that is needed. From this program of requirements, a game is designed that configures a floor plan according to its rules. The floor plan gives the topology and the connectivity of the modules. These are generated by materialization analysis, shaping, forming and structural verification. Finally the construction manual of how the builders will assemble the modules using raw earth.

The final outcome of the reports aims to inspire people to use water in a proper and efficient way, while keeping the outcome open source. When the design game is played a set of rules is generated that can be tested by using a physical floor plan or by writing code on Python. This will hopefully make the world better for all.

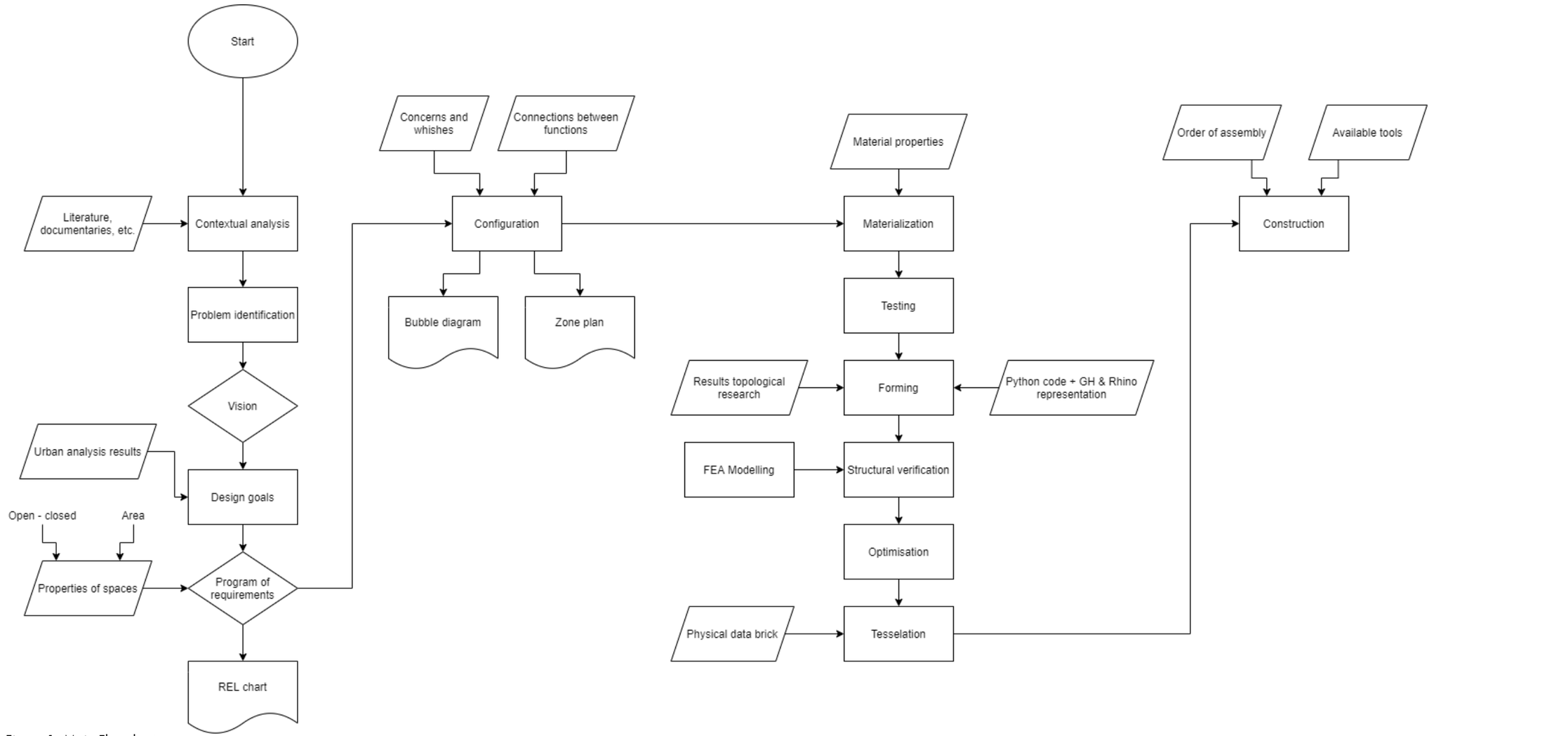


Figure 1: Main Flowchart

1_Research and Analysis

Figure 2: Research into the plot

Urban analysis

Za'atary is located in the north of Jordan. It is a Refugee camp close to the boarder with Syria and houses mostly Syrian refugees. The Syrian refugees are fleeing from the violence of the Syrian civil war. It started in July 2011 with a hundred improvised tents. This was several months after the attack on Daraa. The refugeecamp opened in 2012 and houses around 80.000 people. It was not planned to have this amount of people, which means the camp has expanded over the years. It also evolved from a temporary solution to a permanent one. (de Haas et al., 2020)

In terms of size, it is almost as big as the old city center of Amsterdam. The camp is located near the city of Mafraq, a city that was created because of the international trade routes to Damascus and Iraq. There is also a military airbase and a few universities nearby. The camp was placed here because of the good accessibility with regard to logistics and goods. When the conflict ends, the refugees can return to Daraa, just across the border.

Centuries ago, this area was a steppe, where the nomads let their cattle graze. The soil now consists mainly of clay with sand. The area is characterized by the amount of basalt blocks. It comes from the very old volcano 150 kilometers north of the region that was active thousands of years ago.

Figure 3: Current situation Zaatari (ESA,s.d).

Urban analysis

For the urban analysis, the dimensions of streets and souks in Syrian bazaars have been looked at. The main takeaway from this analysis is the width of the streets, souks and most of the shops. The size of these functions is approximately 5 meters.

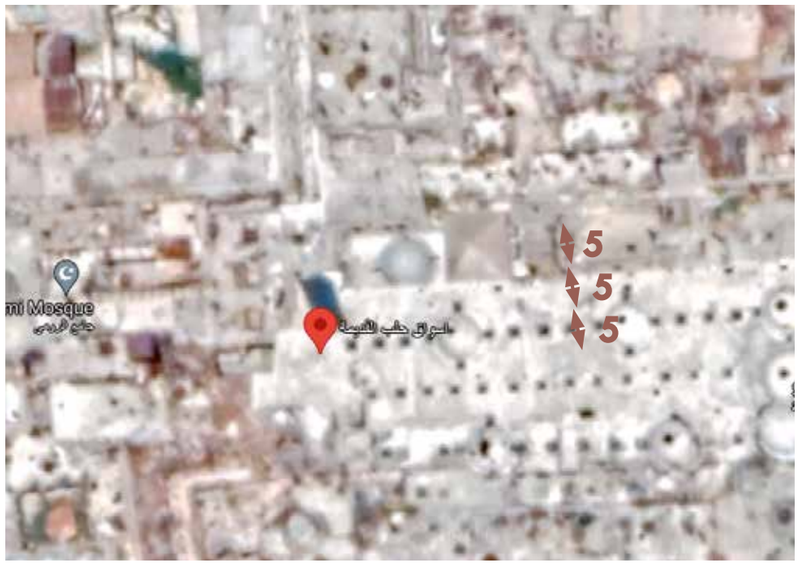


Figure 4: Google Maps, Dimensions Bazaar Aleppo

Figure 5: Google Maps, Bazaar Aleppo

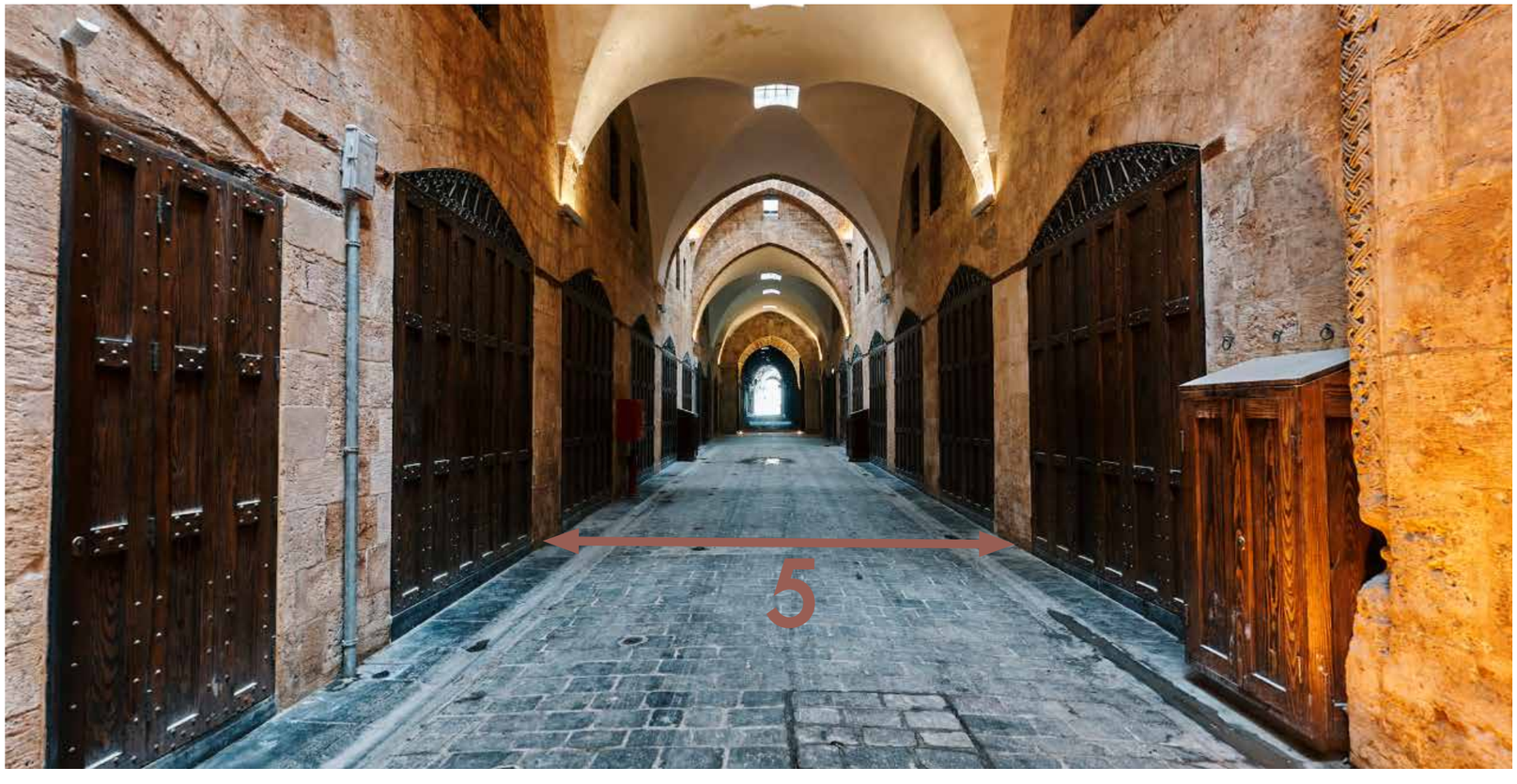


Figure 6: Google Maps, Dimensions Bazaar Aleppo

Climate

The refugee camp is located at the transition between the temperate climate of the Amman region and the desert. The climate in the area of the refugee camp is dry and warm. The summers are long, hot, arid and the winters are cold & windy. Every year about 665 million liters of water falls into the camp. This is an average annual precipitation of 150-200 mm/year/m2, which can be seen in the image below.

The inhabitants of Jordan and the refugees use more water than the amount it rains per year, because of this some groundwater levels have dropped more than 1 meter per year. Agriculture consumes around 60% percent of the water supplies, with more than half of the water estimated to be lost by theft and leaky pipes. Due to the many refugees in the country, the demand for water has also risen sharply. Extreme water scarcity and wide disparities in public water supplies are potent ingredients for conflict.

When it rains it falls in large quantities and in a short period of time, water is not collected and removed. This causes floodings in the refugee camp. This is partly because the soil quality is very poor. The soil consists mainly out of clay and is disturbed because the large group of refugees further disturbed the soil. The soil has become increasingly compacted, so no more water infiltrates into the ground. The rain is damaging their fragile new homes and offers almost no protection against the harsh climate. It causes problems with housing, hygiene and creates health risks (Vilsteren, 2019).

In the refugee camp, the water is pumped from the basin at three different places. The water from the basin is used for drinking water and water for irrigation of small vegetable gardens. This leads to even greater pressure on the water supply. The water is also additionally supplied with the help of trucks.

The image below shows that in the places indicated in blue, the water remains for more than 14 days after heavy rainfall. The blue lines indicate the flow direction of the water. The brown areas are the other areas where the water cannot drain well and creates large mud spots.

During the realization of the camp little or no account was taken of the existing landscape. The camp is located between a plateau and a depression of the landscape. The difference in height in the camp is about 20 meters. The higher parts are at approximately 667.5 meters and the lower parts at 647.5 meters (Vilsteren, 2019).

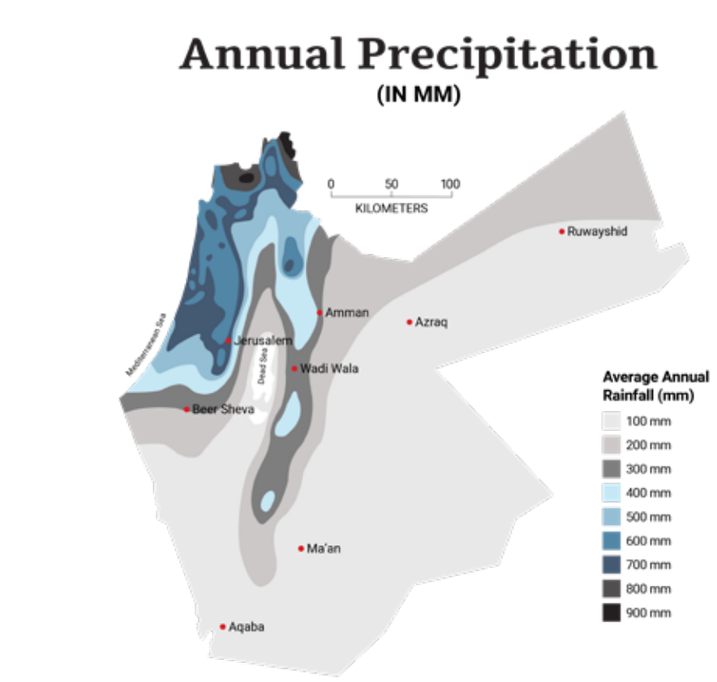


Figure 7: Annual precipitation Jordan (cbsnews,s.d).

Figure 8: Floodings Zaatari (weather, s.d)

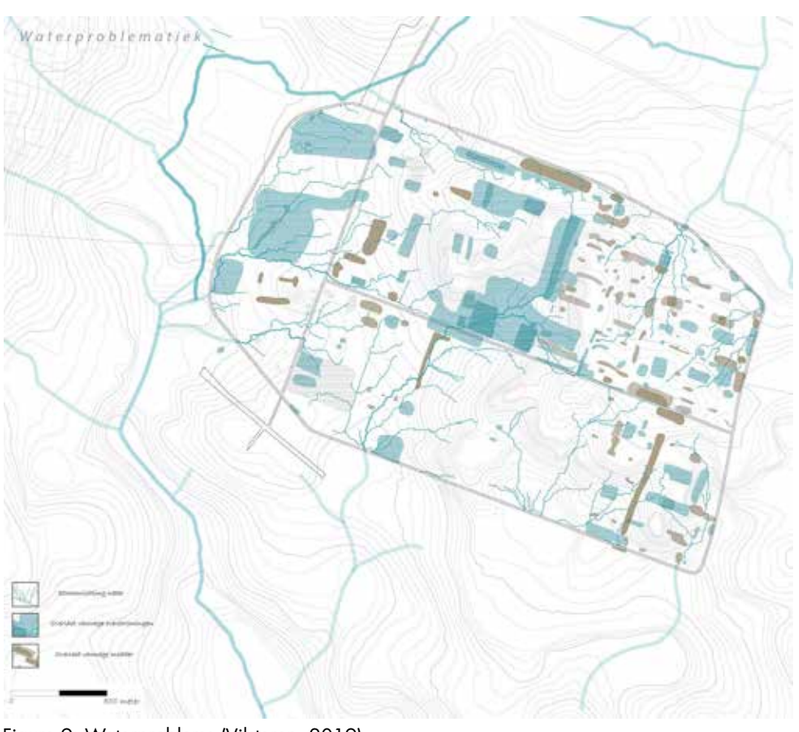


Figure 9: Waterproblems (Vilsteren, 2019)

Climate

Economy

The economy of the area is mainly linked to agriculture, small-scale livestock and trade that has developed along the various international roads. Typical of the agricultural system is that the agricultural land is used temporarily. After two years the ground is exhausted and people move to a new piece of land. In relation to Jordan, the region of Al'Zaatari is poor in raw materials and economy. This is partly due to the higher regions being better anchored due to historic trade routes, such as the Silk Road.

In the refugeecamp the shops, community centres and other facilities are located along the red lines in the picture below. As can be seen these are found on the edge of the camp and along the two main roads in the camp. More and more illegal shops have established themselves near the two main roads. There are currently about 3000 illegal/legal shops in the camp with a monthly economic value of 10 million dollars per month (Rettman, 2014)
.Almost every product is available in the camp. Even people from outside come to the camp to get certain products. The main legal general facilities, such as the hospital, are mainly located in the center of the camp near the central road so that it remains easily accessible to everyone. (Vilsteren, 2019)

Figure 10: General facilities

Current water storage

Water is currently stored in small water towers and some wells. As can be seen from the picture below, most of the wells can be found on the east side of the camp (the less population dense area). The westside (old part) of the camp is the oldest part of the camp and was created without a structure. The new part has been set up with a structure. A water purification installation has been realized outside the camp. When draining the dirty water, no distinction is made between gray and black water. This puts even more pressure on the groundwater level.

Amount of water what can be collected

In the image below, the camp is divided into 8 different districts. To see how much water we can collect, it has been calculated for each district how large the water basin must be to collect all the water. It has been taken into account that all the water is collected and nothing disappears into the ground or evaporates. In order to collect all the water, the size of the basins for the different districts is as follows:

Distric nr.	Hectares	capacity (m^3)	Required size water basin (m)
1	107	128800	100 x 8 x 161
2	51	61600	100 x 8 x 77
3	64	77600	100 x 8 x 97
4	34	41600	100 x 8 x 52
5	76	92000	100 x 8 x 115
6	72	87200	100 x 8 x 109
7	65	78400	100 x 8 x 98
8	35	43200	100 x 8 54

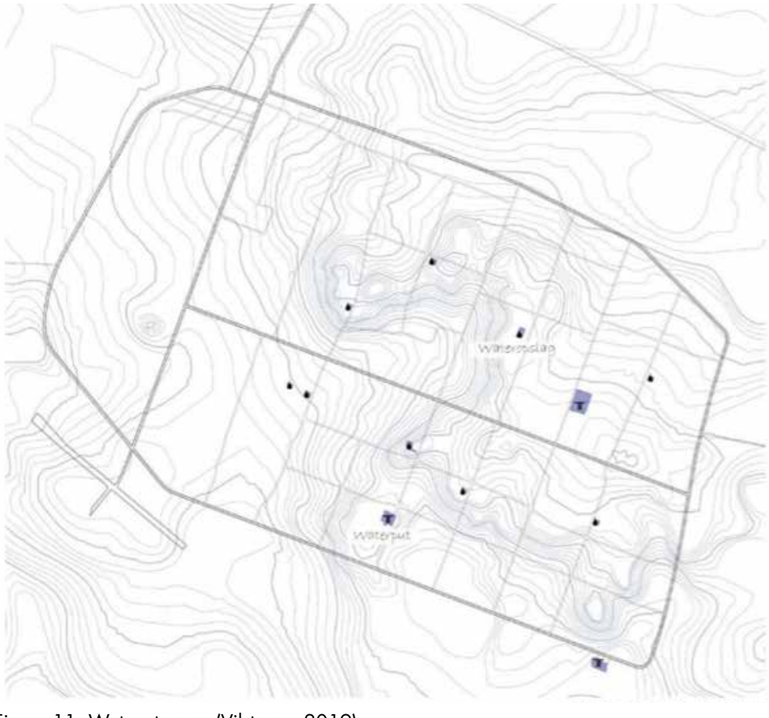


Figure 11: Water storage (Vilsteren, 2019)

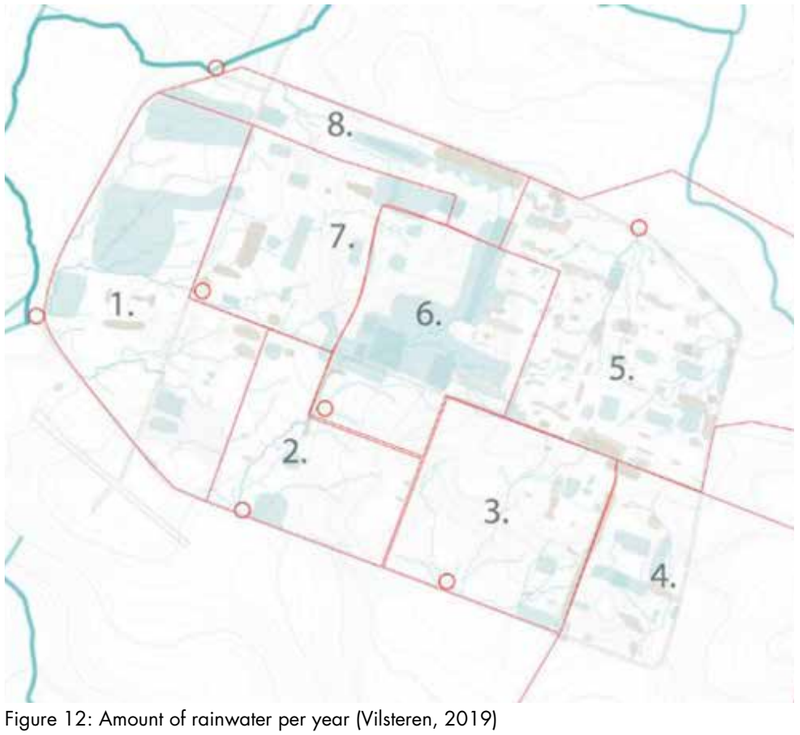


Figure 12: Amount of rainwater per year (Vilsteren, 2019)

Problem statement

Problems

To get an idea of what can be improved in the camp, we first analyzed the problems in the camp. With these possibilities we can create the right vision for the refugeecamp.

- Zaatari is designed as a temporary camp. With the exception of asphalt roads, there is no permanent structure. The temporary infrastructure is therefore more heavily used than it actually can handle. This causes many problems during heavy rainfall.

- There is a lack of social control, which leaves room for bad behaviour and organized crime.

- The environment is very unhygienic. The camp is filthy and it attracts vermin. People easily get sick from exposure to these pests. During heavy rainfall, the washing facilities will flood. All black water will spread through the camp during heavy rainfall.

- A very hot summer with lots of sandstorms and a cold winter with lots of rainfall make the weather conditions very harsh. Houses are not designed for these conditions.

- There is no sewer network and black water is temporarily stored at wash facilities and collected once a week.

- Because of the low employment rate of 17% throughout the camp, there is a lot of boredom which leads to violence. 20% of the people with a working permit is female (Kruit, 2014).

Vision

Zaatari Refugee Camp was built to be temporary, but grew out to be a permanent living space for many people. This means the facilities that make a city are missing, which causes boredom and violence. Water and food are massive scarcities and seasonal floods cause bad living conditions. We give the solution to this, by designing a building that tackles multiple problems. We propose to build the Oasis, where people can come to work, get educated and meet with other residents. The bazaar we create is built around a canal system where floodwater is stored. It gives a place fo product chains like carpets, fabrics and agriculture where the flood water is utilized. This way we create a development centre and local economy for the current and the future generations to come.

It's a city

Figure 13: Aerial view Zaatari, (2013, AP)

But does not have the facilities for a city

Figure 14: Zaatari refugee camp, (Adayleh, 2015).

Urban vision

Improvement of the infrastructure

Most of the streets have no proper roads. With the exception of the two asphalt roads there is no permanent infrastructure. The camp is overcrowded and with the temporary infrastructure it is more used than it actually can handle. The roads are dusty in the summer and muddy in the winter. With the bazaar, people can used it to get from point A to point B, while being protected from the sun and sandstorms. To collect all the rainwater, a profile can be chosen in which the water can be directed to the bazaar and outside the camp. The infrastructure will also immediately be improved.

Decrease the risk of flooding

During Jordan's rain season, homes, schools and hospitals are usually flooded with water. The water also stops essential interventions in the camps, blocking access for water trucks. Our vision for the design is that the water will be collected in a basin, so that the flooding can be remedied. During the dry periods it can be used for irrigating plants and cooling the buildings. By collecting the water, the floodings in the camp will also be reduced. The water also ensures more greening in the camp.

If you only look at the floodings the camp, the problems whats comes with the floodings, are the most severe in two places in the camp. The 2 locations are shown in red in the image below. The plateau of the landscape is in the middle and the depression of the landscpape is at the bottom of the map.

Position 1: Position 1 is in the old part of the camp. This area is also the most densely populated and with a heavy floodings this will cause inconvenience for many people. The older parts will be the most problematic in terms of structure and quality in the future.

Position 2: Position 2 is in the center of the camp. This is where most of the water comes together and causes nuisance to important general facilities, such as the hospital, schools and activity centers.

Figure 15: Floodings in the camp (Tomaszewski, 2019)

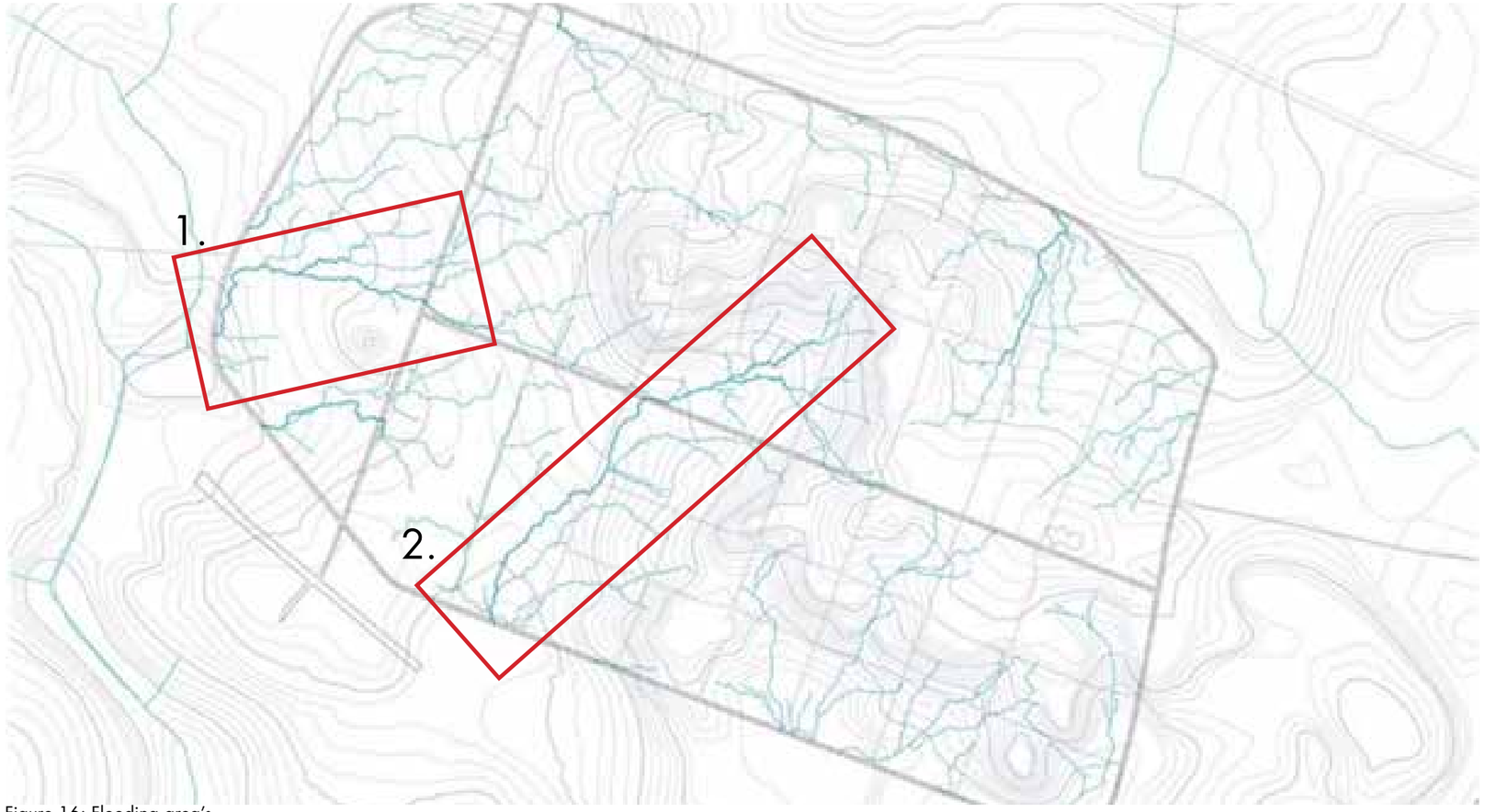


Figure 16: Flooding area's

References

Hanging gardens of babylon

The gardens must have existed for over 250 years. It is possible to determine from the descriptions how large the gardens must have been, but traces of such foundations have never been found. The gardens consisted of terraces, surrounded by walls on which trees, flowers and plants were planted. The plants hung over the rivers of the Euphrates. The walls consisted of a large hollow construction, filled with fertile soil. The construction of the irrigation system was not visible. Each terrace needed a supply large enough to supply all the plants with water. The irrigation system consisted of 2 wheels, one in the upper water storage and one in the water storage below. The two wheels were connected by a cable with buckets connected to the cable. Slaves provided movement in the cable so that the upper water storage could be filled. A ledge was created at the water basin, causing the buckets to turn around at the water basin. (Wereldwonderen,sd)

Ab anbar

An Ab anbar is a water storage tank originally found in Iran. the tanks are build largely underground so that they have a good resistance against earthquakes. The pressure of the water in the basin is absorbed by the surrounding ground. The basin is covered with a layer of bricks with a special type of cement, saruzj.
The dome build over the tank is made to capture water that would otherwise evaporated in the high temperatures. Wind towers can be build with an ab anbar to keep the water cool throughout the year. An added effect of the windcathers is that the room is very well ventilated preventing it from being damp. An ab anbar can be filled by underground tunnels originating at a qanat. (Ghanavati, sd)

Caravanserai

A caravanserai is a place along the silk route where travelers could rest safe and even trade. The building had multiple functions and there are many found on the silk route. One of the functions of a caravanserai is to provide a safe overnight stay for traveling merchants, their wares and their animals. Buildings were generally situated one day's travel of each other.
A caravanserai usually consisted of a large courtyard with a closed ring of buildings around it. The gate had to be wide enough for camels to pass. In the buildings there were small niches and a room where both travelers and animals could sleep. In the middle was usually the courtyard where most of the activities took place. Here people met, animals rested and small bazaars were set up where merchants could trade their goods. The courtyard often housed a fountain or a large well to provide water. (Society, 2019)

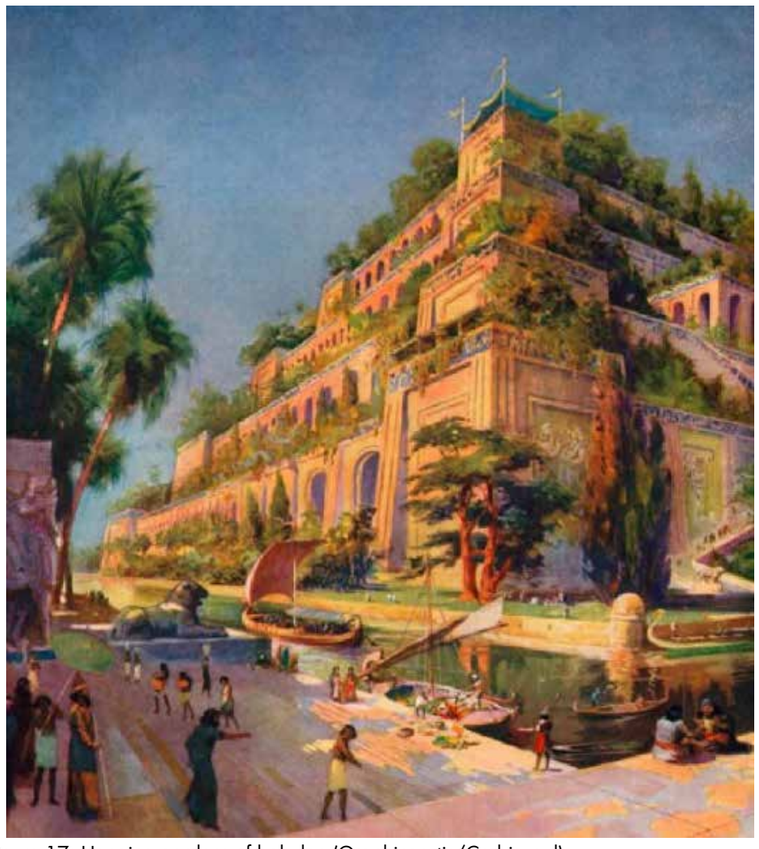


Figure 17: Hanging gardens of babylon (Graphicaartis/Corbis, s.d),

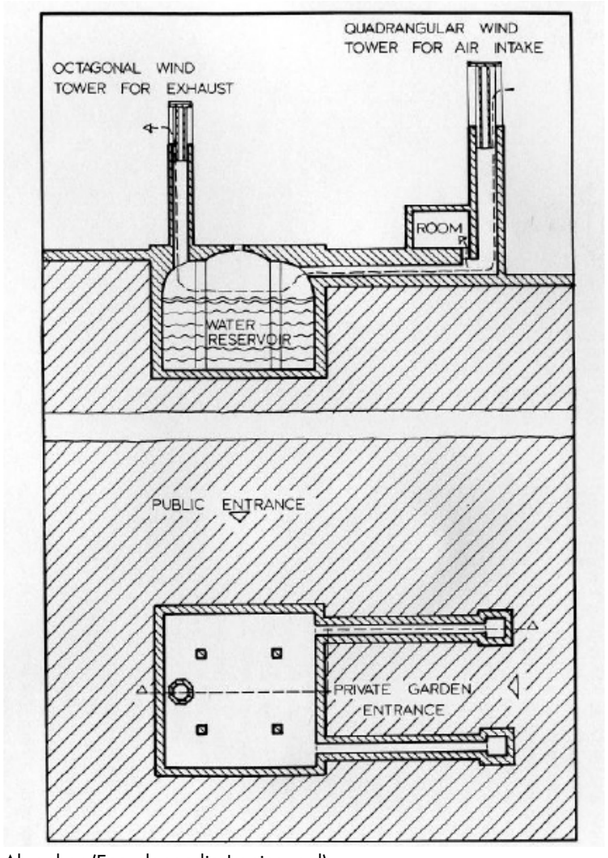


Figure 18: Ab anbar (Encyclopaedia Iranica, s.d)

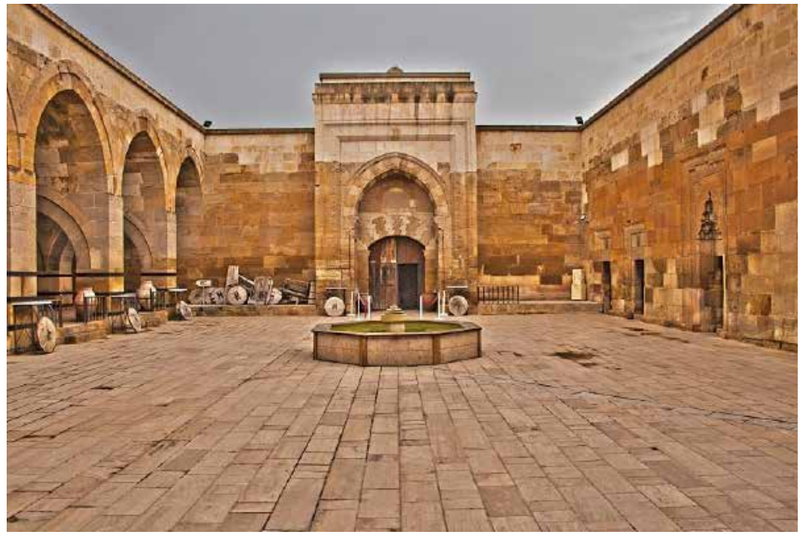


Figure 19: Caravanserai (pasabag, s.d)

References

Bazaar

Bazaars are permanently enclosed marketplaces that are central to everyday life in the Middle-East, North- and West Africa and parts of (South-East) Asia. These buildings carry great cultural value and date back to 3000BC. Some bazaars are known to be over 400 years old. Bazaars are a network of merchants, bankers and craftspersons who often group with businesses like their own. This enhances the economy because they all compete for market share and it is easy for customers to compare prices and quality. There are different kind of shops to be found: Jewelry and pottery shops, Furniture and carpet shops, Leather and clothes shops, Tea and herbs shops, Restaurants and tea houses. The bazaar also forms an important meeting space. The layout of a bazaar has four main elements, which each fulfill a different function:

- -Rasteh Lanes: with shops across both sides
- Caesarea: Similar to the Rasteh, but more expensive products, like jewelry and silk are sold here.
- Chaharsough: This is the junction between Rastehs, which is used as a meeting space. It has a tall dome and a fountain in the middle.
- Timcheh A small indoor space which houses small factories and furniture shops.

Most bazaars are always open, products are mostly produced on site in small factories at the back. They are located near mosques and have their entrances turned towards them. Often, pop-up stores appear around the entrances. (Smit et al., 2019)

Indian stepwell

Stepwells are underground buildings usually with a central pond or well. They are usually located in arid regions of western India, Pakistan, Indonesia and surrounding regions. They were built to provide the local population with water during drier periods of the year. The water can be used for drinking, washing, bathing and irrigating crops. It is a central well that extends down into the underlying water table. The integration of stairs makes it easier for people to access the water supply. Many stepwells also served as cool shrines for traveling caravans and pilgrims. Stepwells were built by excavating several floors underground to reach the water level. When the water level is reached, the wells and walls were lined with masonry and stairs were also added from ground level to the water reservoir. Many stairwells gradually narrowed from the surface to the lowest point, where the temperature is refreshingly cool. In villages they were also known as withdrawal wells. Since fetching water was a task often assigned to women, it was also a social activity. (McFadden, 2020)

Conclusion

Certain points emerged from the reference study that we want to include in our design. We want different levels with roof terraces(Babylon) where they are connected with a design for a stepwell. The water is stored underground to prevent evaporation (Ab anbar). The water can then be used again in drier periods during the year. To get the water up there, there will be several holes in the ground to get the water out of the storage with rope and buckets. This is also more hygienic and prevents it from becoming unhygienic after a while, just like the stepwells. We want to have several shops in the plan where all products are produced on site in small factories (Bazaar). There are houses around the shops to make the area attractive at night.

Figure 20: Bazaar (Wood, s.d)

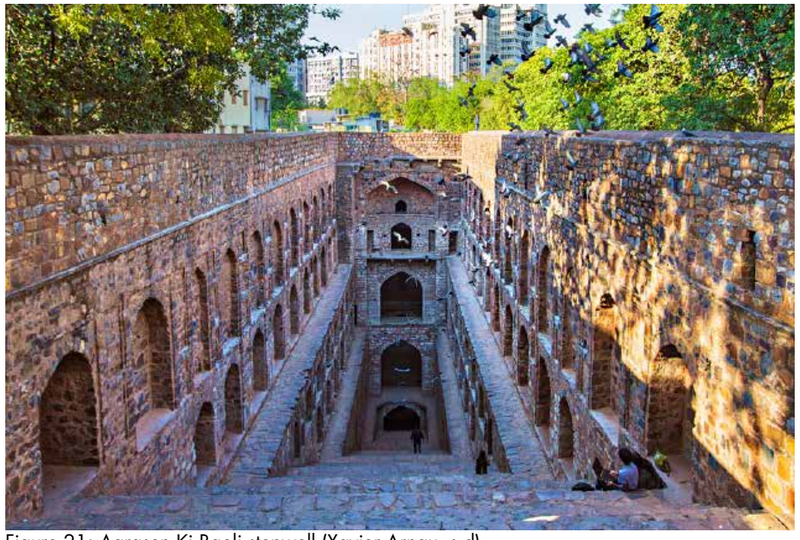


Figure 21: Agrasen Ki Baoli stepwell (Xavier Arnau, s.d)

Figure 22: Chand Baori stepwell (Matthias Kestel , s.d)

Location

The Oasis contains functions for living, working and storing water. In addition it needs to be accessible. So for determining the right location, four different maps are considered:

- Flooding: a map that shows where floodings occur in combination with a height map.
- Shops: where in the camp are the shops currently located?
- Dwellings: where in the camp is space to construct dwellings? What is the current density of the dwellings?
- Infrastructure: where are the main roads located and will the location be accessible?

These four maps and their questions will give the location of the plot. According to the questions mentioned above, a scoring system is created. The location with the highest score will be the location of the Oasis. This will be explained on the following page.

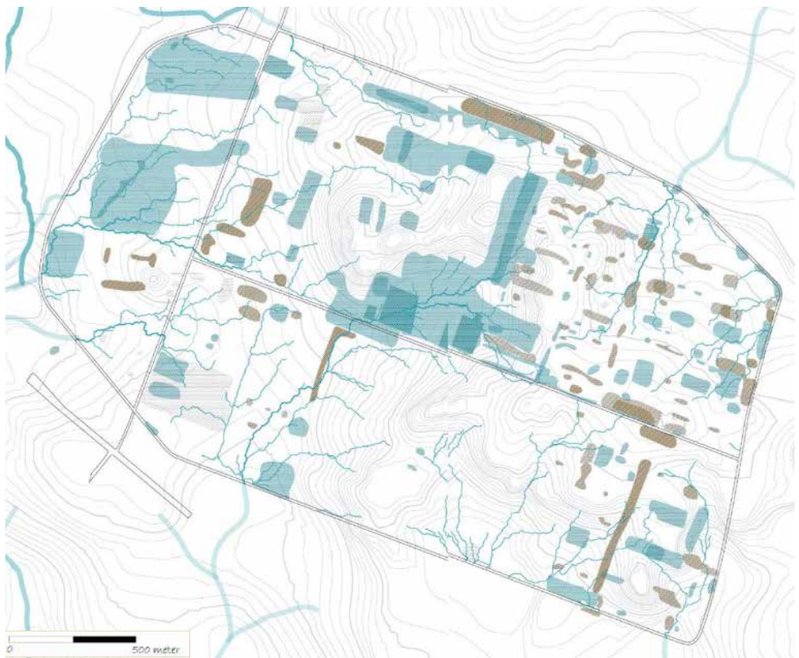


Figure 23: Flooding map. (Vilsteren, 2019)

Figure 24:, Shops map. (Vilsteren, 2019)

Figure 25: Infrastructure map. (Vilsteren, 2019)

Figure 26: Dwelling map. (Vilsteren, 2019)

Location

According to the previous page, a scoring system per map is needed to define the final location of the Oasis.

As can be seen in the images to the right, the plot is divided in a matrix. Within each cell a score from 0-5 is given. For example for water storage, zero is given when there is no potential water storage. Five is given when there is a lot of potential in storing water. This scoring system is repeated for the infrastructure, shops and dwellings.

Eventually, all the cells are given a score for each topic. When all these cells are added. A final score for each cell will confirm the most appropriate location.

Within this scoring system a weight factor is added. Because water storage is more important than the available space for the dwellings. So the following ranking is set up: Water has a weight of three, Infrastructure and shops 2 and finally dwellings 1. This will give a final matrix, which will be explained on the following page.

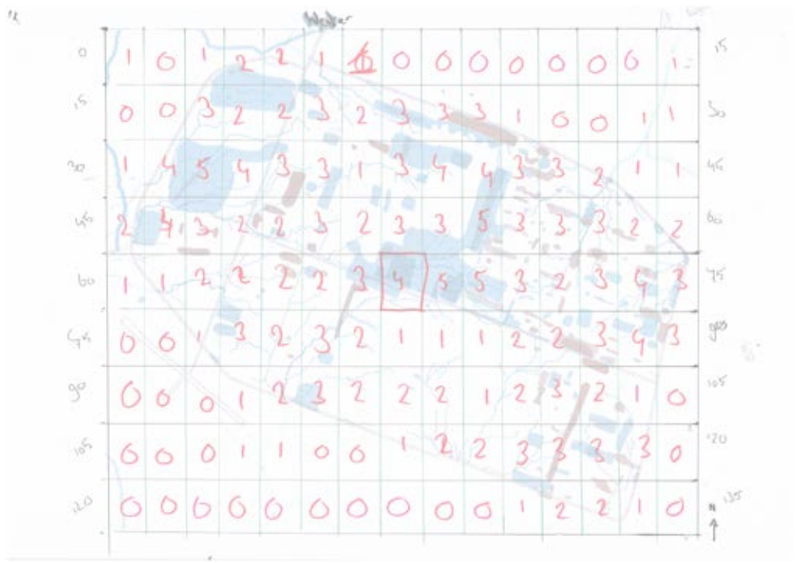


Figure 28: Scores of water.

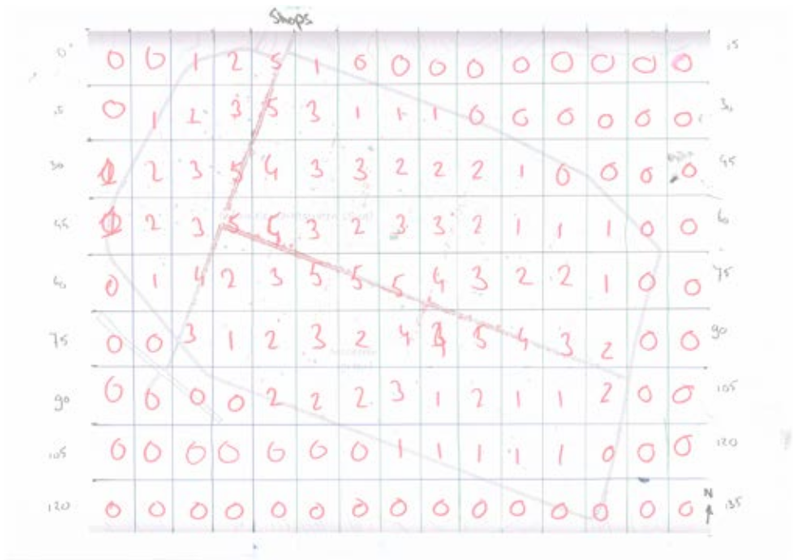


Figure 29: Scores of shops.

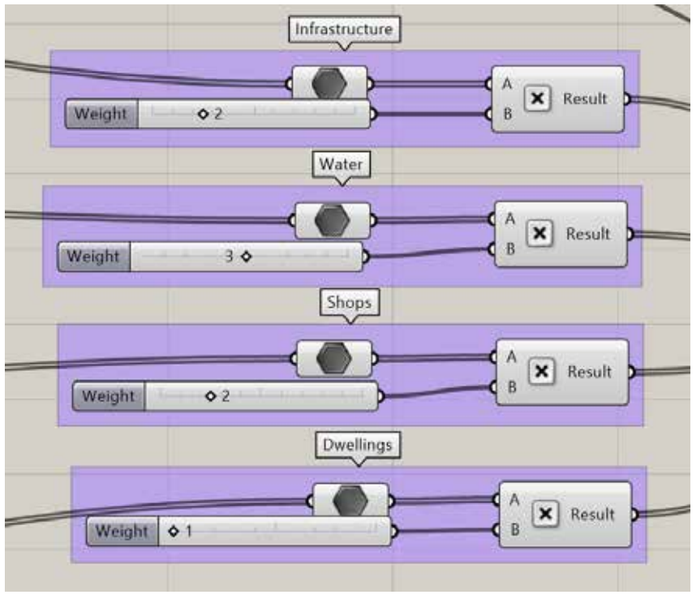


Figure 27: Weight on data.

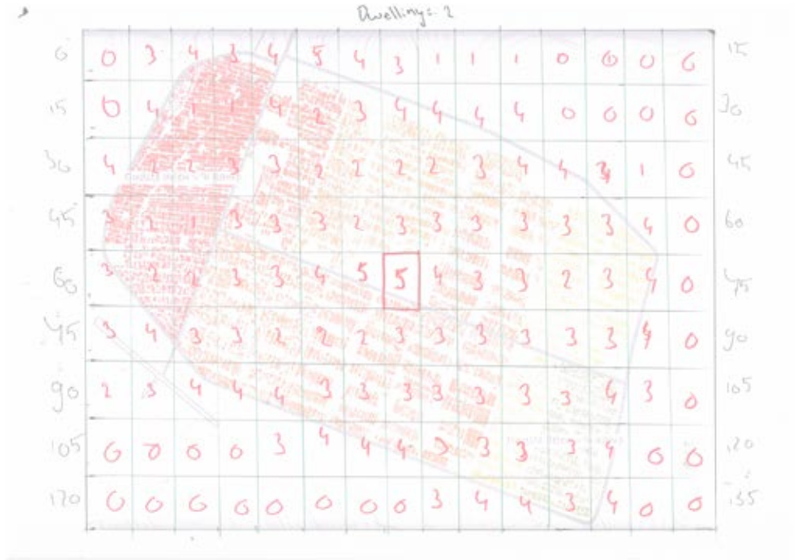


Figure 30: Scores of dwellings.

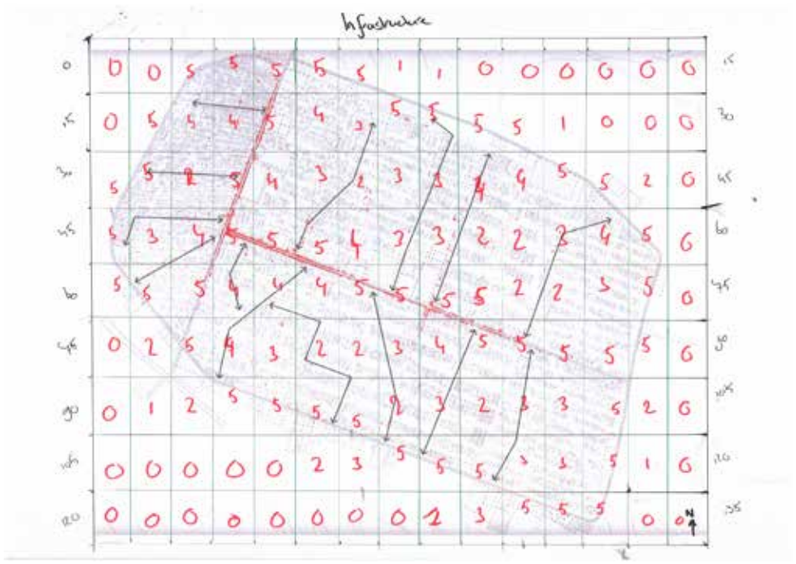


Figure 31: Scores of infrastructure.

Location

A final matrix with the scores are shown below. To visualize this, a heat map is made. The heatmap on the right shows the scores of each spot.

The red rectangle indicates the plot chosen for the bazaar. This plot is within the cells that score over 35 points. The reason to place the cell to the south-west of the highest score is the height map. Water flows to the lowest point.
This will be illustrated on the following page.

3.000	3.000	19.0…	23.0…	30.0…	20.0…	17.0…	5.000	3.000	1.000	1.000	0.000	0.000	0.000	3.000
0.000	16.0…	22.0…	21.0…	30.0…	25.0…	17.0…	25.0…	25.0…	23.0…	17.0…	2.000	0.000	3.000	3.000
19.0…	28.0…	27.0…	35.0…	28.0…	23.0…	15.0…	21.0…	24.0…	27.0…	23.0…	23.0…	20.0…	8.000	3.000
21.0…	24.0…	24.0…	29.0…	27.0…	28.0…	20.0…	24.0…	24.0…	26.0…	18.0…	20.0…	22.0…	20.0…	6.000
16.0…	17.0…	26.0…	21.0…	23.0…	28.0…	34.0…	37.0…	37.0…	34.0…	20.0…	16.0…	20.0…	26.0…	9.000
3.000	8.000	22.0…	22.0…	18.0…	21.0…	16.0…	20.0…	22.0…	26.0…	27.0…	25.0…	26.0…	26.0…	9.000
2.000	5.000	8.000	17.0…	24.0…	26.0…	23.0…	19.0…	17.0…	14.0…	17.0…	20.0…	24.0…	8.000	0.000
0.000	0.000	0.000	3.000	6.000	8.000	10.0…	19.0…	21.0…	21.0…	20.0…	20.0…	23.0…	11.0…	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000	10.0…	17.0…	19.0…	20.0…	3.000	0.000

Figure 32: Matrix of total final scores.

Figure 33: Heat map of total final scores.

Location

The following figure to the right illustrates the final location that is chosen for the oasis. The figure zooms in on the three cells that are above 35 points.

Eventually, the spot to the south of the hospital is chosen. As a result the hospital is not removed to place the oasis. The other reason is related to the height map. Which can be seen in the figure underneath. In this figure can be seen that the water flows from the flooded area (where the hospital is located) towards the south-west of the map.

By adding canals near the hospital, the water is directed to the plot and can be stored over there. So the flooded area is irrigated, and the water is collected downstream. By this the final location is defined.

Figure 34: Height map and water. (Vilsteren, 2019)

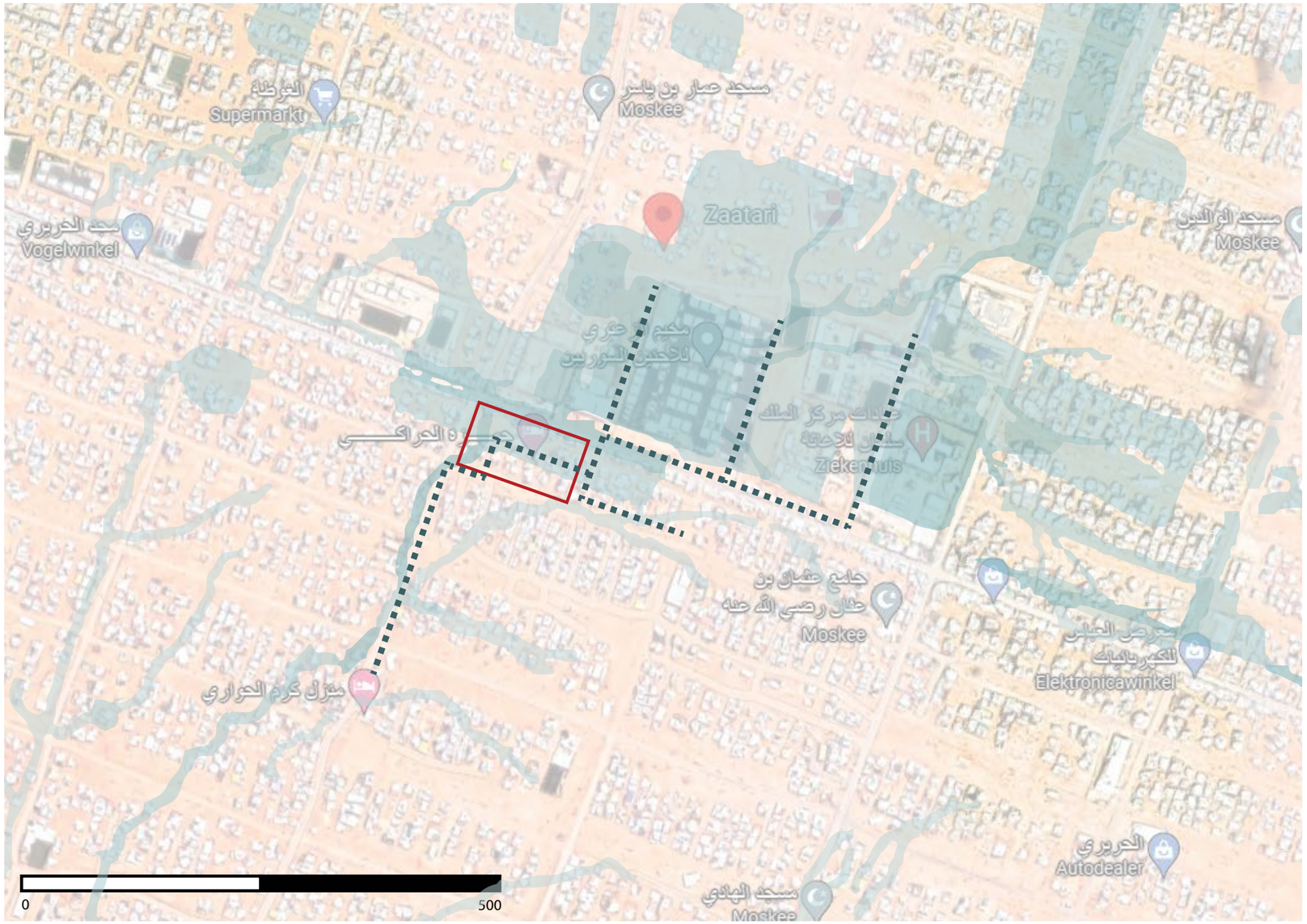


Figure 35: Chanels for irrigating and flooding.

Pre midterm Bubble diagram

After the lectures of the first few weeks we started working on the bubble diagram. Due to a misunderstanding we started by trying to make the diagram in python and directly make it into a floorplan. Although the script was not used for the final presentation we would like to present it here to show the progress that was made on it. The full script can be found in appendix C.

Import libraries
For this script multiple libraries are used. Here is a list and what we use them for:
- Pandas: Pandas is used to visualise the data about the functions and the connection matrix.
- Numpy: Numpy is used to for easier way of working with matrixes and to export the file so it can be used in rhino.
- Networkx: Network x is used to visualise the graphs made with this script, and to move the functions to their new location

Inputs
In the inputs section of the script the number of the functions can be inputted by the user of the script. Besides that a list is created with the name of all of the functions and areas are assigned to all of the functions. When all the values are collected a table is created with all the data.

Connections
To create a graph two list are needed. The first list contains all of the points that make up the graph. This list is created by looking at the length of the table and putting a list of numbers as long as the table in the list.

The second list is the list of connections. This is a list of tuples indicating the two points that the connection connects. The connection each function makes is dependent on the functions that is currently being looked at. Below an the connection between a shop and a square will be further explained:

1. Look for a shop
2. Look for a square
3. Check if the shop is already assigned to a square
4. Check what square got a shop last time. Take the next one in the list. If it is the last square in the list was used. Start at the top of the list.
5. Assign the shop to the correct square (create the connection)
6. Set the shop as assigned
7. Remember the index of the square for the next loop

Most functions have a similar way of finding the connections to the method described above. After all the connections are defined some of the shops are assigned to a second square to make them more central.

Initial visualization
When all the connections are defined and the points are in a list, it is time for the initial visualization. The connections are plotted in a connection matrix and in a graph. To plot the graph all the functions need an initial position, a name, and the size that was previously defined.

Relaxation
To steer the relaxing of the graph we used four anchor points. These anchors are the entrances to the plot. After adding the anchors the spring layout from network x was used on the graph. Afterwards the functions are moved along their connections to make the discs that represent the functions touch

Final visualization
The final steps of the script are to visualize the graph in its new state. Here we found a weird feature of network x. Although network x allows you to give positions and sizes to the functions, the software scales the graph. This makes it so that the visualization does not resemble the data that is used the create it. This became even more obvious when the graph was exported to rhino for post processing.

Rhino
After the graph was exported to rhino a visual check was done to see if the location and orientation of the functions made sense. After moving some of the functions to a more logical location a floorplan was created directly from this layout.

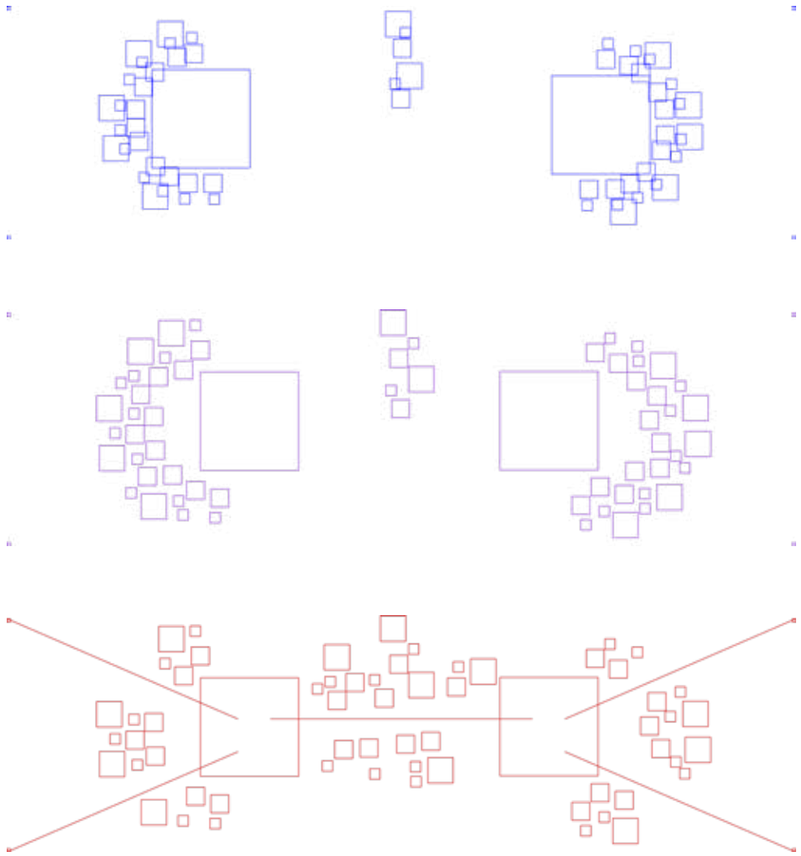


Figure 36: Post processing in Rhino

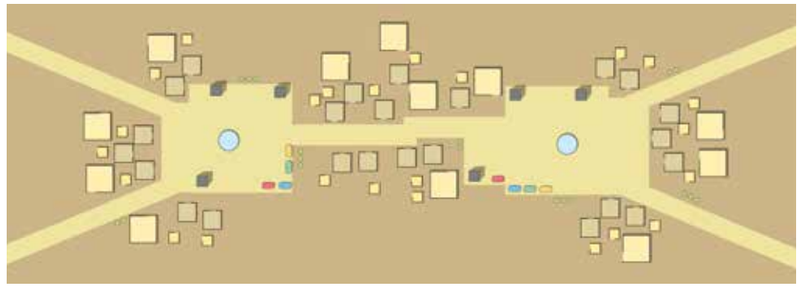


Figure 37: Bubble diagram floorplan 1

01_Configuring

When generating possible configurations a design game is being played. The vision serves as a starting point for the people playing the game. The program of requirements is translated into a REL-chart and a Bubble diagram. With this information the player makes rules for placing the functions. A grid is plotted on the building site with the measurements of the module in x and y direction. The rules the player makes are tested by writing code in python and using a physical model. In this chapter all components of the flowchart below are elaborated on.

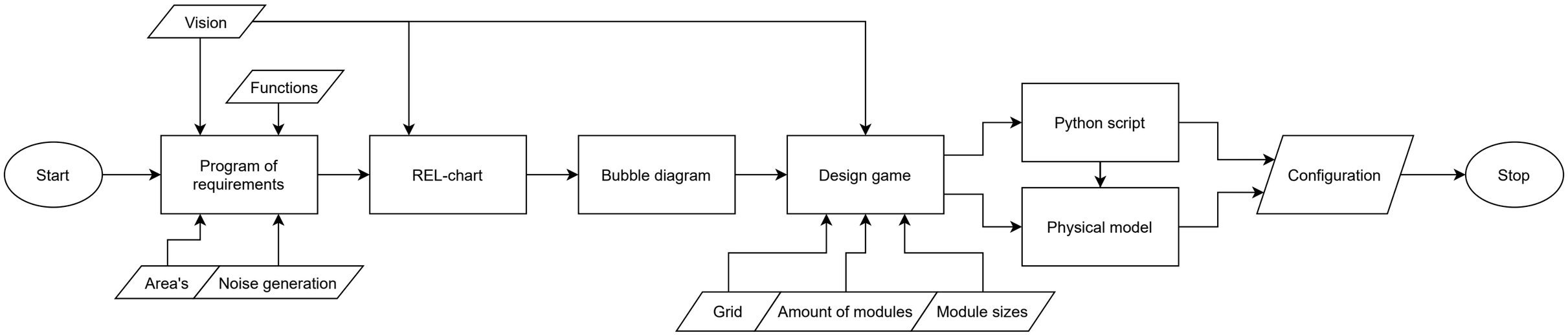


Figure 38: Flowchart of Configuring

Analysis functions

The Oasis bazaar will be centred around water. The products made and sold in this building will all need water. It is important to select the right functions that fit into the refugee camp. For this reason, an analysis was made for making and dyeing textiles, producing food and a leather tannery.

Textile

Syria has a history of making textiles and carpets. Cotton textiles are dyed, and designs are printed on them. Weaving carpets is a handcraft that takes a lot of skill and time. A handwoven carpet can take more than a month to make. They are mostly made of wool yarn, which is dyed in different colours. (Project, 2021; Karapetyan, 2019)
Dyeing fabrics and yarn uses a large amount of water and is very polluting if it is not done in a sustainable way. To prevent this, fabric can be pre-processed and dyed in an ecological way. Then, only12.5L water is needed for dyeing 1kg of cotton. (How Sustainable Dyeing is Changing the Textile Industry, n.d.)
Because the making of these products is interwoven with Syrian culture and clothing is necessary for good living conditions, it is a logical choice to make these products in the bazaar. Carpets are more of a luxury product, but they could be sold outside the camp.

Food

Having enough food for the residents in the camp is paramount. In this climate it is not easy to grow food. But it is possible, especially when water is at hand. The amount of water needed depends on the climate. But grass will need about 8L/m2 per day of water. (Crop Water Needs, n.d.)

Leather tannery

Tanning leather is a very water intesive process. Per hide 170-550 Liter water is needed. While the leather is being tanned the water becomes polluted. There are different steps in making leather. The tanning itself is done with tannins. Besides the tannis a salt water bath is required. Products made with leather are mostly luxury products. (One4Leather, 2021)

Conclusion

Both textile and food add a lot of value to the camp. Leather is not very useful and does not especially fit into the culture very well. For this reason textile and food processing are chosen. When splitting textile up in two different categories, making clothes and weaving carpets, even more jobs are created. Each of the functions can be divided into multiple sub-processes. For example, carpets need to be pre-processed, then dyed, woven, and can then be sold. This way a large variety in jobs is created. For each of the functions water is only needed in one part of the process. To calculate how much water this is, first the area the function needs is determined for each part of the process. Then the amount of water needed for that workshop is determined. The results can be found in the table.

	Function	Module size	Amount of water needed	water storage	Result
weaving carpets	pre-processing	10X5			
	dyeing room	10X5	9765 L/d - 3 564 225 L/y (Or 45 L/d - 16546.7 L/y)	4mX4mX1m = 16m3	
	weaving room	10X5			8 machine woven carpets per day (or 1 hand woven per month)
dyeing fabrics	pre-processing of fabric	5X5			
	dyeing room	5X5	8.4L/d - 3,079.7L/y	3mX1mX1m= 3m3	5m2 dyed cotton per day
	Tailor	5x5			
crops	field	10X10	2.2L/d - 800L/y	0.8mX1mX1m = 0.8m3	
	sorting room	5X5			40kg/y

Figure 39: Water needed for different functions

Program of Requirements

The program of requirements is based on the vision. This means making a bazaar where water is utilized in certain product chains. Research shows that carpets and fabrics are common products produced in Syria. As the production (dyeing) of these products requires water, they are a perfect fit for the bazaar. Yarn is imported and pre-processed in the pre-processing workshops. The fabrics and threads are dyed using water and made into final products in the carpet and tailoring ateliers. The final products are sold in the shops.

Figure 40: Dyeing fabric. Maiwa, 2018

Besides these chains, the bazaar is home to an agricultural product chain as well. In contrast to the other chains, this chain does go from raw material to product sold to consumers. Food is grown on the fields, processed and either sold by the greengrocers or used to cook meals in the restaurants.

Based on the skills and interests of the people in Al Zaatari, the amount of modules per function are calculated (UNHCR, 2017). The sheet used for calculations is shown in appendix A. The employment opportunities of the bazaar are based on the amount of potential employees living on the plot. It is assumed that the amount of job opportunities in the rest of the camp are around the same in the busy 'shopping' area's and that it's lower in the 'living' area's.

Category	Function	Open/closed	Daylight	Area (m2)	Minimum clear height (m)
Infrastructure	Merchant road	Open	No	25	-
	Sub-road	Open	No	25	-
Carpet	Pre-processing	Semi-open	Yes	50	3
	Dyeing	Semi-open	Yes	50	3
	Weaving	Semi-open	Yes	50	2.7
Shops	Carpet shop	Semi-open	Yes	50	2.7
Fabric	Pre-processing	Semi-open	Yes	50	3
	Dyeing	Semi-open	Yes	50	3
	Tailor	Semi-open	Yes	50	2.7
Shops	Clothing store	Semi-open	Yes	50	2.7
Food	Field	Open	Yes	25	-
	Food processing	Semi-open	Yes	50	3
Shops	Greengrocers	Semi-open	Yes	50	2.7
Shops	Restaurants	Semi-open	Yes	50	2.7
Essentials	Market square	Open	No	150	-
	House	Semi open	Yes	50	2.5
	Courtyard	Open	Yes	25	-
	Bridge	-	No	25	-
	Pavillion	Semi open	Yes	25	2.7
	Well	Open	No	25	-

Figure 41: Program of requirements

Connectivity

Bubble diagram
As shown in the bubble diagram clusters of workshops are connected with each other. Workshops are connected to their shops and shops are connected to squares. Houses are standing more on their own.
In the depth chart it can be noticed that shops cluster around the square. The market square is the first function reached when entering the bazaar. Workshops lay more towards the back.

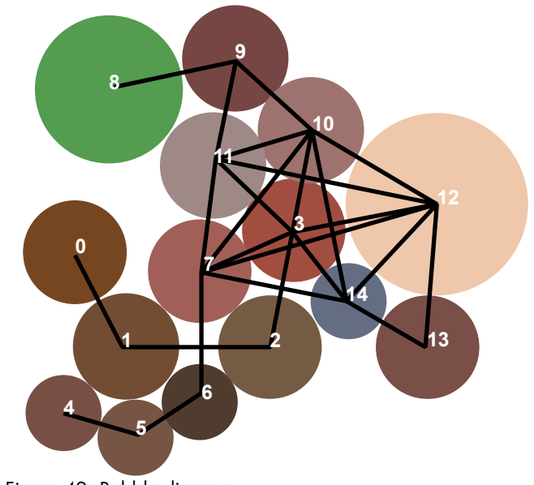
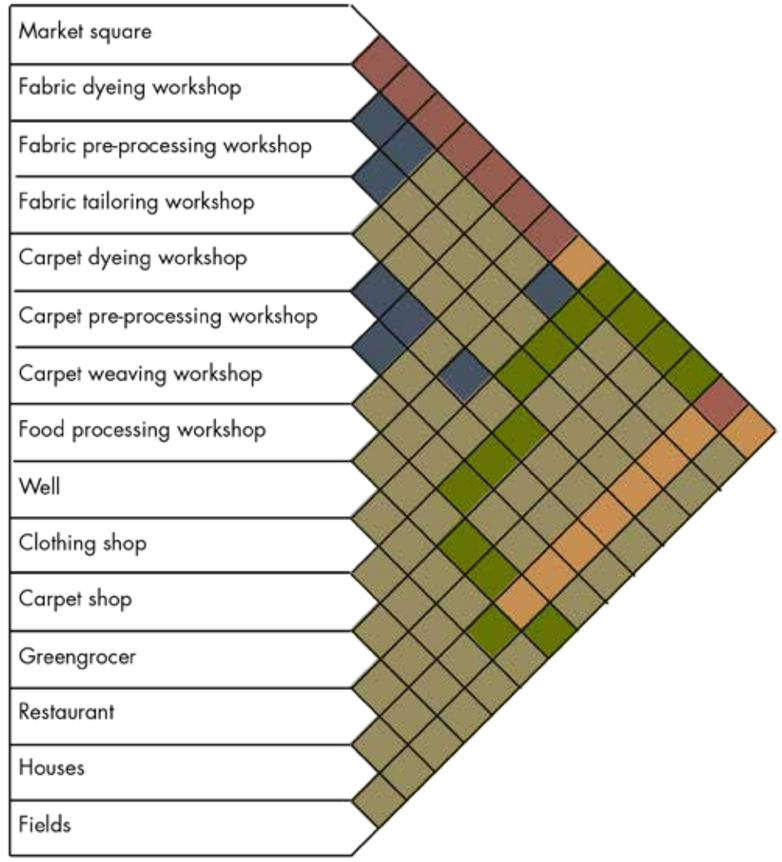


Figure 42: Bubble diagram

REL chart
To ensure an easy process of making carpets and clothes the workshops should be clustered per function. After making products, they are brought to the shops to be sold, so it is important the shops and workshops are somewhat close to each other. To ensure customers do not need to walk too far between shops, they are mostly clustered around market squares. These squares will also give room to market stalls from other parts of the camp. People living in houses often want a little less noise and privacy, so for this reason houses are not located around squares or close to workshops.



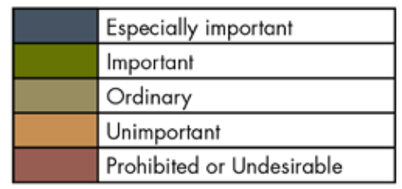


Figure 44: REL chart

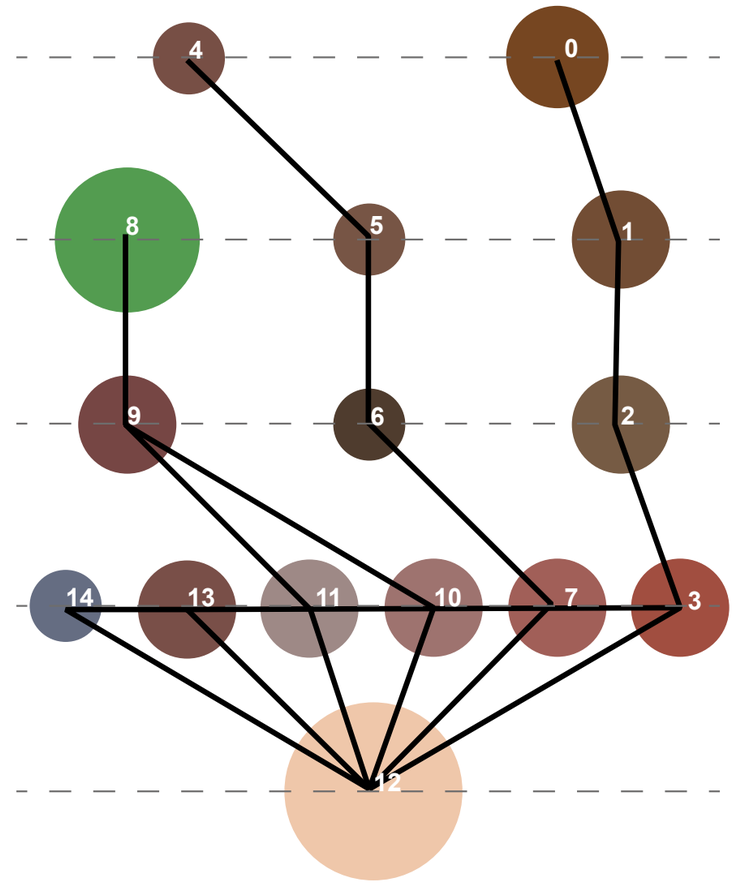


Figure 43: Depth chart

0. Pre-processing carpet
1. Dyeing carpet
2. Weaving carpet
3. Carpet shop
4. Pre-processing fabric
5. Dyeing fabric
6. Tailor
7. Clothing store
8. Field
9. Food processing
10. Greengrocers
11. Restaurants
12. Market square
13. House
14. Wells

Design game

Configurations are generated by playing a design game. The game consists of making rules, testing them and adjusting or adding them. The flowchart shows that this process does not have an end since rules can always be improved.

Up until this point, the rules that are used for the configuration presented in this report are stated below. An elaboration on these rules is given as well. The rules are tested by physically placing modules on a grid and by programming the rules in Python. A physical model is shown containing a possible configuration.

A score is given to the final configuration. The first part of the score is based on how many functions are placed according to the rules. The second part of the score is based on how well the attracting and repelling relations are satisfied.

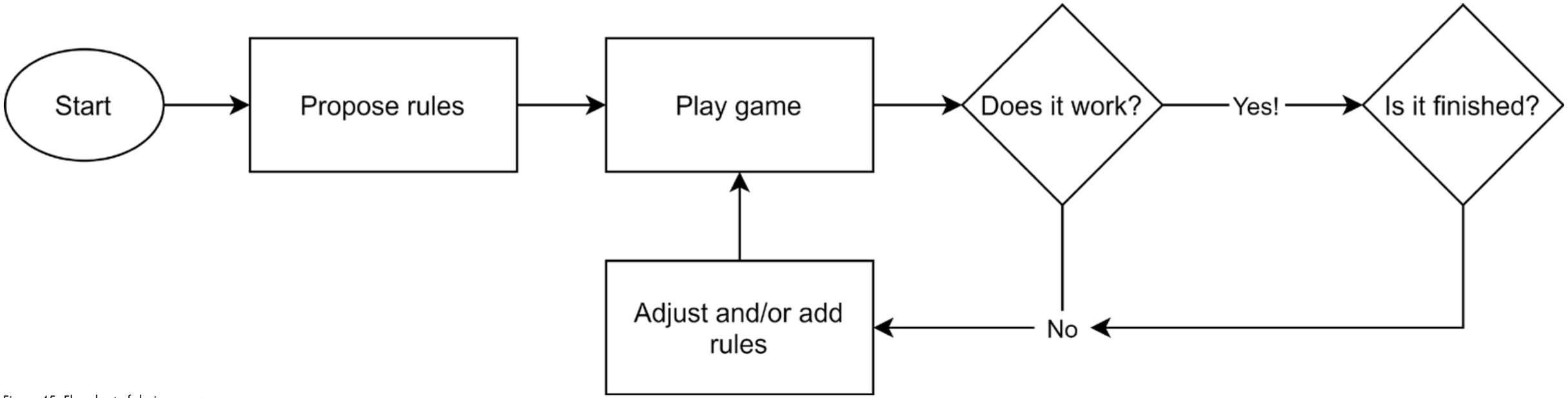


Figure 45: Flowchart of design game

Creating rules

The first part in creating the rules is deciding what they are based on. When analyzing the REL-chart and bubble diagram important connections come forward. These connections should be maintained in the plan.
A decision should be made in what way the different functions are placed. One option is to brute-force different options by placing all functions at the same time and then checking how well it is laid out. Doing this until a good solution comes up. This however, is very hard to test and it can take a long time because of the complexity of the program of requirements.
The other option is to place the functions one at the time. Either placing the same functions at the same time or looping through the functions. This last option is what we went with, trying out both varieties.

Making a paper game

Based on the module size, a grid size is chosen and laid over the plot. The water going through the plot is taken as context and placed first on the plot.
To test the rule set, a paper game is made with a color code for the different functions and a card for each part of the building that should be placed. Each set of rules consists of an order in functions and all the conditions they should meet.

One of the challenges turned out to be how to determine the order in which the functions are placed. First an order was made based on the importance of a function, but this does not consider important connections, so these are sometimes lost. Randomizing the order of placing functions was then tried to avoid too much clustering of functions. But the same problem as before arises.

The first set of rules was short and turned out not to be very specific. The order of placement was determined to be workshops, shops and then houses. But the plot was too large for all functions, and it created a hard separation. This order was tried with and without large market square in the center of the plot. The square was too large and did not add value to the building.

For the second iteration of the rules, more shops were added to improve the balance between houses and shops. Houses were more evenly spaced to form courtyards between them. A problem arose because a first floor was added. Stairs need to be placed to reach the first floor, but when placing them early on it is possible they are not leading anywhere. If they are placed later, there possibly is no space leftover for them.

At the third iteration even more functions were added. Smaller squares are added, and shops are located around them.
After this iteration, the rules seemed to work fine, but still not everything was made explicit. As a human it can easily happen that cards are just put down where it seems to make sense, while that may not be the most logical location. For that reason, at the fourth iteration the rules are made more explicit.

Then for the last iteration, some changes were made to the design, the roads turned out to be too wide and it was more a series of buildings than one building. To solve this, bridges were added to connect the different parts. Also, fields and courtyards are assigned to the open spaces.


Figure 46: Trying rules iteration 1 - with square


Figure 47: Trying rules iteration 1 - without square


Figure 48: Trying rules iteration 2


Figure 49:Trying rules iteration 3


Figure 50: Trying rules iteration 4


Figure 51: Trying rules iteration 5

- Base plate
- Road
- Water
- Well
- Workshop
- Shop
- House
- Square
- Stair
- Windtower
- Bridge

Programming Rules

When testing the rules, we made a paper game and placed the different functions ourselves. This always gives a bias when implementing rules, so to test whether they really work, we also wrote code for a part of the rules. The code can be found in appendix B, it consists of two parts. In the first part water is manually filled in, and roads and wells are placed with for loops.
In the second part, squares are placed with the find.neighbour function. In the rules it is stated that squares should be six voxels large, be placed next to a road and not on top of an existing function. In this case "next to" does not mean diagonally from a point, but only to the left, right, top or bottom of a voxel. So, in the code a Neumann neighbourhood was used.

From the plot, which includes the roads, wells and stairs, some information was extracted:

- Voxels with available space, everywhere without already placed functions.
- All voxels next to roads.
- All voxels not next to roads.

The voxels of the square where then placed one by one, because in this way the orientation of a square can be based on the context. In the examples a square of 4 by 4 voxels is placed.
For each voxel, all available locations are found, and one voxel is randomly chosen from this list. The available locations are found through multiplying the needed maps together. In the maps, all dark green is a 1 and all white is 0, this ensures a chosen location meets all necessary conditions.

Voxel one
The conditions for voxel one are:
- it should be on an available voxel
- It should be next to the road
This gives a lot of possibilities, and one of these possibilities is chosen.

Voxel two
The conditions for voxel two are:
- it should be on an available voxel
- It should be next to the road
- It should be next to voxel one
This only gives a few possibilities because voxel one has only four neighbours.

Voxel three
The conditions for voxel three are:
- it should be on an available voxel
- It should not be next to the road
- It should be next to voxel two
This only leaves one possible location open.

Voxel four
The conditions for voxel four are:
- it should be on an available voxel
- It should not be next to the road
- It should be next to voxel one and three
This only leaves one possible location open.

Conclusion
With the code, each time it runs again, it finds a different location for the square. This almost always works properly. Only when placing a voxel, it does not check whether there is enough free space around the voxel to place the rest of the voxels. This can be improved in the script. The next steps in writing this script would be to include a for loop to place multiple squares at once. Most other functions can be placed using the same method. But some need a different method, for example when going to the first floor how to determine what voxels can be reached.

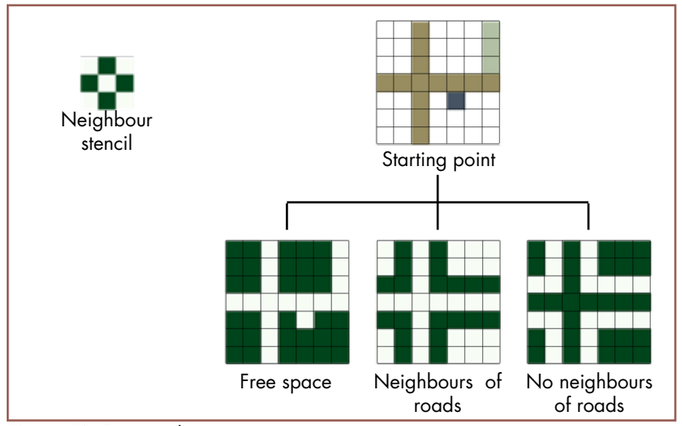


Figure 52: Base and context maps

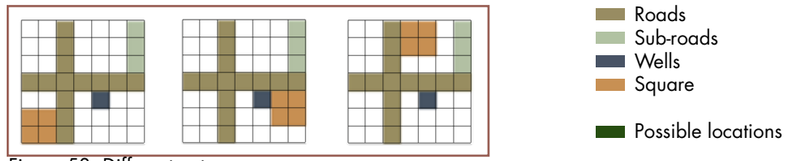


Figure 53: Different outcomes

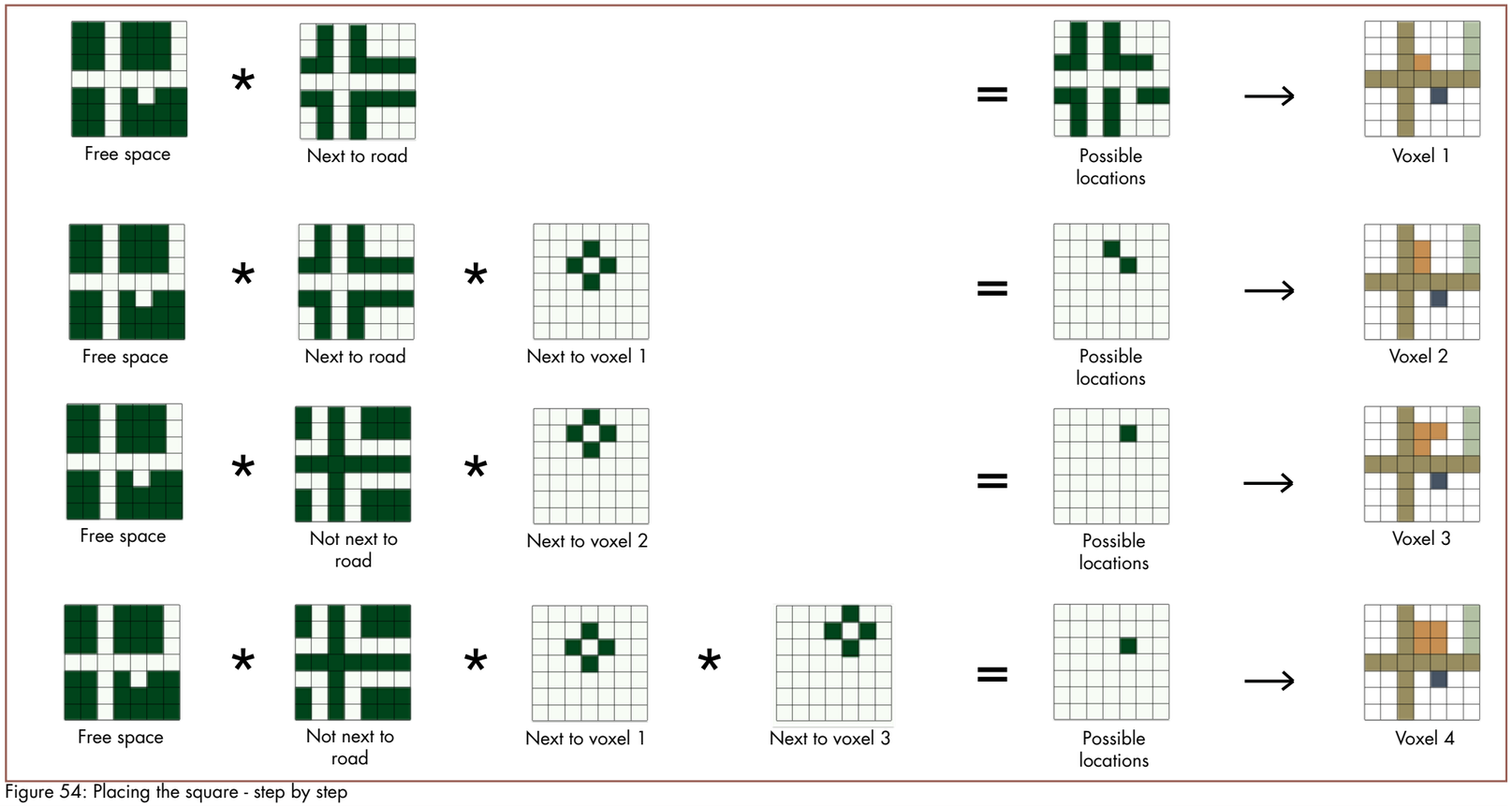


Figure 54: Placing the square - step by step

Current Rules

Water Canals

Water canals are manually placed on the plot. The existing flooding areas (height map) determine the location.

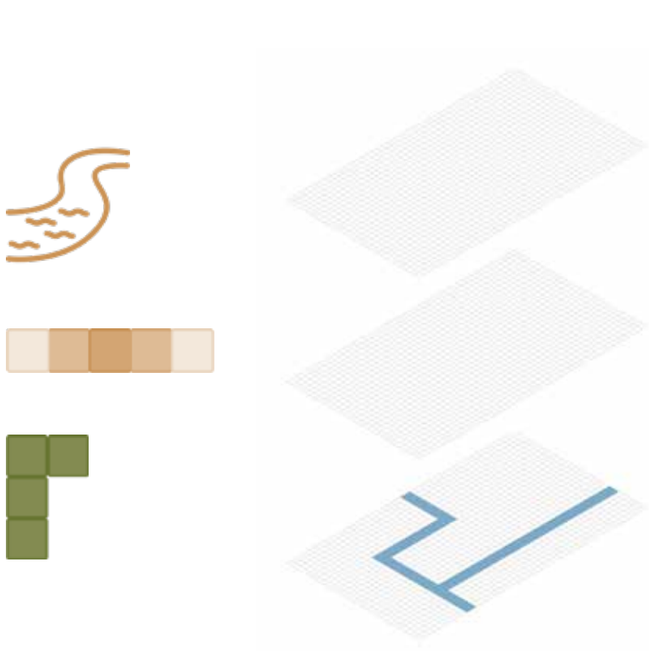


Figure 55: Placing water

1. Place water canals (width = 1 pixel)
- Level = -1
- Water goes from high to low.
- Canals follow the height map.
- The amount of bends should be kept as minimal as possible.

Marchant roads

The merchant roads connect the bazaar with surrounding infrastructure. Most functions that are part of a product chain will be placed around these roads, hence the name. To have the biggest benefit of the water, the roads are placed near as much water as possible.

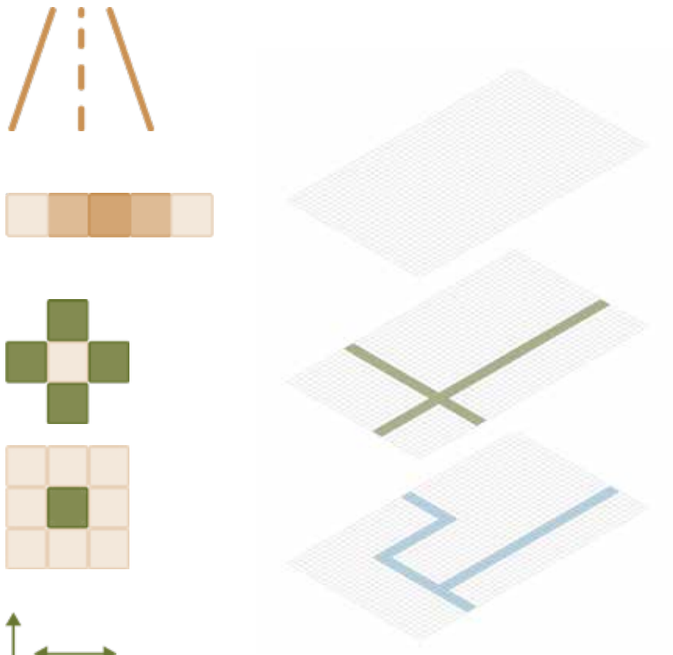


Figure 56: Placing marchant roads

2. Place merchant roads (width = 1 pixel)
- The roads are straight.
- One horizontal road and one vertical.
- Both roads should be connected to the longest waterway in said direction.
- The roads should be on the side of the canal that is most central regarding the plot.

Sub-roads

To prevent spaces not being reachable extra roads are placed. They are located approximately 5 voxels apart, because this is the optimal space for placing other functions.

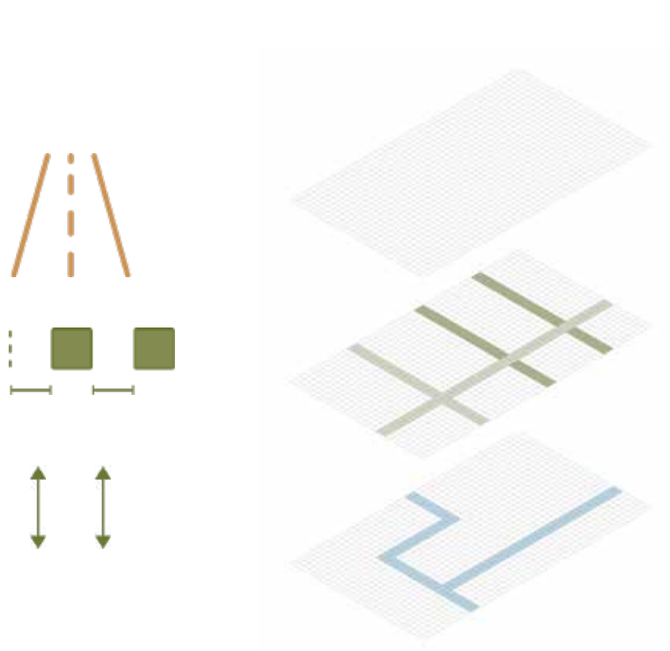


Figure 57: Placing Sub-roads

3. Place sub-roads (width = 1 pixel)
- The roads should be straight and run vertically.
- Two roads should be placed.
- Space between roads and plot boundaries is evenly divided.
- The sub-roads should not cover more than 2 canal modules.

Current Rules

Wells

Throughout the plot water is needed, especially by the water workshops. Wells must be placed on top of the canals and the space between should be evenly divided. This makes the wells most accessible, taking into account the shape of the canals.

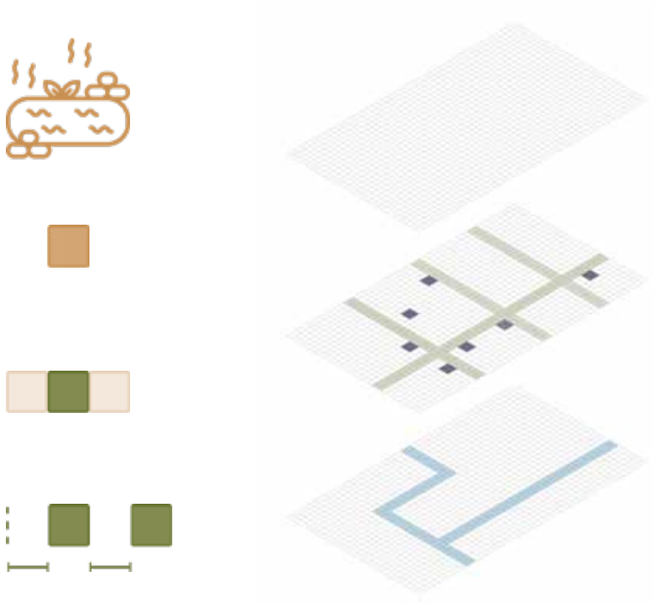


Figure 58: Placing Wells

Squares

Squares form small centres surrounded by shops. To get an equal distribution of shops throughout the plot, the squares should be evenly divided.

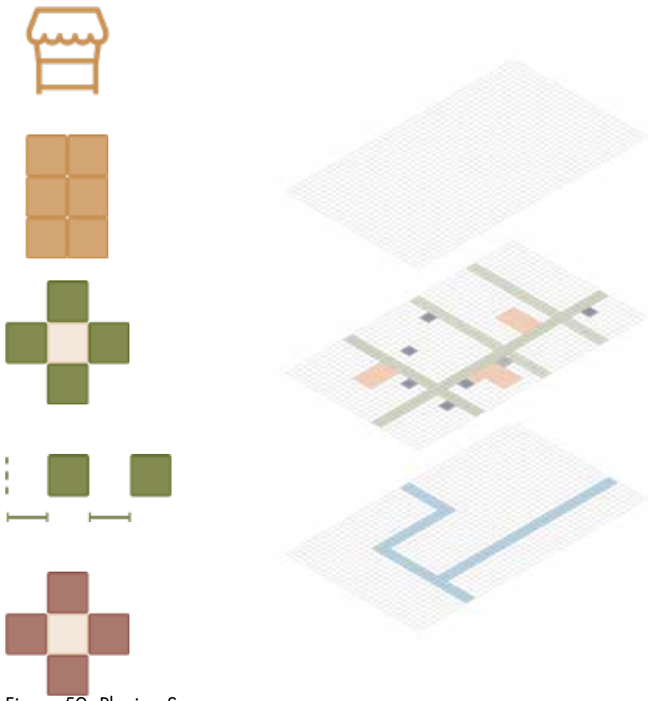


Figure 59: Placing Squares

Stairs

The configuration will have a first floor because not all modules can be placed on the ground floor. To reach this floor, stairs are placed. The functions located on the first floor are houses and food workshops. The houses seek for calmness and privacy, which is found on the first floor. The food workshops have a lower intensity looking at transporting products since crops can't be harvested every day. Two stair modules are placed per building block to create the best connectivity between functions.

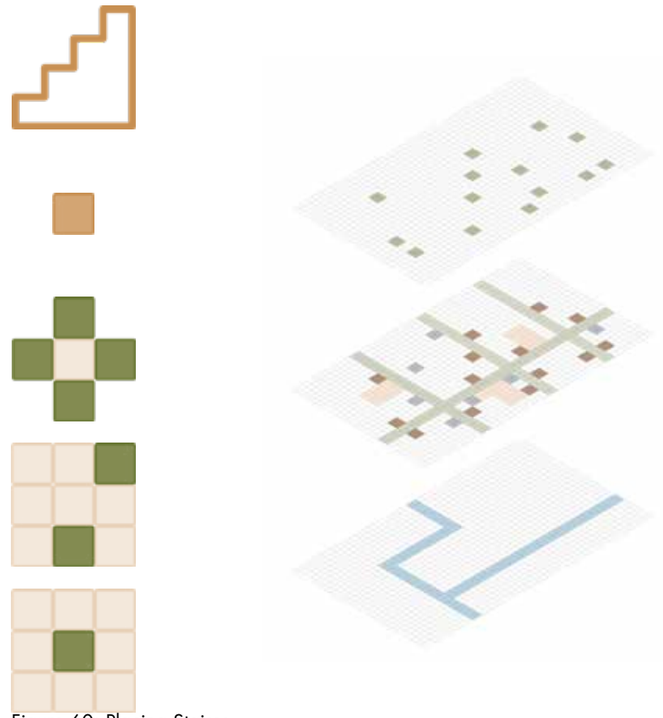


Figure 60: Placing Staires

4. Place wells (1 pixel)
- Place wells on top of canals.
- Total of water workshops x 2, evenly divided.
- Start at 3 pixels from the edge of the plot.

5. Place 3 squares (3 x 2 pixels each)
- Distance between the squares is evenly divided.
- The short side should be connected to the main roads.
- Minimum of 1 pixel between any roads and a square.
- Not on top of wells.
- After courtyards are placed: free pixels connected to squares become a square.

6. Place Stair modules (1 pixel)
- Stairs are placed next to roads.
- Within each building block (in between the roads) 2 stairs are placed.
- The stairs should be placed along the edge of the building block that is the most central. On this edge, place the stairs on the pixel closest to the middle.
- Stairs can not be placed on top of water.

Current Rules

Wind towers

Wind towers do not have to be reached from the roads often, because they are low in maintenance. For this reason, they are not located next to the roads. They are evenly divided over the plot to ensure the building is cooled everywhere.

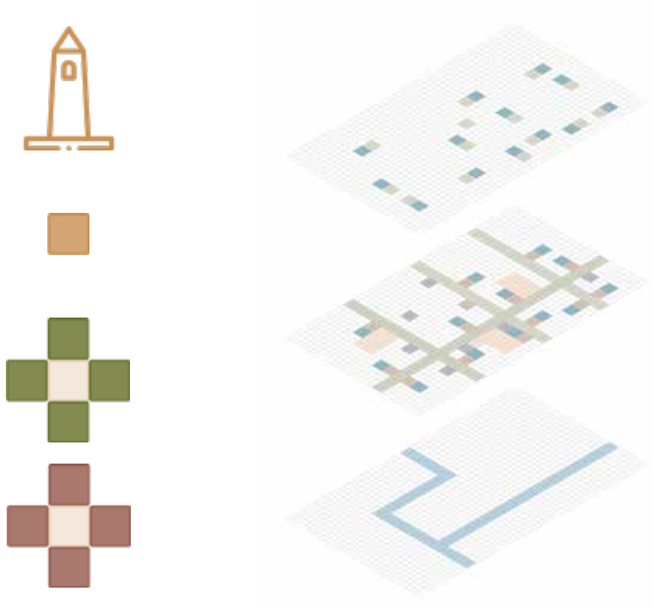


Figure 61: Placing wind towers

7. Place wind towers (1 pixel)
- For each stair module, a wind tower is placed.
- Connect to stairs, at the opposite side of where stairs are connected to the road.

Water workshops

Water workshops need water to function so they are placed next to a well. They should be accessible from the merchant roads. They also generate a lot of noise so they should not be placed near squares. Workshops for fabrics and carpets are evenly divided so each square will be differentiated more.

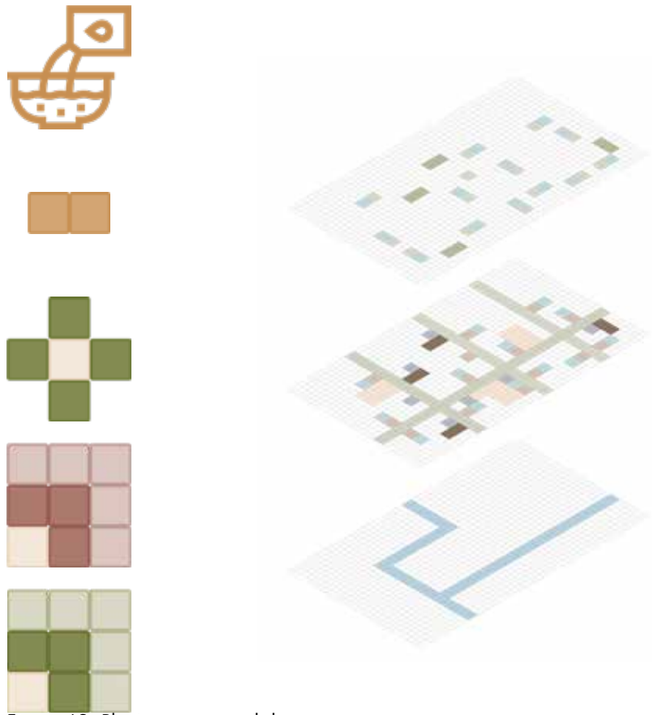


Figure 62: Placing water workshops

8. Water workshops (2 pixels)
- Place as close to a well as possible.
- Maximum of 3 pixels away from a well.
- Short side connected to a road.
- Away from squares.

Non-water workshops

Workshops are clustered together with the water workshop when they are part of the same product chain. Products are constantly moved between these workshops, so the clustered workshops should be placed directly next to each other.

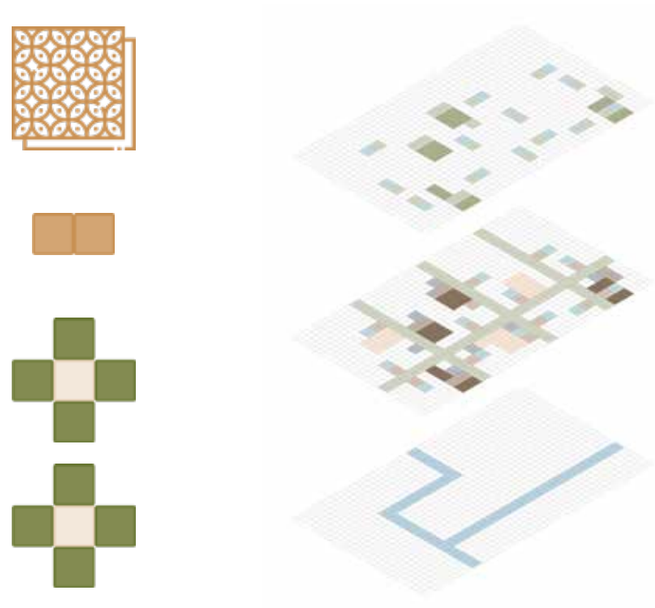


Figure 63: Placing non-water workshops

9. Place workshops (2 pixels)
- As close to the water workshop of the same product chain as possible.
- Evenly divide the workshops of a product chain over the existing water workshops.
- Short side connects to the road.

Current Rules

Shops

Shops are visited by a lot of people, so they are clustered around the squares. This way they can be easily located and accessed by visitors. The distance between the shop and the workshop from which the shops get their products should be minimized for transportation.

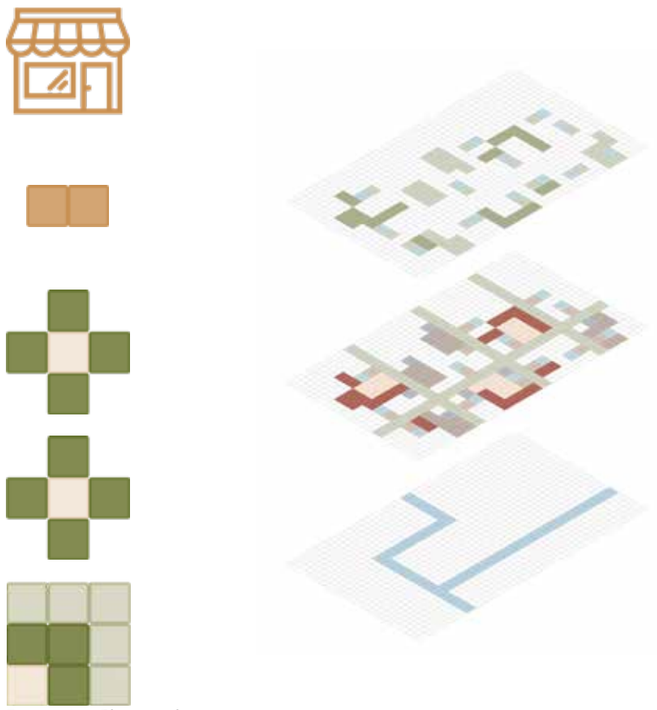


Figure 64: Placing shops

10. Place shops: (2 pixels)
- Connected to or close to squares.
- Short side (entrance) connected to a merchant road.
- Close to the cluster of workshops the shops get their goods from.

Houses ground floor

Houses need more calmness and privacy so they are located away from the squares and workshops as much as possible. However, people want to live close to facilities so the houses are not located at the edge of the plot.

Figure 65: Placing houses ground floor

11. Place houses on the ground floor (2 pixels)
- Attracted by sub-roads
- Attracted by other buildings
- Short side connected to a road
- Stairs as close as possible
- At least five homes should be placed on the first floor.

Bridges

To connect all parts of the building, bridges are placed. Bridges are placed in such a way that all blocks are connected to each other in the most efficient way.

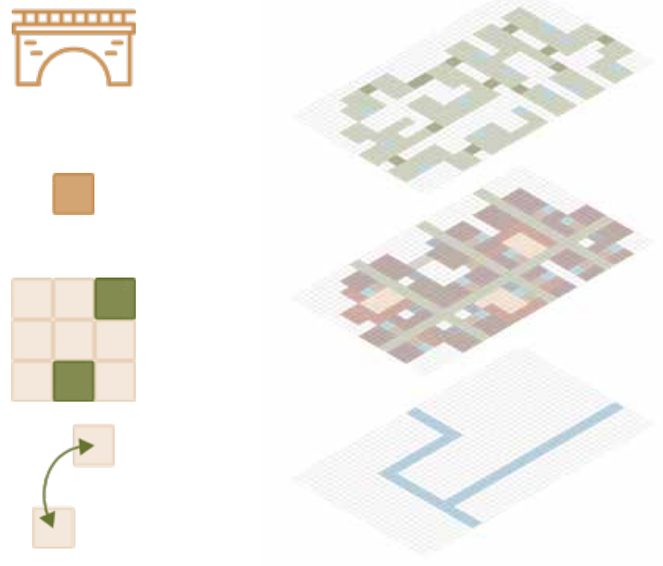


Figure 66: Placing bridges

12. Bridges
- Locate all roofs accessible via stairs.
- Locate all roofs accessible by these roofs.
- Make a continuous line that connects all building blocks with each other in the most efficient way.
- Place bridges where there are no accessible roofs.

Current Rules

Houses first floor

The houses on the first floor follow the same rules as the houses on the ground floor.

Figure 67: Placing houses first floor

13. Place houses on the first floor (2 pixels)
- Attracted by sub-roads
- Attracted by other buildings
- Short side connected to a road
- Stairs as close as possible
- At least five homes should be placed on the first floor.

Food processing

Food processing workshops are used less frequently and thus do not have a lot of specific demands. This is why they are placed on the first floor. The food processing workshops follow the same logic as the other non-water workshops.

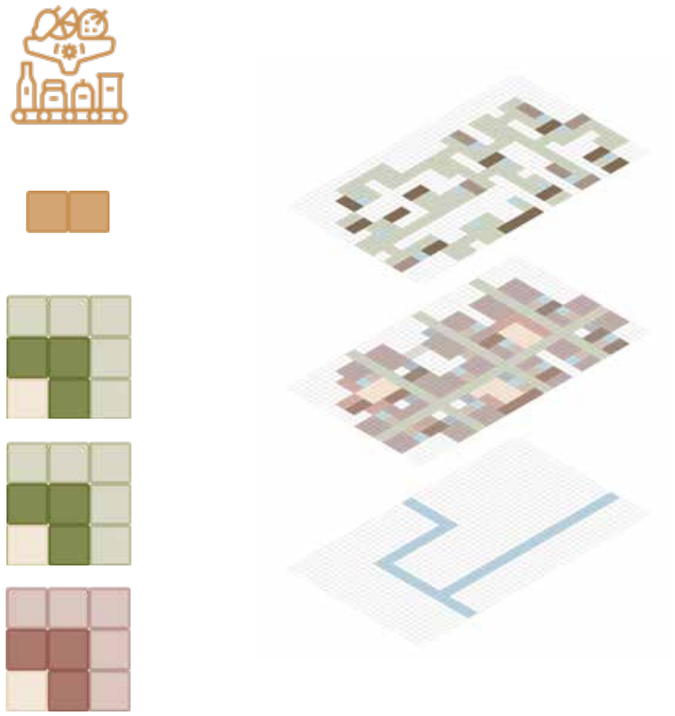


Figure 68: Placing food processing

14. Place Food processing (2 pixels)
- Located on the first floor
- Within 3 pixels of stairs.
- Attracted by shops
- Repelled by homes.

Courtyards

Courtyards are part of Syrian culture. It is an outside living space that is located next to a house. As many houses as possible are provided with a courtyard. Courtyards are shared between houses with a maximum of four houses.

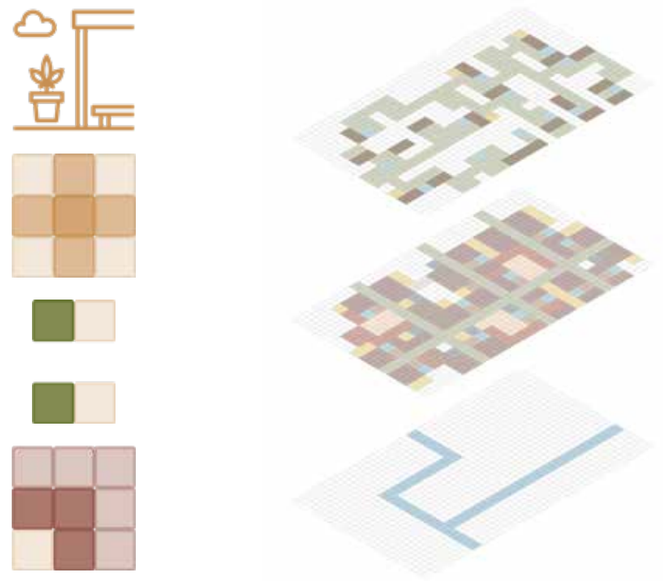


Figure 69: Placing courtyard

15. Courtyards
- If possible, every house gets a courtyard at the opposite side of the door.
- Else, the courtyard is located as far from the front door as possible.

Current Rules

Fields

All voxels on the ground floor that don't have a function yet become fields, as long as they're not surrounded by buildings and roads. This way all fields have a adequate connection to the infrastructure and harvested crops can easily be transported.

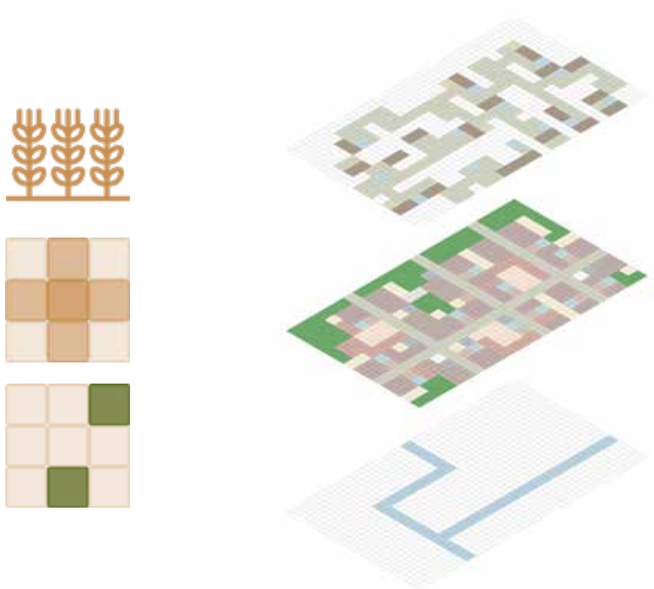


Figure 70: Placing fields

16. Fields
- All pixels that have no function and are not surrounded by buildings become fields.

Pavilions

To make the public space on the second floor more attractive we add pavilions. They stand alone and should not be located above canals since the weight will be too much.

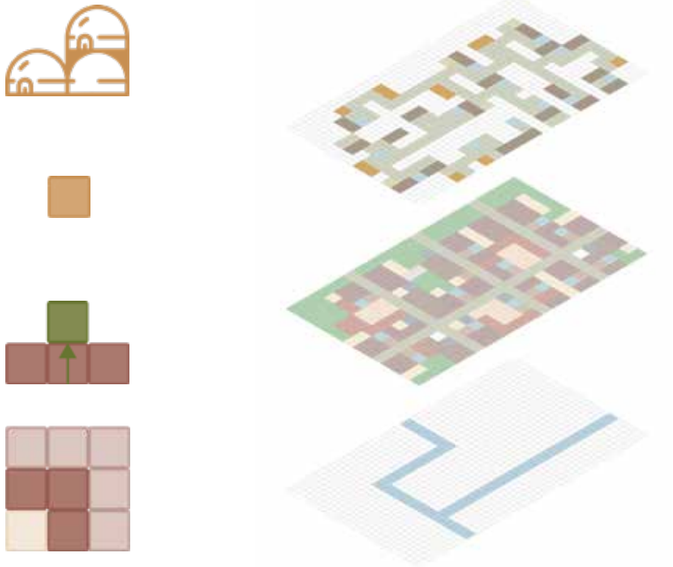


Figure 71: Placing pavilions

16. Pavilions
- Don't place above canals
- Don't place next to an existing building on the first floor
- Don't close off paths
- Place a maximum of two per building block

Full plan

The full plan contains all modules that need to be placed. As can be seen there is a lot of public space on the first floor that can be accessed easily via the stairs.

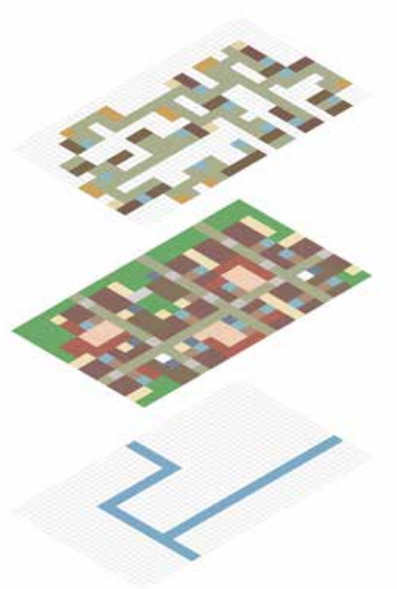


Figure 72: Full plan

Current Rules

Floor plan evolution

Figure 73: Link to evolution of the floor plan

Current Rules

The rules of the game create a certain connectivity between all of the functions. The diagram below shows the connections and influences according to the stated rules. Arrows indicate a qualitative connection, double lines show attracting (green) and repelling (red).

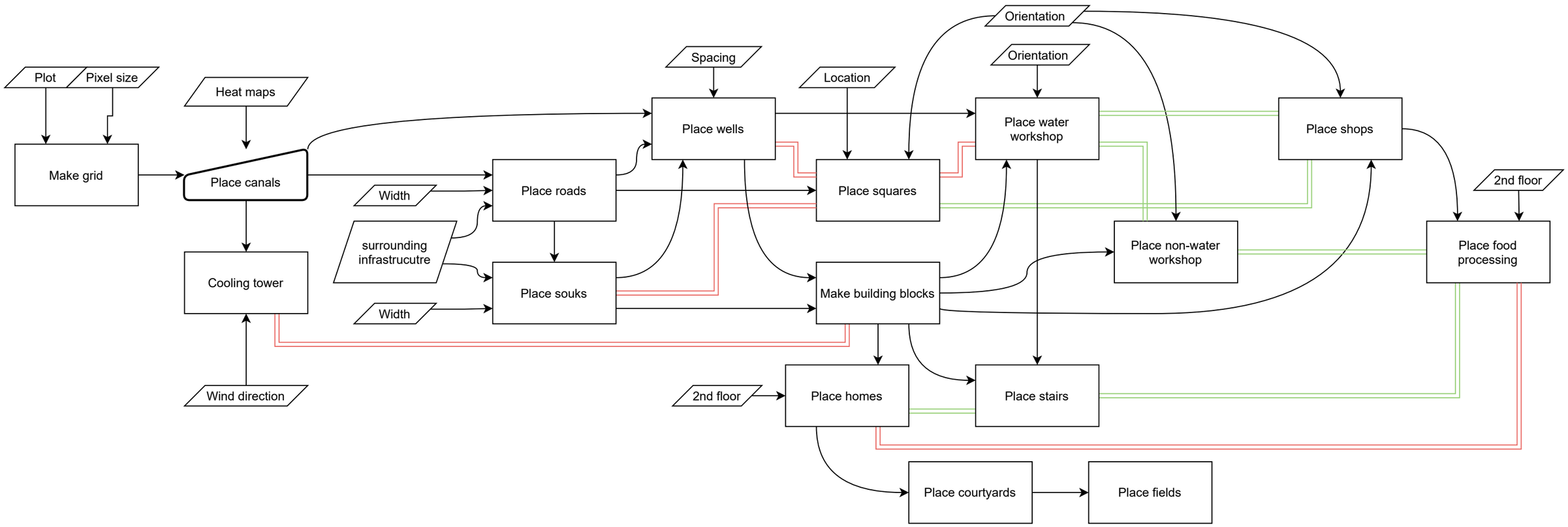


Figure 74: Flowchart rules od the game

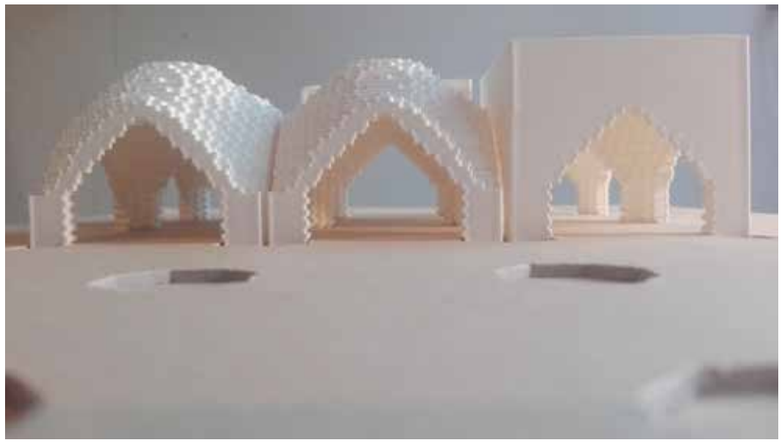
Physical Model

To test the modularity of the modules, a 3D model was printed. The model consisted out of two different modules, one with a flat roof for placing a second floor on and one without a flat roof, that does not have an extra floor on top. Of both modules, multiple were printed.
For the base plate, a flat surface with recesses for the location of the supports was made.

With this model, multiple configurations are tried. It added a lot of value to be able to feel the modules and see the configurations for ourselves. This showed us that the modules and configuration worked in the way we planned it.



Figure 75: Module



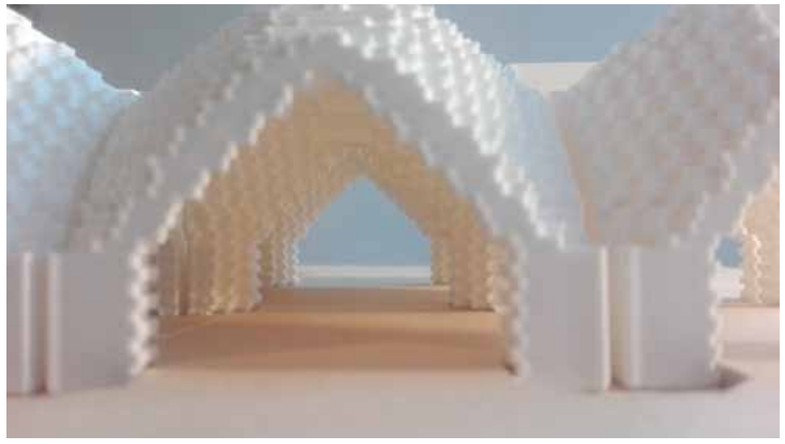


Figure 79: Close-up of possible configuration



Figure 76: Module with flat roof for first floor



Figure 77: Possible street profile



Figure 78: Possible configuration

Water management

In the oasis there are multiple functions which require water. It is brought to the workshops through the underground tunnels leading it out of the camp. The supply of water is a main topic within our project. The waste stream of water is not currently implemented. This is something that could be implemented in the next version of the oasis. Here some thoughts about the water waste are presented.

When the water is used it becomes polluted. The polluted water needs to be transported out of the camp because if it stays in the camp it can cause multiple problems including the spread of diseases or damage to the adobe buildings. One solution to this is making a second canal underground to transport the waste water. This would increase the complexity of the configuration because on top of the underground canals only one story can be built because of the stress concentrations in the bottom columns. Another solution would be to locally filter the water so it can be used again for certain functions and does not need to be transported out of the camp. The problem with this is that filters can be quite expensive and need regular cleaning and maintenance.

Figure 80: Clean water

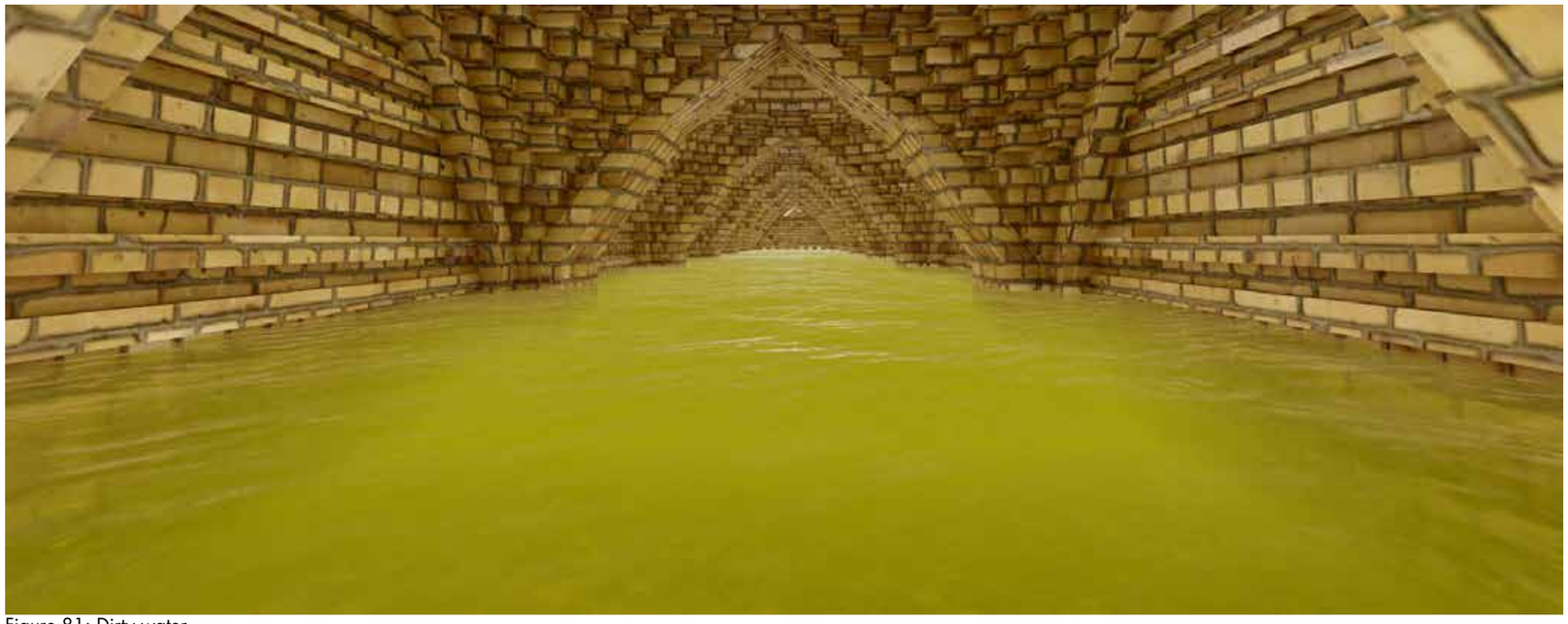


Figure 81: Dirty water

Visuals

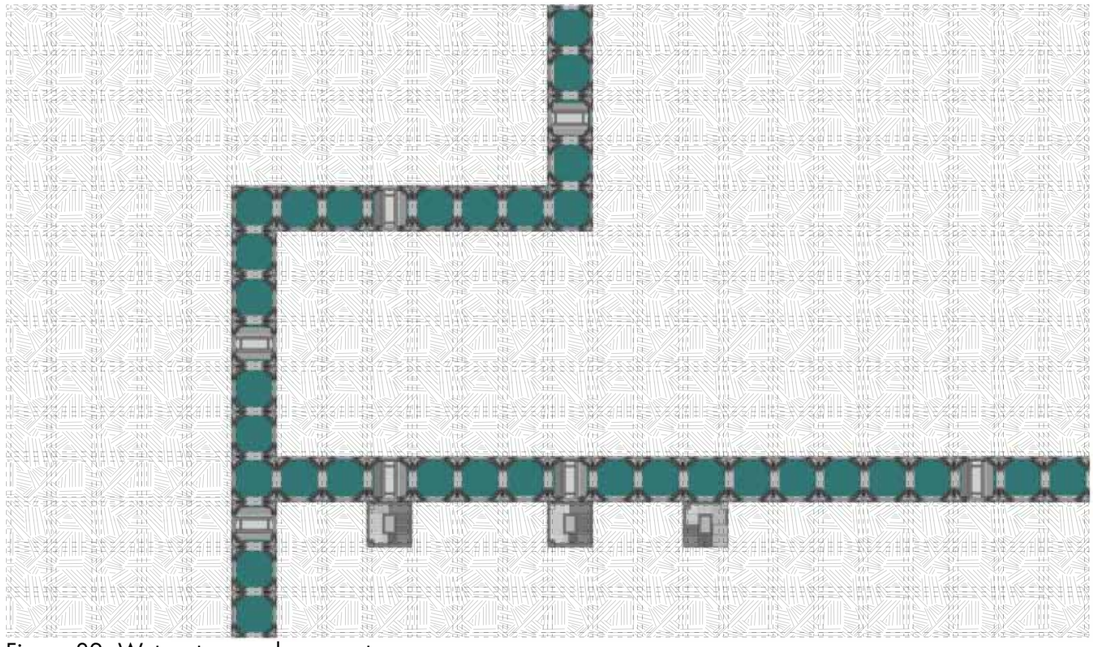


Figure 82: Water storage, basement

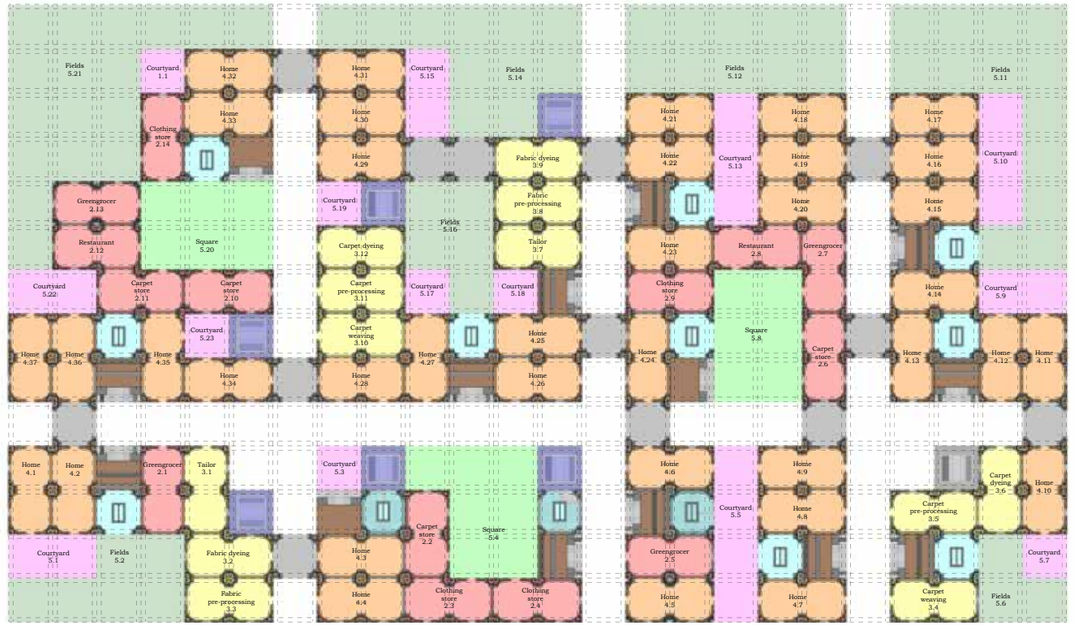


Figure 83: Ground floor, with functions

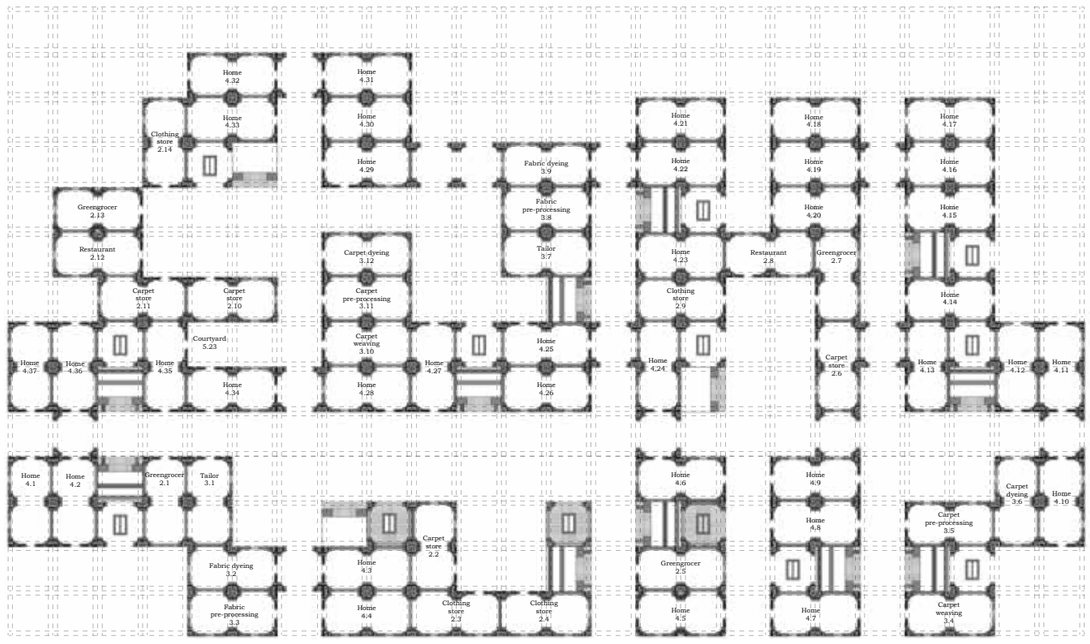


Figure 84: Ground floor, without functions

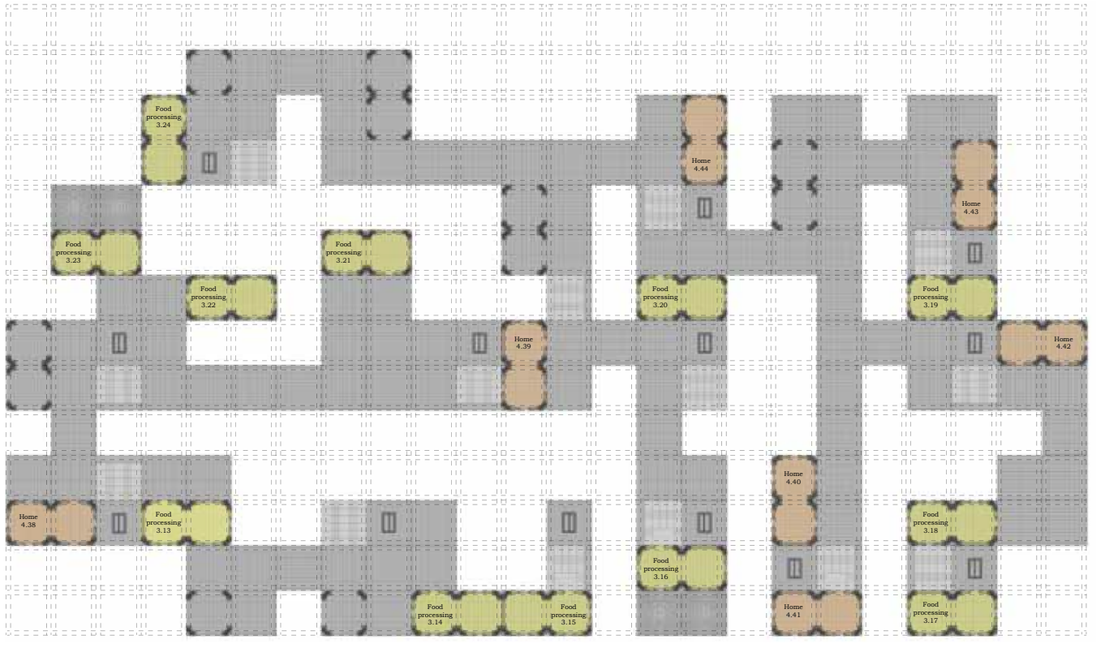


Figure 85: First floor

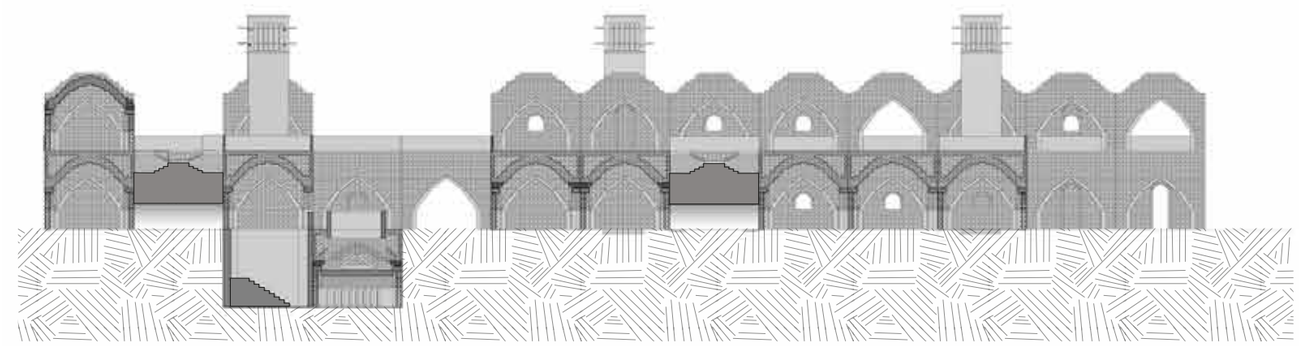


Figure 86: Section North - South

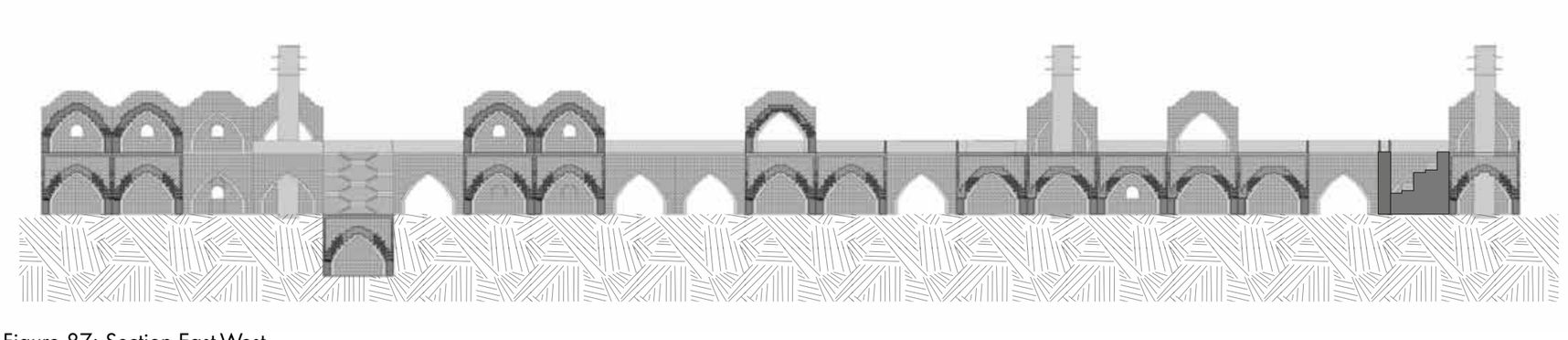


Figure 87: Section East-West

Figure 88: Overview

Figure 89: First floor view

Figure 90: Shop inside view

Figure 91: Market square

Figure 92: Water storage

3_Shaping

This chapter discusses the shaping of the modules. Below a flowchart can be seen which describes the processes in this chapter. First the size of the module and the brick are discussed. These two sizes serve as a guideline for the rest of the shaping, structuring and the construction. After that the tessellation and the placement of the bricks will be discussed. Parts of the flowchart will be discussed in the corresponding sub chapters.

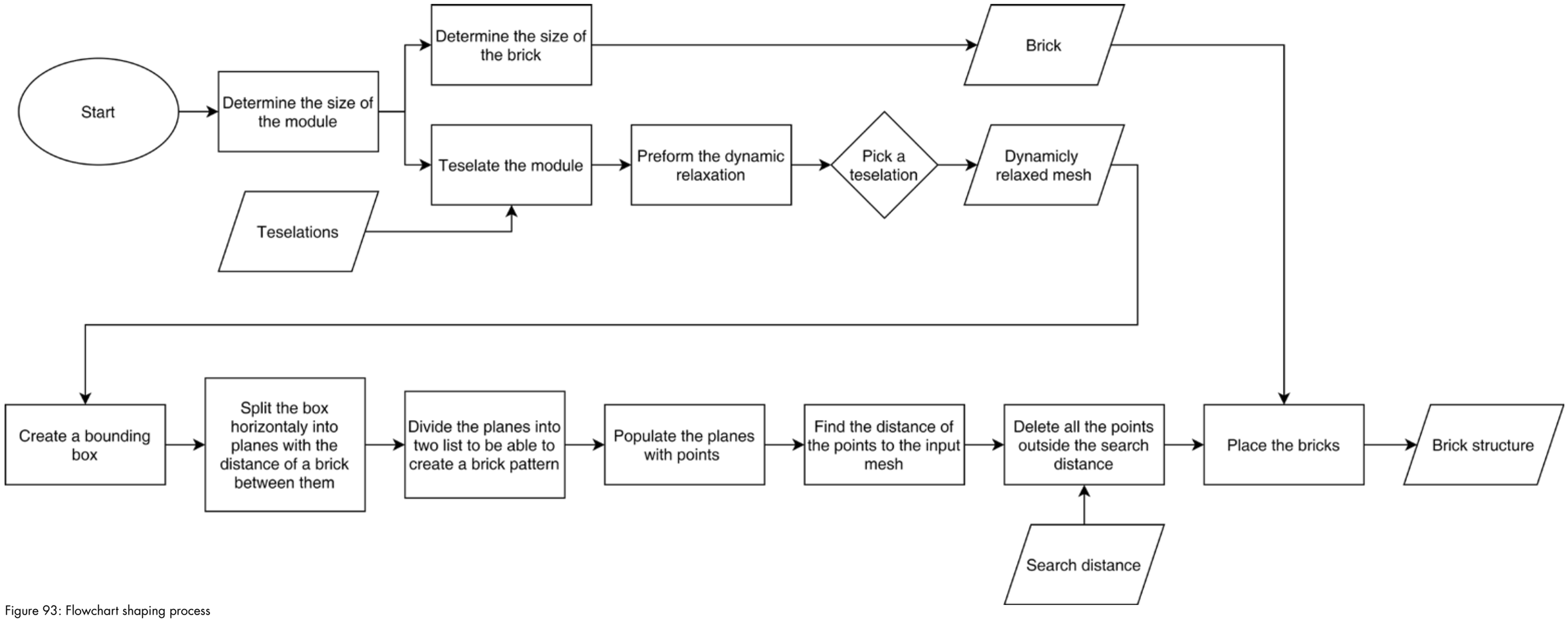


Figure 93: Flowchart shaping process

Module Size

Quite early in the process we decided that we wanted only one module size. This is because we could make the building and the plot completely modular. This means that the module size needs to meet a couple of requirements:

- Big enough to house a function in only one module.
- Small enough to still have a reasonable size after modules are linked.
- Must be self-standing to give as many possibilities in the plan as possible.
- The span of the modules must me achievable with adobe bricks without the need for a very large span.
- The size of the module will also be the width of the street. Therefor the size must make sense as a street width

After some discussion the final dimensions are 5 meters by 5 meters. This for fills all of the previously mentioned requirements. The height of the module followed later as a result of the tessellation and the dynamic relaxation. The goal was to keep the module relatively low. This is because otherwise it could cause problems with the stairs that were designed later.

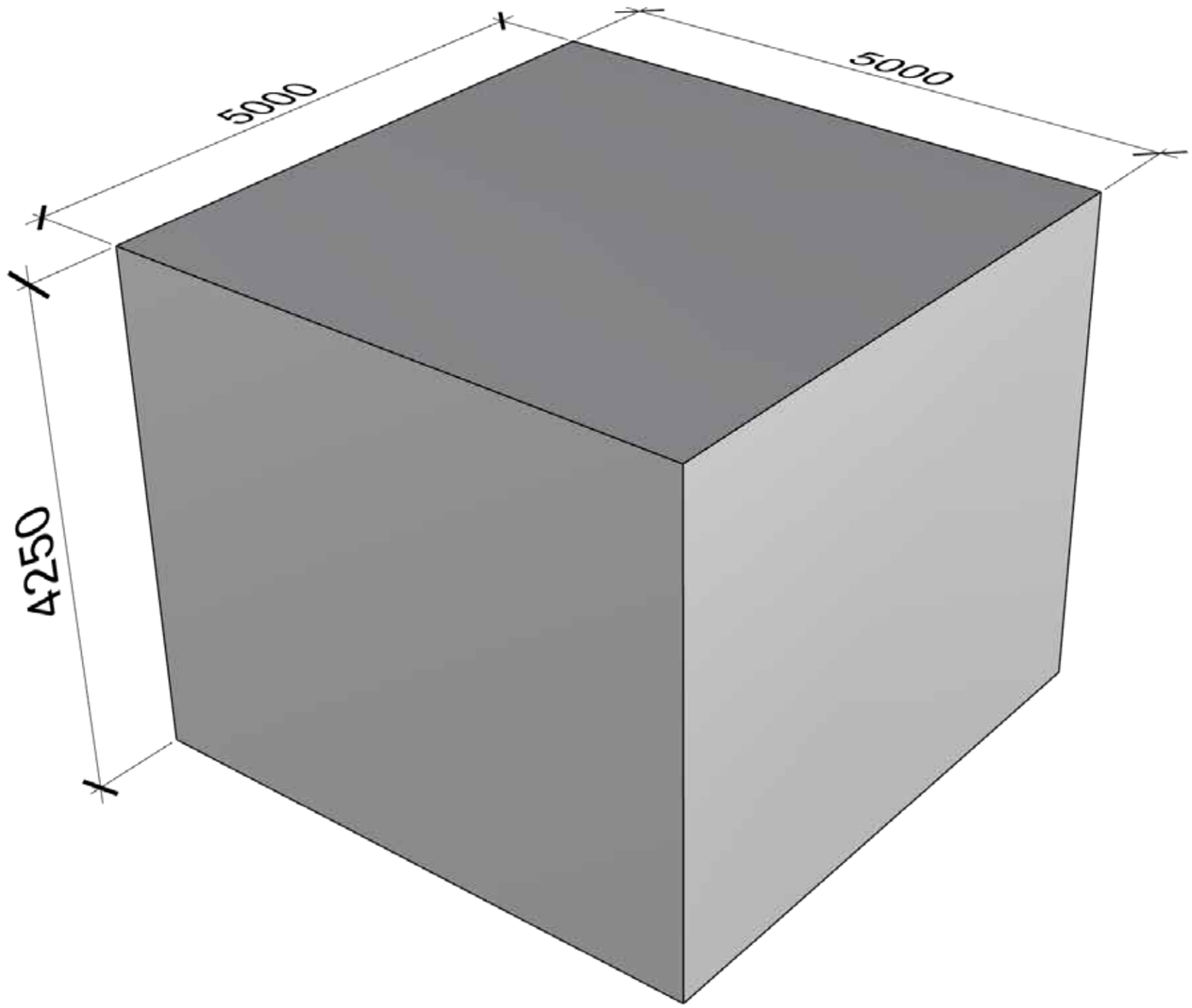


Figure 94: Module size

Brick size

After determining the module size, the next step was to think about the size of a brick. We considered making multiple brick sizes to make the constructions easier to make. After some discussion it was decided that one brick size would be more practical because of the limited skill of the builders on site.

Having only one brick size allows for easier production of the bricks in the beginning of the process. Besides that it provides a fool proof product chain on the building site. As the plan is to let the inhabitants of the camp produce the buildings themselves, it seemed logical to us to make the production as easy as possible. Most of the people working on the project are most likely not skilled builders.

The size that was decided on is 250 mm x 125 mm x 125 mm. The length and the width of the brick fit in multiples in the module of five meters, making the need to cut the bricks a lot lower. The length is twice the size of the width. This makes it so that the bricks can be stacked to make a square but are also wide enough to create other patterns. The height of the brick is the same as the width. This was done so the brick would not cause any problems when rotated. This is useful when an arch is created.

At the end of the project we were not so sure about the height of the brick anymore. The width and the length preformed as expected but the height did not work well with the stair modules. This will be discussed further in the reflection.

Besides the standard brick, there are four more building elements. These elements are used in the arches. These elements are a lot larger than the standard brick therefor the time it takes to dry these elements will be a lot longer. These elements are necessary to make the construction a lot easier for the people working on it. Therefore we think it is a good trade-off between having only one element and on the other hand the ease of constructibility.

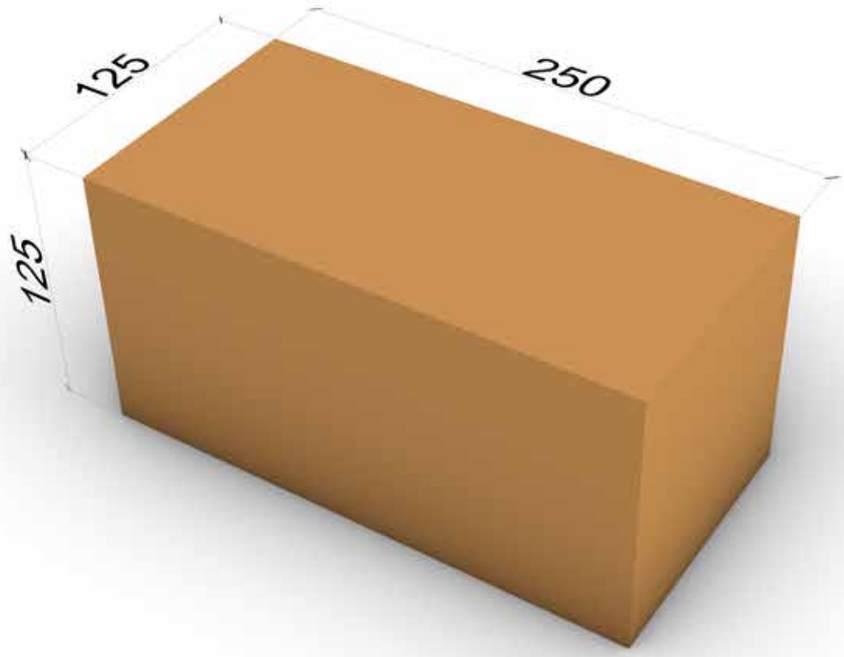


Figure 95: Brick size

Tessellation

The tessellation was done to find the optimal shape for the module. We started by making a tessellation of a regular rectangular mesh to have a baseline to compare the other tessellations to. Eventually we came up with six different ways to tessellate the mesh each time learning from the previous one to improve the mesh.

1. Standard square tessellation: functions as a benchmark to compare the other relaxed meshes to.

2. Diagonal lines between the anchors in the corners: This was done with the idea that if these mesh lines were arches in three dimensions, then the shape would be very easy to construct. The problem with this tessellation is that the edges of the relaxed mesh are straight. In between of these straight lines is a double curved plane.

3. From corners to centre with open centre: This tessellation was created to prevent the conversion of all of the mesh edges in the centre of the mesh. As can be seen this was not very successful because the hole became a lot bigger and the shape became unusable.

4. From corners to centre with closed centre: this mesh is quite similar to number 3 only with a closed centre. In this mesh triangular mesh faces are prevented by making all the parts of the tessellations quadrilaterals. We were very pleased with the results of this dynamically relaxed mesh.

5. Diagonal lines from anchors and between midpoints of the arcs Mesh 5 is a combination between mesh two and four. For some reason this mesh kept on breaking when the dynamic relaxation was performed. We tried multiple things to repair the mesh. After a couple attempts we decided that there was no point in spending a lot of time on this single mesh.

6. Square centre with diagonals from the corners: Mesh 6 is a combination between mesh one and four. In this mesh an attempt was made to keep all of the mesh faces quadrilateral. And keep concentrations of the mesh edges low. Because the resultant dynamically relaxed mesh met all of our requirements and it did not have the mesh edge concentrations the other meshed had, this was our final choice.

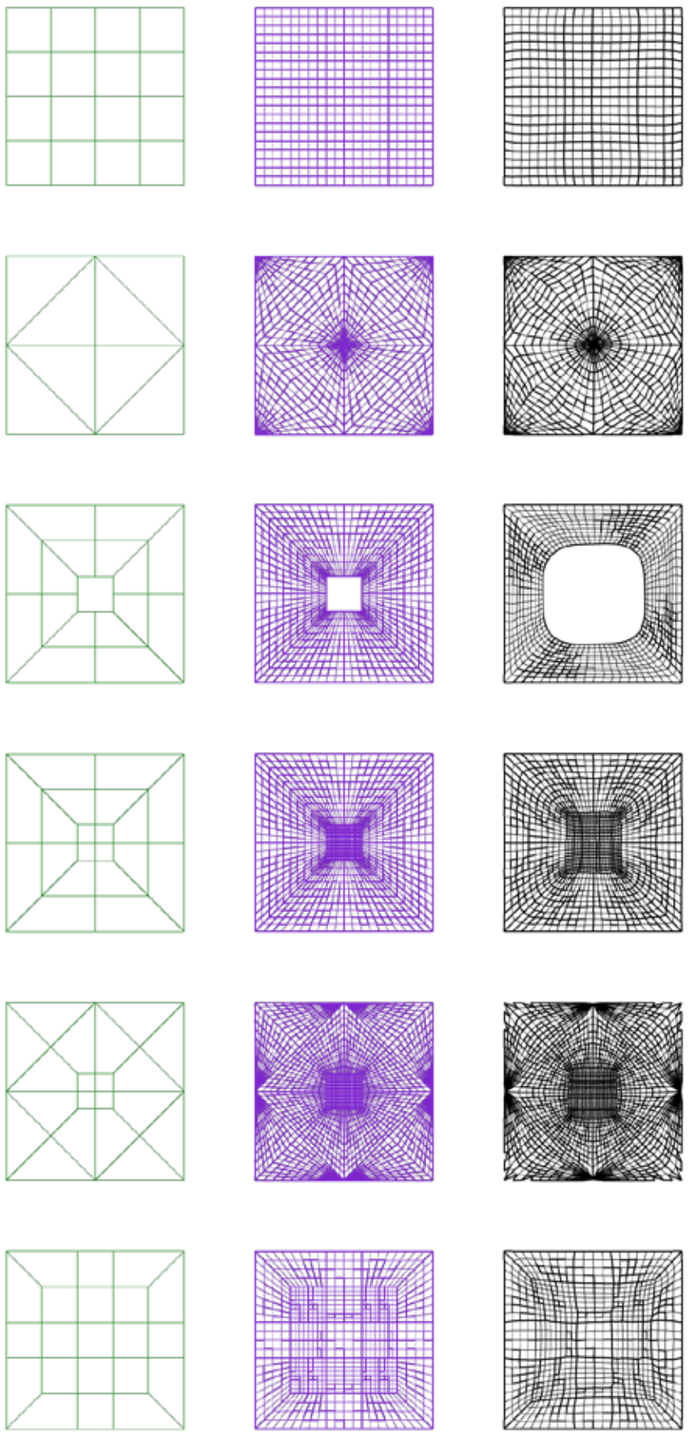


Figure 96: Topvieuw tessellations

Figure 97: Sidevieuw tessellations

Placement of bricks

After a tessellation was found that met the criteria that were set for the mesh, it was time to approximate the mesh with a brick structure that could actually be build. The first version of the brick placing script made arches on the edge of the dynamically relaxed mesh out of bricks and then filled in the rest of the shape with bricks oriented towards the centre of the dome. This way the bricks would have the maximal overlap in direction of the centre. The problem with placing the bricks in this way was that the overlap of the bricks was not guaranteed and some of the bricks could fall out of the construction.

After some discussion another method of placing the bricks was picked. Part of the flowchart explains the process of placing the bricks. Below the steps are described in more detail.

1. A bounding box is created around the dynamically relaxed mesh. The bounding box is then split into layers with the height of the brick. The layers are then split into two layers to get an alternating brick pattern.
2. The layers are populated with points. The distance between the points is the size of the bricks in that direction.
3. The distance from all of the points to the mesh are calculated. If the distance is smaller than a predefined maximum distance the points are kept. Everything outside of the maximum distance is removed.
4. The bricks are placed on the remaining points in the brick pattern that was initialized in step one.

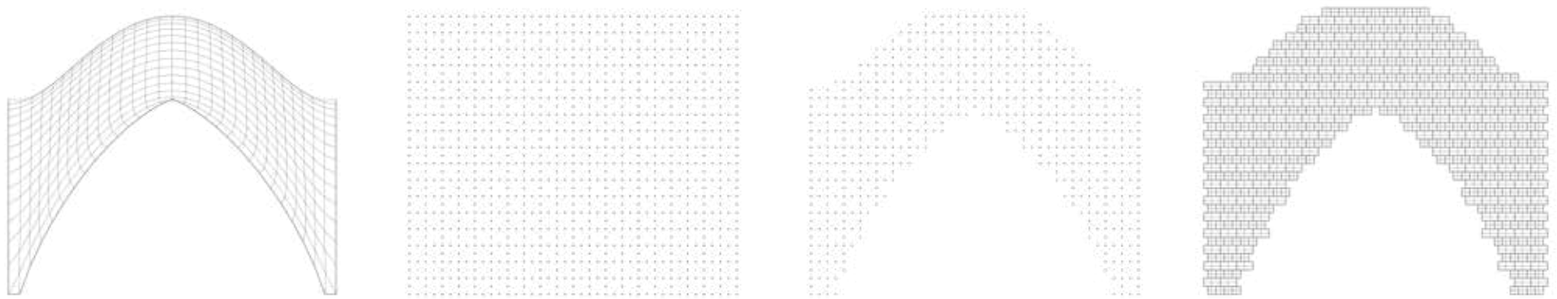


Figure 98: Placement of the bricks

After placing the bricks in this way, some of the bricks where still unsupported. To resolve this we implemented arcs in the round openings on the module. The arcs have two main benefits. The first is that the bricks that were previously unsupported are now supported by the arch. The second benefit is that the arch is built with only seven parts. This means that it can be built by nearly anyone and provides a starting point for the rest of the module.

Despite implementing the arch into the module not all of the bricks were supported. Particularly on the top most parts of the module the inclination of the relaxed mesh was to flat. This meant that the module needed to be checked by hand to see if all of the bricks that were placed would also stay in place. Eventually we decided that we needed two more layers of bricks on top of the module to ensure that the none of the bricks would be able to fall out.

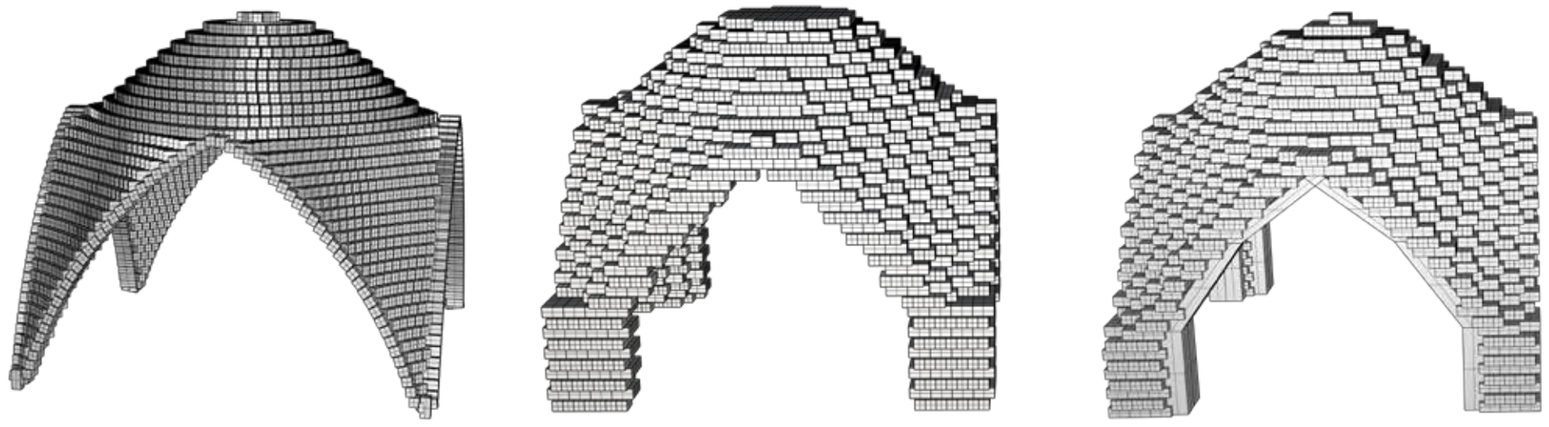


Figure 99: Iterations of the brick placement

Staircase

The stair design is an important part of a floor plan. They connect the ground floor with the 1st floor, but our design also connects different functions on the same floor with each other. With the help of a grasshopper script we were able to produce different possibilities for a design based on the Indian stepwell. Our staircase design always starts in 2 places on the ground floor, on the left and right side of a grid. This allows one function to be placed in front of the stairs or two functions to the left and right of the stairs(see option 1). We made the script so that we can enter the following and get a staircase design: Height of the platforms, Number of steps per ascent, Riser height, Tread depth, max height. The width of the stairs will be determined by dividing the required number of ascents by the grid size (5 meters). The stairs will therefore always end up at the height of the floor above. In the 2 images below it can be seen that if the number of steps per ascent is lower, the width of the stairs also become less wide. Two more options have been added to make the stairs connect even more functions in the floorplan with eachother.

Option 1: If you want to connect the functions on the ground floor with each other, an arch can be added under the stairs. The height of the arch and the width of the arch are adjustable.

Option 2: When no modules are placed on the 1st floor to the left and right of the stairs, an arch is automatically placed above the stairs so that the 2 roofs are connected to each other.

Figure 104: 3D of staircase

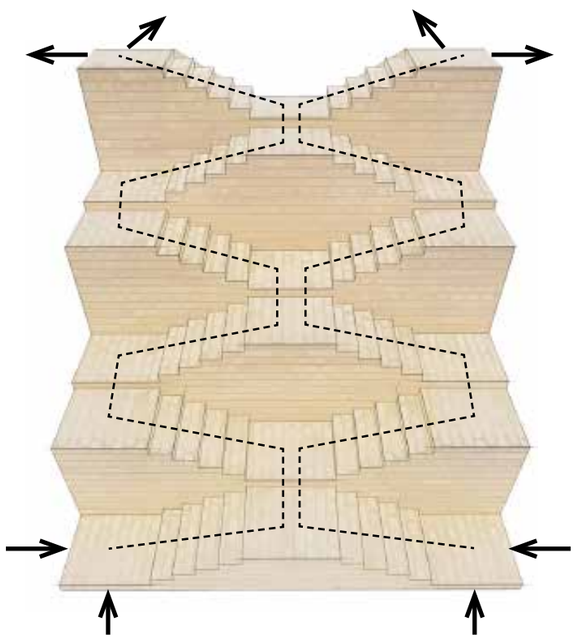


Figure 100: Staircase (default)

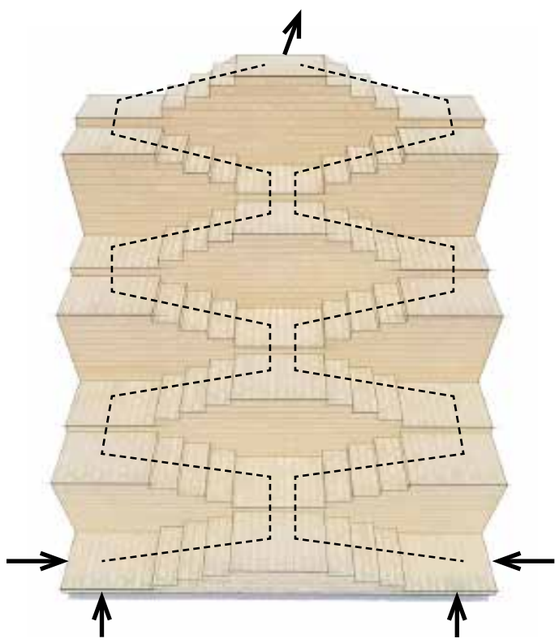


Figure 101: Staircase, changing the input in grasshopper

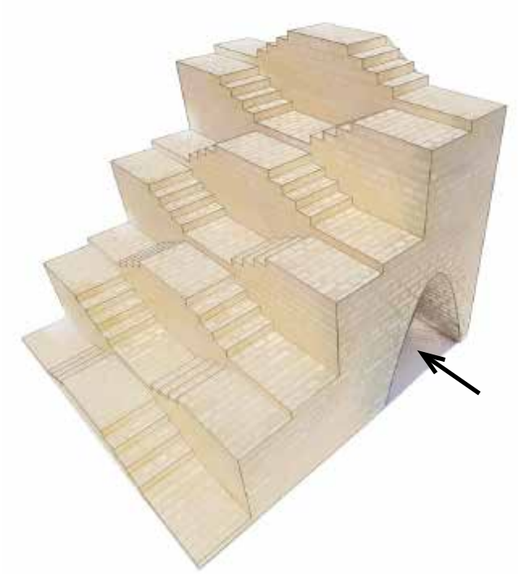


Figure 102: Staircase, arch below the stairs (connecting the ground floor)

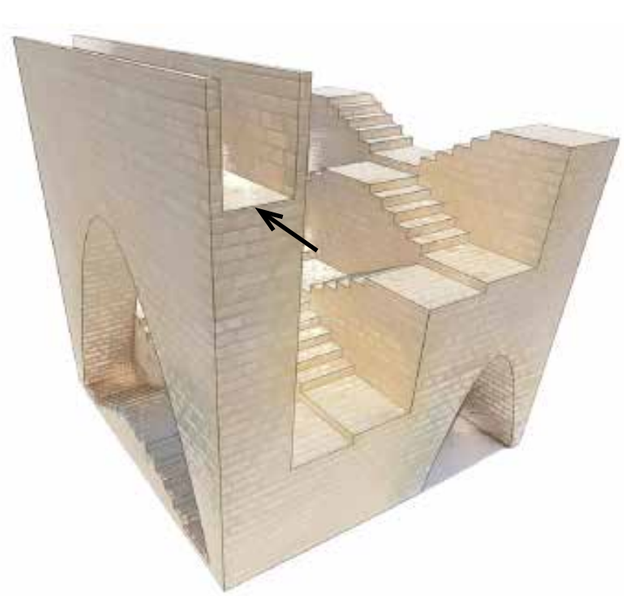


Figure 103:Staircase, arch above stairs (connecting the first floor)

Windtower

Throughout the building there are windtowers placed. These windtowers both ensure the water is ventilated well and thus will not go bad, and it also cools the building.
The design of the windtowers is based on the Ab Anbar (See image to the right). The method of naturally cooling by catching wind and then ventilating over water is used for a long time in the Ab Anbars. (Wikipedia contributors, 2021)

In the Oasis the cooling works with a pair of towers. These towers have openings toward the west to catch the wind coming from that direction. The air is pulled down to the water because it is cooler, it is then cooled itself and because the second tower pulls air out of the building, a draft is created and the building is cooled.

The windtowers do not need the full size of the voxel to function, so they are made smaller than the 5 X 5 meters. This way a functional walking space is created around the tower at the first floor. Wherever the windtower stands next to the underground waterstorage, the extra space around the windtower is used for a stair leading to the water. These underground spaces can then be maintained from here.

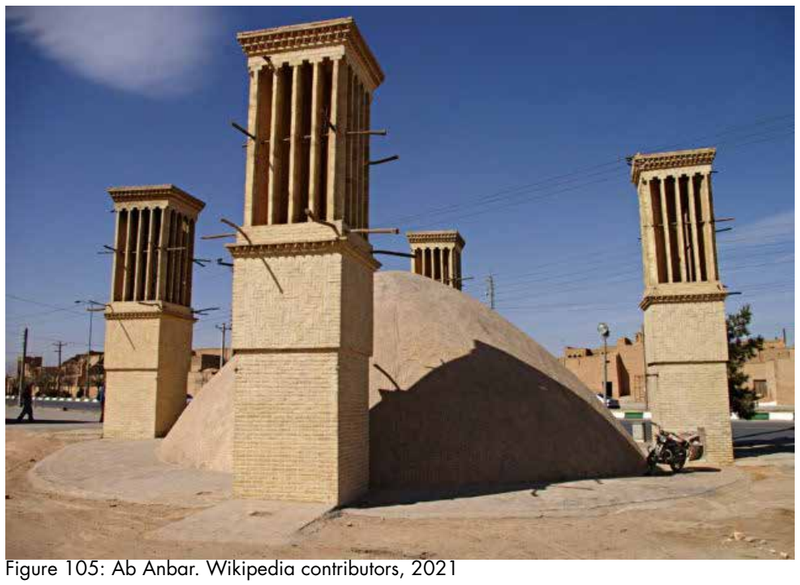


Figure 105: Ab Anbar. Wikipedia contributors, 2021

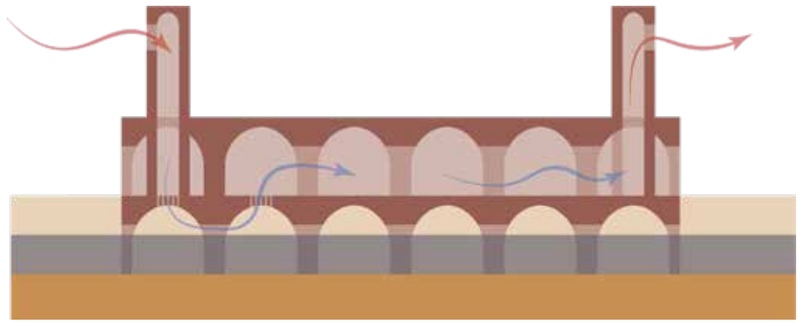


Figure 106: Windtower Diagram

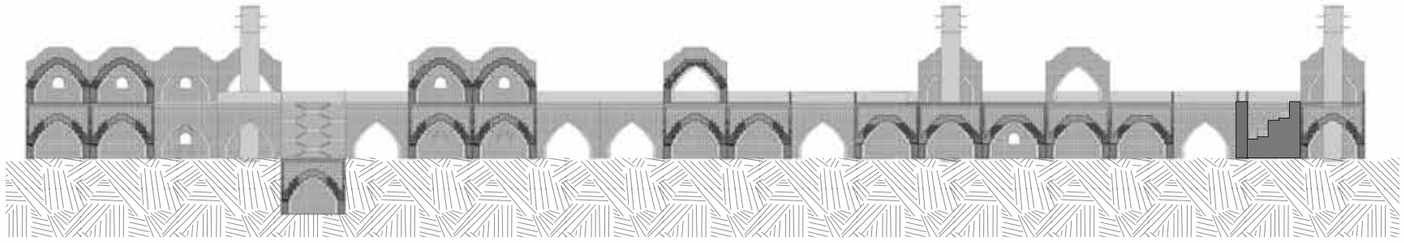


Figure 107: Section

Figure 108:Render

4_Structuring

This chapter discusses the structural part of the project. In this version of Earthy a new approach for the structural calculation was proposed. Last years the brickwork was analysed as a shell structure with a constant thickness. This year a couple of teams attempted to model the brickwork as a truss structure. We as group 4 where one of these teams.

In the first part of this chapter the material properties that were used are discussed. The second part consists of the method that was used to structurally analyse the different structures that make up the oasis. After that in the third part an overview of all the parts can be found, as well as some older versions of structural models that were made. Because these gave interesting results or highlighted a problem that proved to be important for the process. In the fourth part of this chapter all the results of the FEM analysis can be seen. The flowchart for this entire chapter can be seen. The flowchart will be discussed in smaller pieces in the relevant sub chapters.

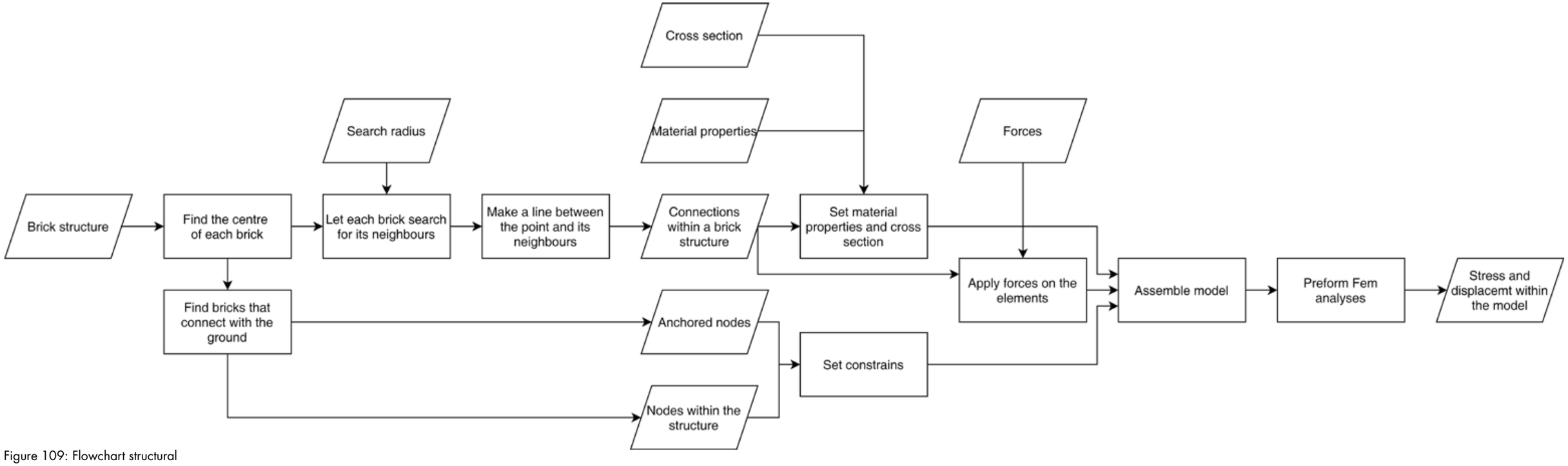


Figure 109: Flowchart structural

Material properties

For the material properties we looked at previous years of Earthy. In earthy 19 material test were performed at the faculty to test the effect of different brick compositions. After reading through all of the reports the results varied quite a lot. Therefore it was decided to keep looking. Eventually we decided on looking at the reports of last year (earthy 20) and going with values used there.

The decision was made to use the bricks made of water, clay and fine sand. These bricks are quite strong and contain materials that can easily be found in the area of the Zataari refugee cap. The values taken for this material are:

Young's modulus	150 N/mm2
Compressive strength	2,5 N/mm2
Compressive strength after safety factor	1,5 N/mm2
Self-weight	12 kN/m3

These values are used as the inputs for Karamba and are taken form the house of heritage group. Karamba is the software that was used for the Finite element modelling in this project because of our previous experience with the program. Karamba is a program that works in Grasshopper which works in Rhino. The values used to perform the Finite elements analyses are:

Yield strength	1000 kN/cm2
Young's modulus	10 kN/cm2
Self-weight	15 kN/m3

In the consultations with C. Andritos it was discussed that the yield strength could be assumed to be unlimited. This is because only the stress and the displacement are relevant. The compressive strength and the tensile strength are not used as inputs for karamba but are compared to the results of the FEM calculation to verify the structure.

Figure 110: Clay (Fontes refractories, 2021)

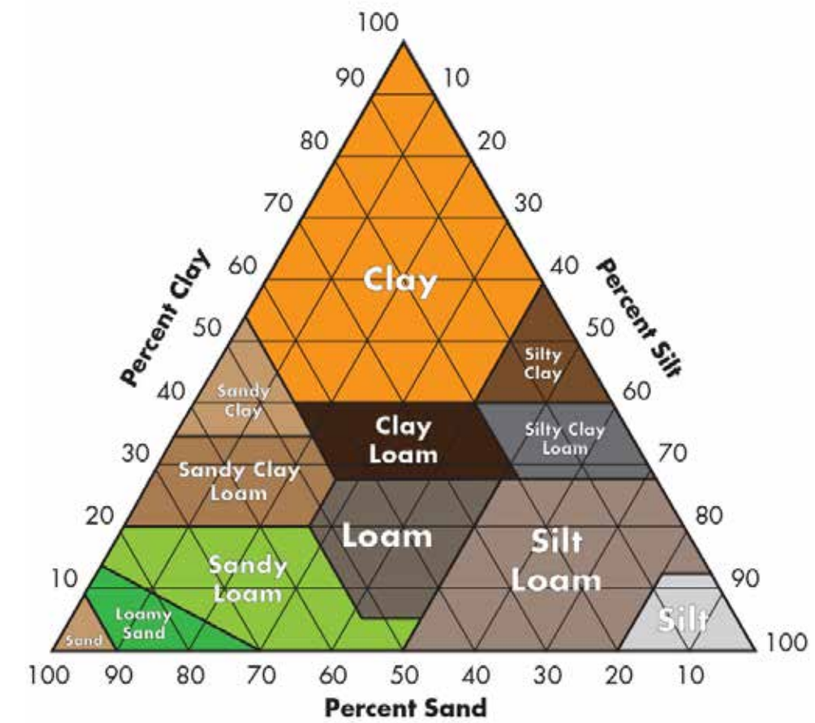


Figure 111: Soil triangle (Trugreen, 2021)

FEM-analyses

As mentioned in the introduction of this chapter the brickwork was not analysed as a shell like it was done previous years. This year the attempt was made to model the brickwork as a 3D truss system. The advantages of the truss system over the shell analyses are:

- The truss structure is a more accurate representation of the actual brickwork because now individual bricks can be observed. Before the exact location wasn't known, only the area in which it is located. The loads that act on the module are easier to assign on the correct brick. The same logic applies as in the previous point. Because each brick is modelled there is more precision.
- The loads are placed on the node's (the bricks) and not on the connections between the bricks. This means that a connection can only be in compression or tension because it is modelled as a truss structure.

Although this method seems very promising there are also some limitations.

- By modelling the bricks as a truss structure you directly assume the bricks to be hinges. This means that the bricks are stuck in place and not dry stacked. Dry stacked bricks can only transfer loads downwards. Hinges in a truss structure can go in all directions where there is a beam to transfer load to.
- Because the centre of the brick is taken as the representation of the brick not all forces are modelled in the same plane as they would in the real wall. Figure x shows the effect of modelling the lines in this way. In this way of modelling it is impossible to keep all of the forces in their original plane. Later on in this chapter a method will be discussed in which the forces would not have to shift planes.
- The connections of the bricks are modelled to have a section of half the surface of the brick. This is done because most bricks transfer their loads to two other bricks. The problem this brings is that the total mass of the structure increases with 40 to 60%. The self-weight of these structures is quite high especially if a second floor is build.

Despite the limitations of this method we took up the challenge to make this way of analysing a brick structure work.

The final modelling method was definably not the first therefor this portion of the chapter will discuss some of the previous methods and discuss the lessons learned from them.

The first model that was used was the direct dual of the mesh used to generate the brickwork with. The result of this fem analyses can be seen in the figures on this page. Therefor this truss system was nothing like the module that would be eventually build. The thing that would be taken to the next analyses is that stress concentrations were visible in the support points. This was to be expected but gave a conformations to our suspicions.

The second model that was analysed is the model with the initial brick placement. This model was not usable because of the reasons discussed in chapter 2 placement of the bricks. By this point it was obvious that the supports would need to be larger.

When the switch was made to the new way of placing the bricks on the mesh a new structural model had to be made. The way the bricks are placed is by looking at the distance of all the bricks to the mesh. By increasing the search distance we were able to increase the thickness of the brickwork to reduce the stress concentrations.

The top of the module only needed a small thickness while the bottom needed a much larger thickness. Making the entire module as thick as the bottom part needs to be would be a waste of material therefor the module was split into two. The lower part was given a larger search radius so the stress could be distributed among more bricks. The upper part got a smaller search radius to reduce the weight. Together this made for a brickwork that was sufficient even in the most extreme scenario. The results can be found in Appendix D en E

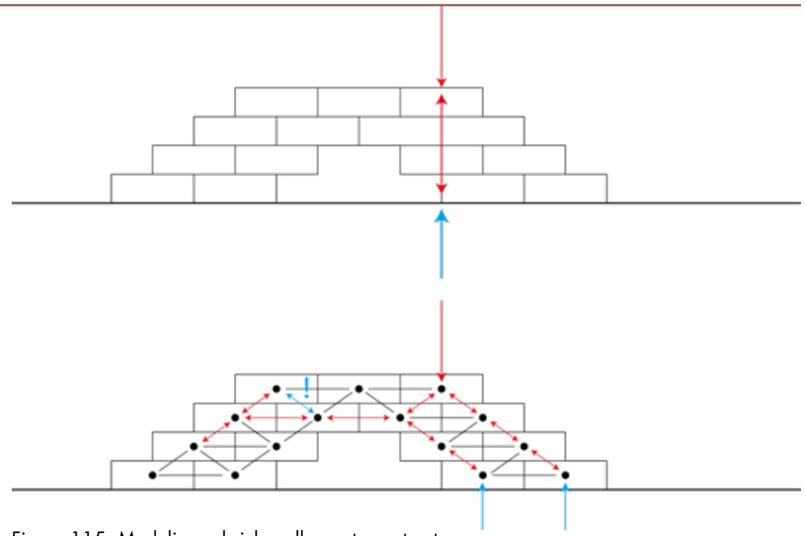


Figure 115: Modeling a brick wall as a truss structure

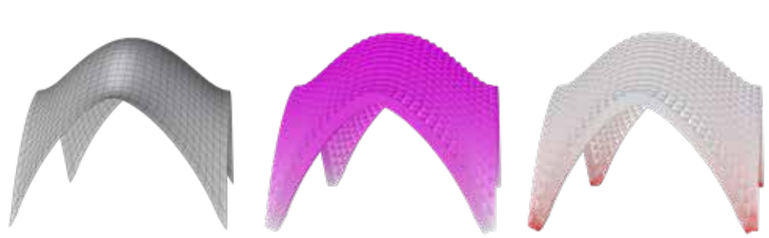


Figure 112: First FEM analyses

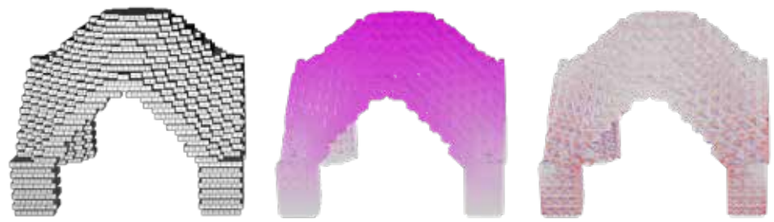


Figure 113: FEM model without arch

Figure 114: Final FEM model for the module

Fem analyses

In the later stages of the project the decision was made to add arches on the edge of each module to help with the construction. These arches have an impact on the structural performance of the module. Modelling these elements into the current structure gave some difficulties.

In practice the bricks that rest on a part of the arch would all transfer their loads on a different place on the arch. If the part of the arch would be modelled the same way the rest of the structure. The forces that the lower bricks transfer to the arch would go upward instead of down. This is not realistic and therefore we came up with a different way of modelling the arch in the structure. By moving the centre of gravity to the lowest point of the piece of the arch all the forces would flow in the correct direction. The problem that comes with this way of modelling is that the displacement is not correct anymore. Because the forces and the beams are in a completely different place they support the roof way more. The resultant displacement is way lower than it would actually be.

Afterwards the idea came up to model the arch as a lot of smaller parts so the beams could be at their actual place and the displacement would be more correct. The nodes in the arches would be modelled so they could not rotate compared to the other nodes within the same element. Due to limitations in time this was not further developed.

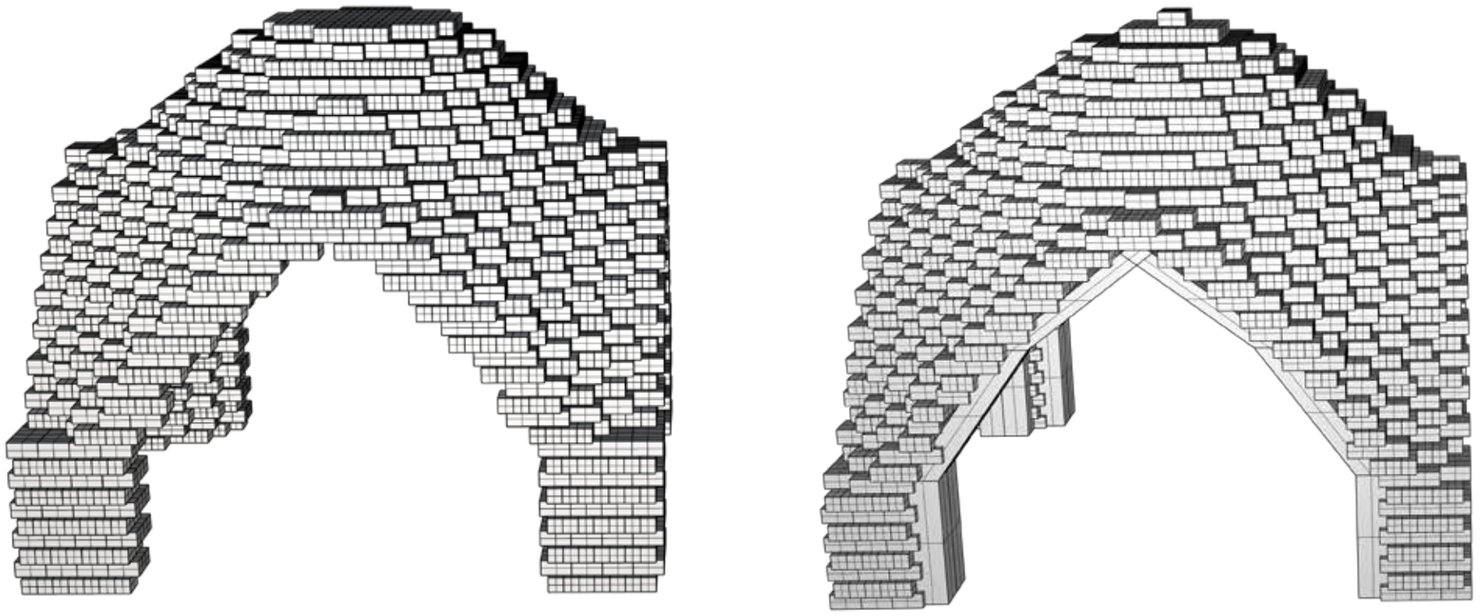


Figure 116: Ways of modeling forces on a beam in a truss structure,

During one of the consultations with C. Andritos the idea came up to model the bricks as planes with unlimited stiffness. In this way the forces could go vertically to the next layer. The problem with forces rotating to another plane would be resolved. Due to time limitations this method was also not developed further.

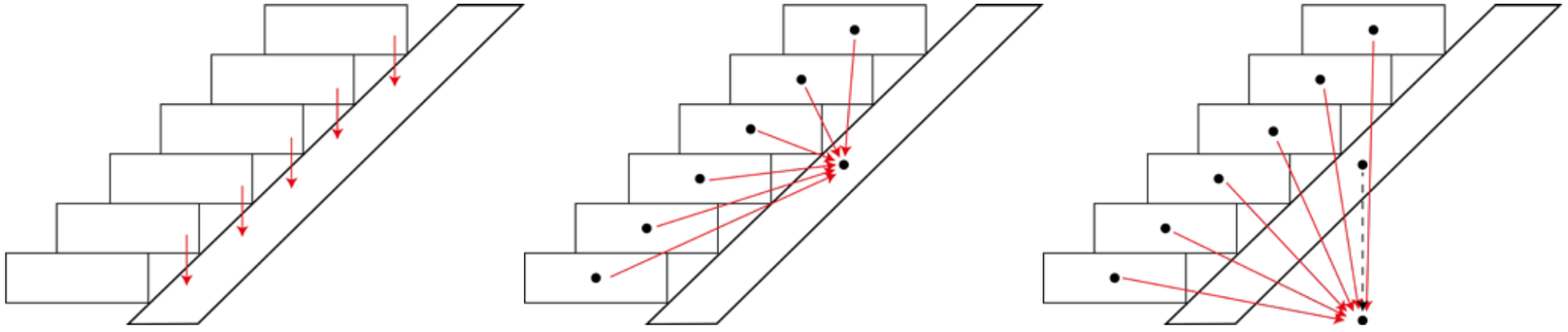


Figure 117: Addition of an arch in the module

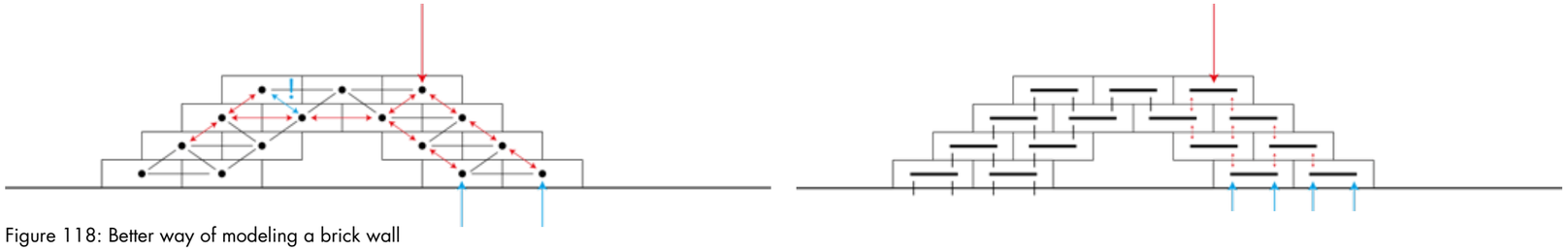


Figure 118: Better way of modeling a brick wall

Inventory of analysed parts

The figure on this page shows the parts that were eventually analysed. Although the entire building can be built with only one module it was decided that multiple scenarios for the module needed to be calculated. The first is the module with the arches. This is the basic module that will be used in the building. The second module is a module with structural walls. The reason for this calculation is to see what effect the walls could have on the stress in the structure. The third module that was calculated is a combination of four modules. All the modules are calculated with loads as if there was a full second floor on top of it. In the oasis this is the most extreme scenario.

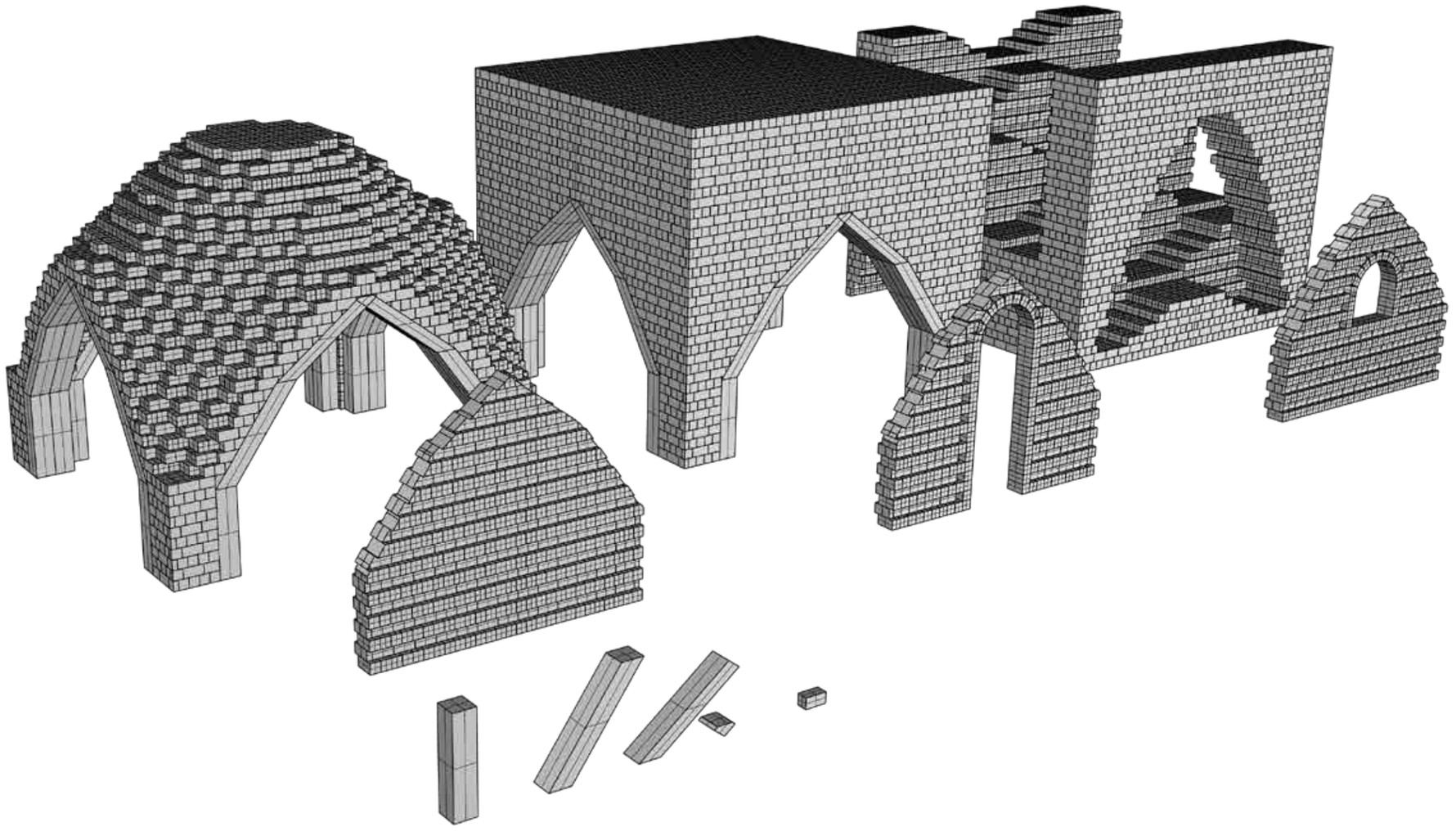


Figure 119: Inventory of analysed parts

Module

Out of all the analysed parts the module went through the most amount of iterations. All of the old versions can be found in the annex as they are already discussed in the part about the Fem analyses. In this chapter the results of three modules will be discussed: the module with arch, the module with structural walls and four modules combined.

Module with arches
As discussed in the chapter FEM analyses the arches are modelled different to the rest of the structure. This makes it so that the brick structure looks different from the finite element analyses. This is the basic module and serves as the basic building block for the oasis building. The compressive strength is above the allowable limit. The reason we acepted this is because the module without the arches is also calculated and is under the limit. The full results can be found in the annex.

Stress, tension: 0.152 kN/cm2
Stress, compression: 0.180 kN/cm2
Displacement: 18.1 mm

Figure 120: Module with arches

Module with structural walls
To see the effect walls could have on the stress an displacement in the module walls were modelled. In the Oasis building walls do not carry any load but their own weight. Therefor it seemed interesting to see the effect it would have if the walls needed to carry weight. This module is the same as the first module only the arches are not there since they are not needed to help with the construction of the module.

Stress, tension: 0.030 kN/cm2
Stress, compression: 0.044 kN/cm2
Displacement: 6.5 mm

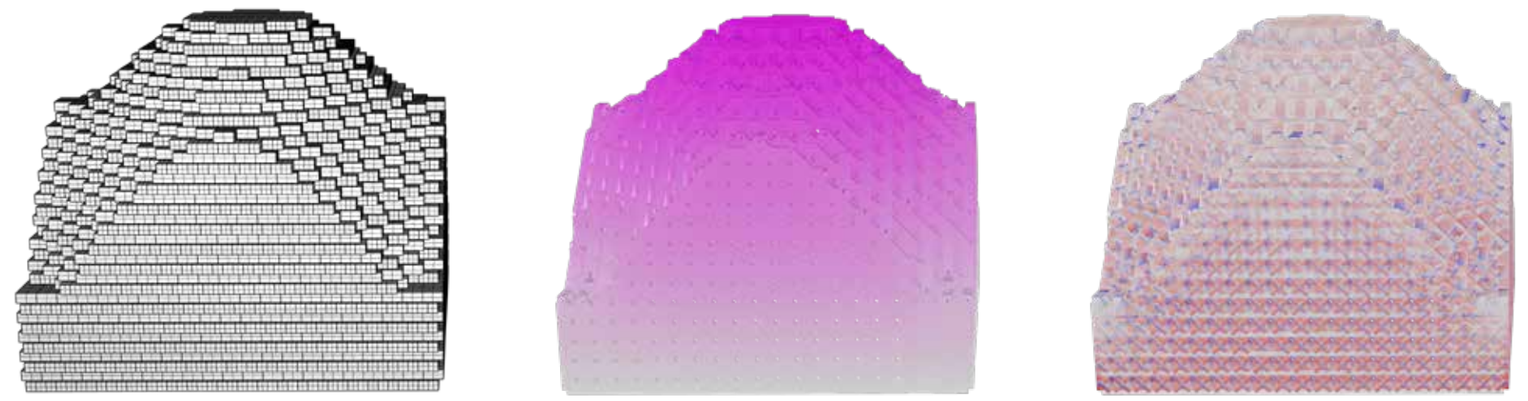


Figure 121: Module with walls

Four modules
The four modules were modelled to see the effect modules could have on each other. The centre column of the four modules it put under pressure of four modules instead of one it is there for four times bigger.

Stress, tension: 0.124 kN/cm2
Stress, compression: 0.146 kN/cm2
Displacement: 16 mm

Figure 122: Four modules

Walls

For the walls three options were considered for the shape of the window. Due to the way the connections are modelled, the results were not as expected. Out of the options the square window came out on top. The reason for this is that the square grid fits perfectly in the brick pattern. The round and semi round windows performed worse because the connections between the bricks are not as regular as with the square window. In practice this could never be the case because an arch causes a lot less stress in the corners of the window. Besides that the bricks at the top of the square window can only be held in place by tension. Due to time constraints we were not able to resolve this issue. For this type of calculation a shell analyses might perform better than a truss system analyses. Because the stresses can be more locally observed and the wall acts more as a continuous structure than a truss structure.

Without openings
The wall without openings is to separate functions within the building. Besides its practical function within the building it served as a comparison for the other walls. The difference between an opening and no opening.

Stress, tension: 0.044 kN/cm2
Stress, compression: 0.084 kN/cm2
Displacement: 7.41 mm

Square window
As said before the results from the square window were not expected. Out of all the windows this shape performed the best. The result that stood out the most was the compression that was lower than the compression in the wall without openings.

Stress, tension: 0.050 kN/cm2
Stress, compression: 0.077 kN/cm2
Displacement: 9.01 mm

Round window
The wall with the round window was expected to outperform all of the other options but eventually came second to the square window. Although the compression is lower the displacement is higher. While realistically it should be lower.

Stress, tension: 0.051 kN/cm2
Stress, compression: 0.075 kN/cm2
Displacement: 9.23 mm

Semi round window
Although the semi round window performed the worst out of all the windows when looking at stresses this is the one used in the project.The reasoning behind this is that this type of window is more practical than a completely round window because of the flat bottom. This means a windowsill can be placed and can be used by the owner of the building. The reason it was picked over the square window is because the method of modelling is not correct and therefor it is expected that this window will outperform the square window in practice.

Stress, tension: 0.052 kN/cm2
Stress, compression: 0.082 kN/cm2
Displacement: 8.88 mm

Door
Besides solid walls and windows doors can be quite useful in a building. The door was designed after the windows therefore only one was designed. To make it easier for the people building the structures the arch is the same as in the window.

Stress, tension: 0.043 kN/cm2
Stress, compression: 0.083 kN/cm2
Displacement: 8 mm

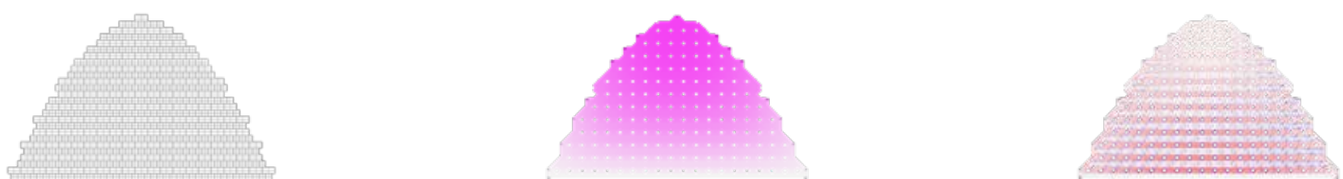


Figure 123: Wall without openings


Figure 124: Wall with square window


Figure 125: Wall with round window


Figure 126: Wall with semi round window


Figure 127: Wall with door

Stairs

The last parts that were structurally verified are the two pieces of the stairwell. The two element pose very different challenges. The walkway under the stairs is a quite small arch but there is a lot of dead load on top of it. The bridge over the stepwell is a very high arc without a lot of load, but the height makes it vulnerable to forces perpendicular to the arc.

Walkway under the stairs
The walkway under the stairs is there to connect two shops or a workshop with a corresponding shop. As can be seen in the top right figure the displacement is quite minimal and the stress is within the allowable limits.

Stress, tension:	0.088 kN/cm2
Stress, compression:	0.121 kN/cm2
Displacement:	5.91 mm

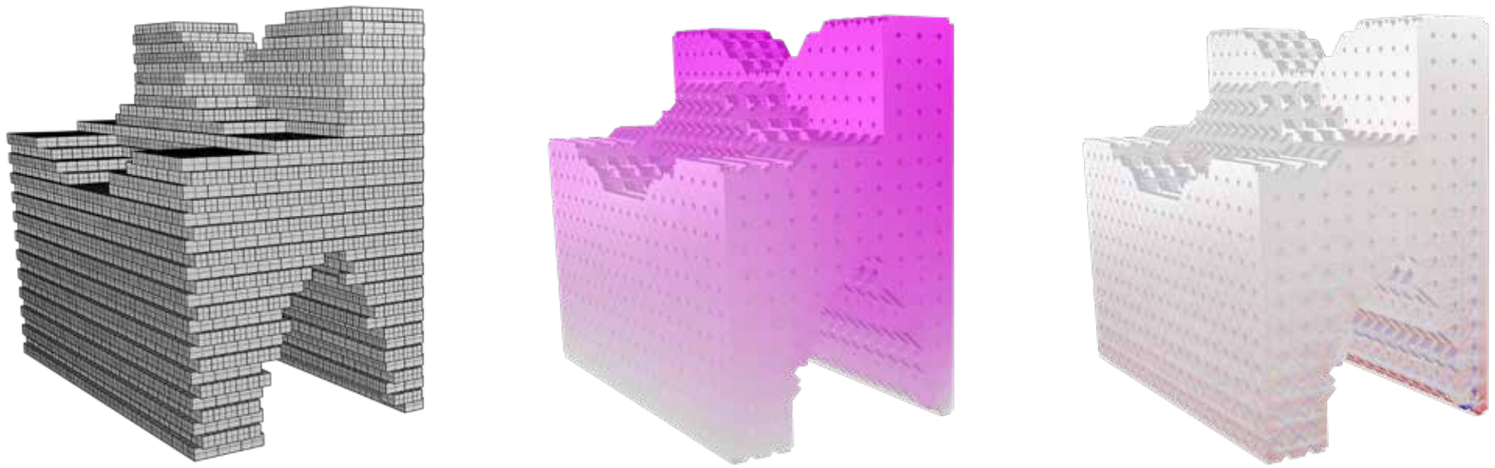


Figure 128: Walkway under the stairs

Bridge over the stairwell
The bridge over the stairwell shows considerably more displacement than the walkway. Although it is a lot higher it is still within the allowable limits because the span is 5 meters. The stress is in the same order of magnitude as the walkway. This can be explained by looking at the amount of bricks in the bottom few layers. Although the walkway has a higher load it also has more bricks to divide the stress over.

Stress, tension:	0.055 kN/cm2
Stress, compression:	0.114 kN/cm2
Displacement:	10.0 mm

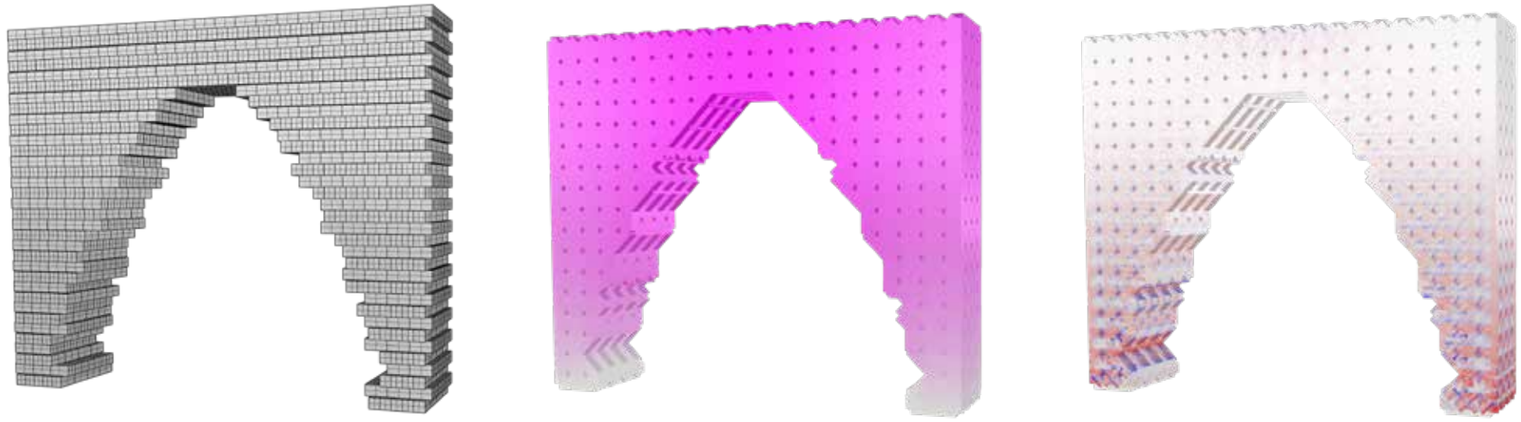


Figure 129: Bridge over the stairwell

5_Construction

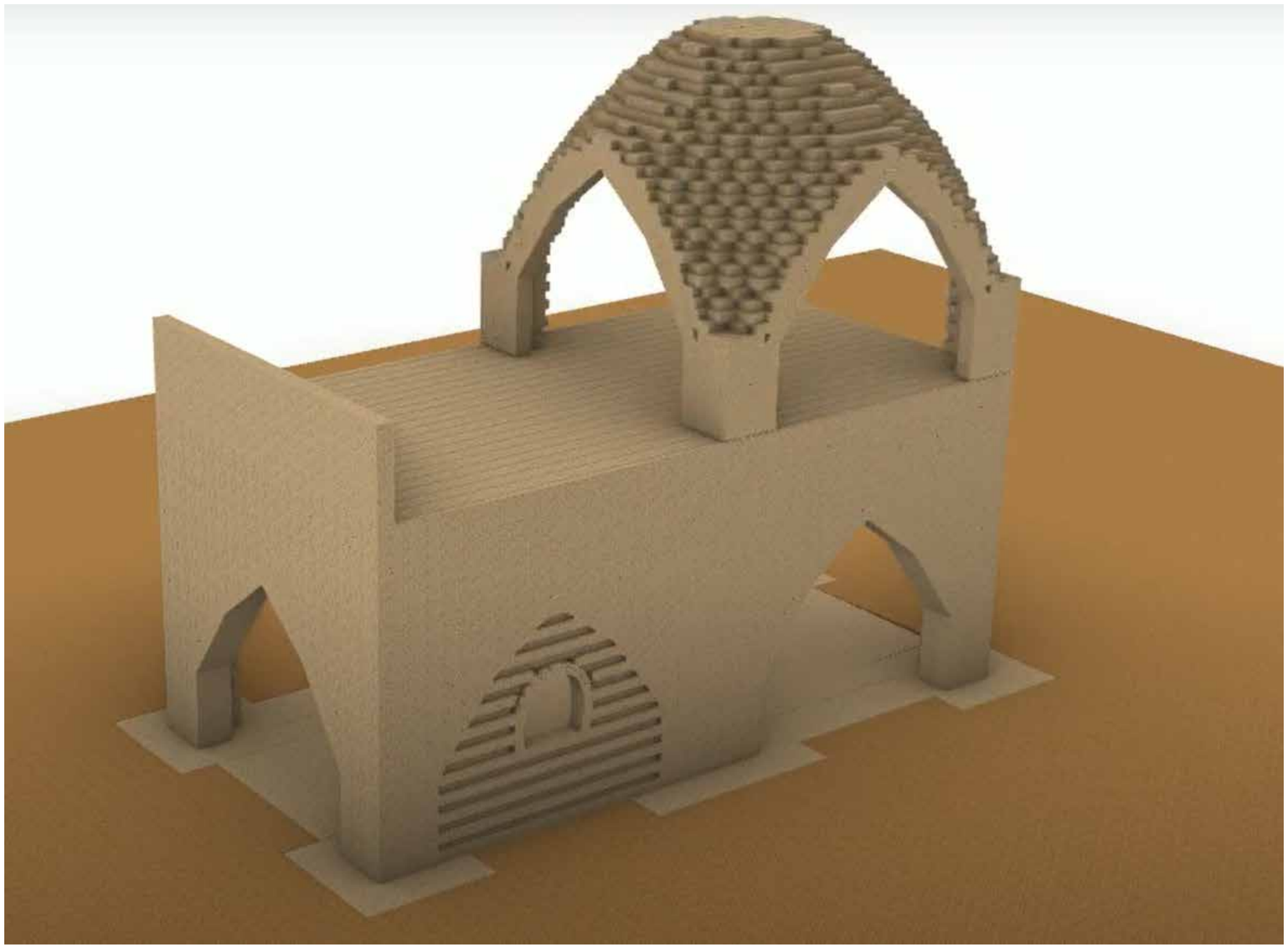


Figure 130: Full view of construction module

Material Analysis

What production methods are available for building with raw earth?
The following five products are available:

- COB (filled with plants)
- Rammed earth
- Poured
- Infill Material
- Compressed Earth Blocks (CEB)

According to the graduation report of van Vilsteren (2019) the soil around the camp mostly contains clay. This causes bad irrigation of the ground. However, clay is a good material to build houses with. So this means the soil of Zaatari is suitable for construction.

For this project, two methods are chosen. Based on the limited skill of the craftsmen that will be on the building site. The methods will be CEB (Compressed Earth Bricks) and rammed earth.

Rammed earth will be used for making the foundation. The CEB-bricks will be used for making the modules.

Figure 131: COB Bricks (Cornel, 2017)

Figure 132: Rammed Earth (Silva et all., 2014)

Figure 133:COB (Earth Architecture, 2021)

Figure 134: Pourced (Home, 2020)

Figure 135: Infill Material (Hueto et all. 2021)

Collection of parts

According to the view shown in figure to the right, in the collection of parts is shown which different modules and parts are used to build with.

A total summary of the collection:
- 2 modules
- 1 stairwell
- 3 wall types
- 4 building elements

Let's explain the parts a little bit more.

- Modules

There are two different modules. Both are made of the same shell structure that is made out of bricks. But the difference is the finishing of the roof. In element 1 can be seen that the roof is still an arch. In element 2, the roof is flat and the walls are straight. The reason for this is that module 1 can't acomodate a second floor and module 2 can. Or that the roof of module 2 function as an roof terrace.

- Stairwell

The stairwell (element 3) is based on the Indian stepwell. This stepwell functions as a stair to get to the roof of module 2.

- Wall types

There are three different walls. Element 4 is a closed wall. Element 5 has a door opening. Element 6 contains a window.

- Building elements.

In element 1 and 2 can be seen that there is an arch that supports the bricks, this arch consist out of 4 building elements. Shown near element number 7.

Figure 136: Market square

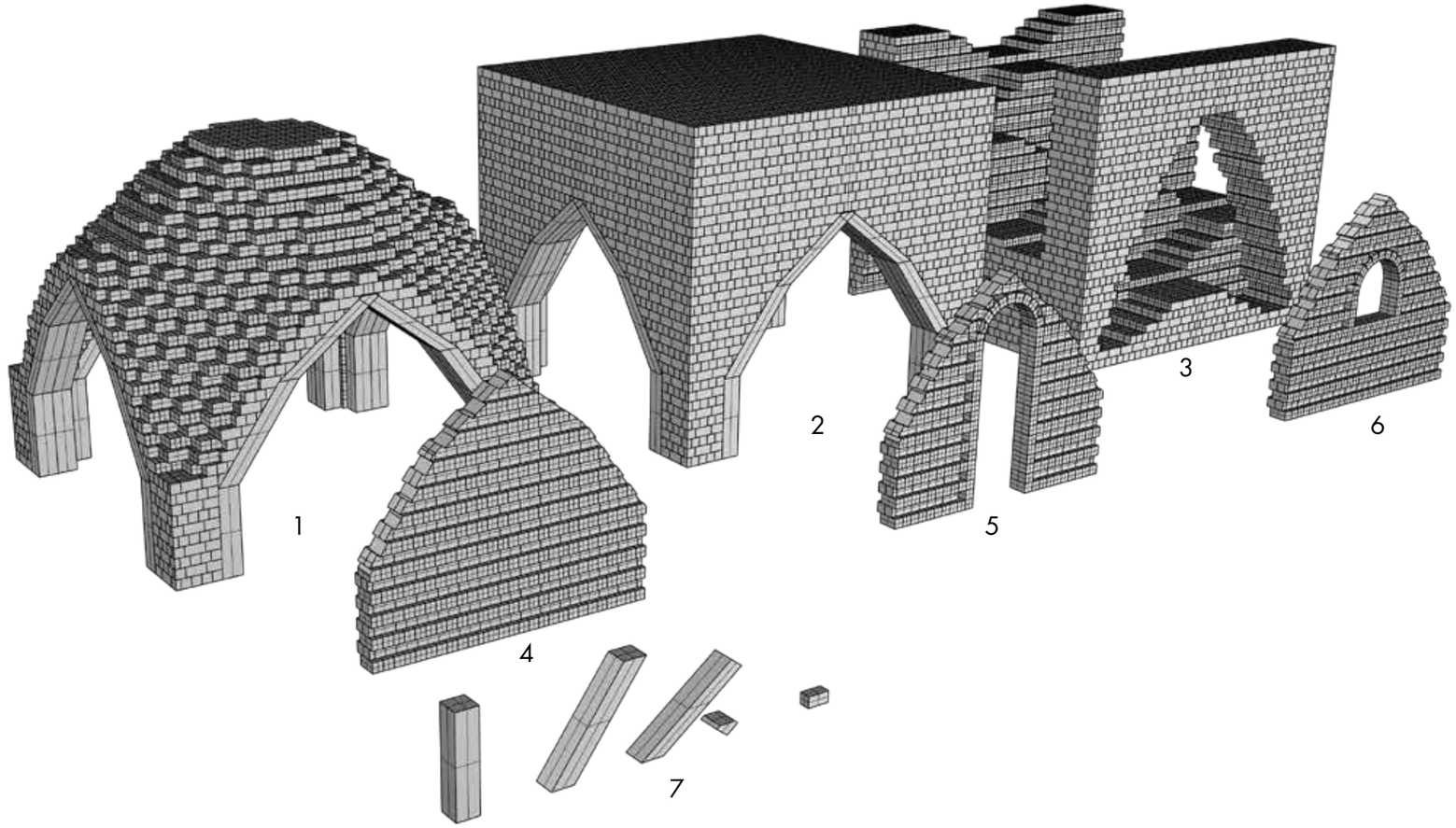


Figure 137: Collection of parts.

Foundation

As mentioned before the foundation is made out of rammed earth. The foundation first needs to be dug out, and then filled with stones and clay. This slab is then smashed and rammed to compress the material and give it more strength.

The foundation supports the walls and columns of the modules.

- Underneath every single corner of a module a foundation block of 1800x1800x850mm is placed. When columns come together, this needs to be increased to a width of 2400mm (as can be seen in the figure to the right).
- Underneath every wall a smaller foundation needs to be placed, it is necessary to prevent settlement of the walls. Alongside the walls a slab of rammed earth of 750x330mm in section needs to be placed.

As can be seen in the figur to the right, underneath the foundation a parabolic shape is placed. This parabol, with its minimum in the middle of the module, is used as a water storage system.

The reason for this parabolic shape is erosion. When a normal module is placed and one of the bricks fails (which is likely when you work with water and earth). The whole shell structure fails. If a parabolic shape is placed and one element or bricks is slightly eroded, the entire shape still stands. Therefore, a parabolic shape for the underground modules is chosen.

When we looked at the ab anbar we found that eggwhite and goathair is used to make the wall waterproof. So the walls of the parabolic shape where the water is stored, should be 'painted with goathair and eggwhite to create a waterproof layer.

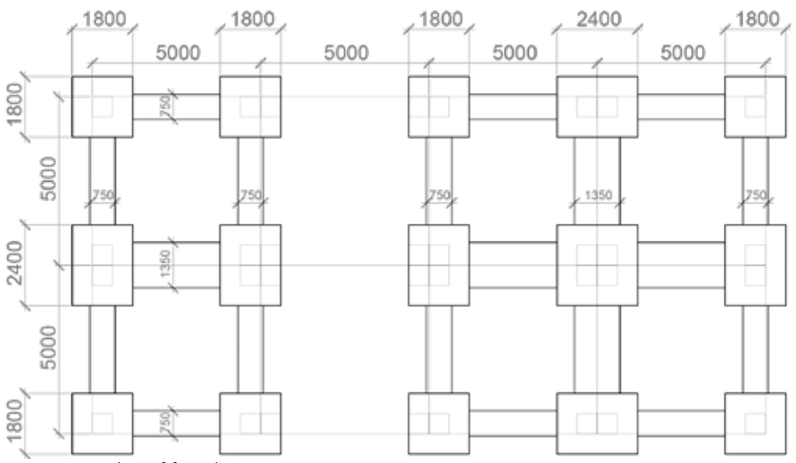


Figure 138: Plan of foundation

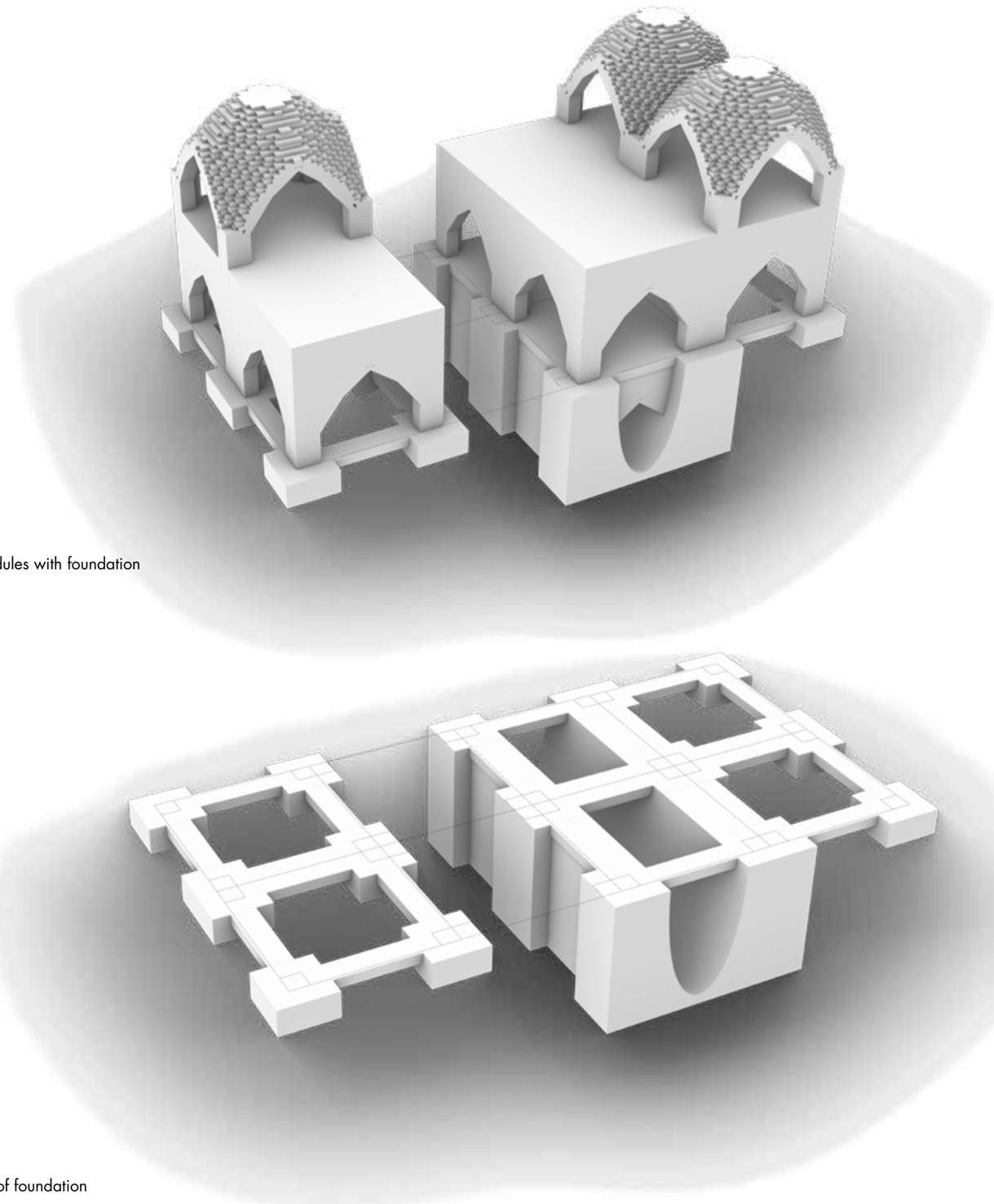


Figure 140: Modules with foundation

Figure 139: 3D of foundation

Manual

As is mentioned before the construction methods are rammed earth and compressed earth bricks. In figure: Rammed earth an example of rammed earth is shown and in figure: Compressed earth bricks making, the production of compressed earth bricks. The size of the bricks is shown in figure: bricksize.

For the craftsmen on site, a visual manual is made (figure: construction manual), to clarify the steps:

1. First the foundation needs to be dug out. The earth is stored for producing the bricks.
2. Place a mould within the space that is dug out and fill with stones and clay. Followed by ramming the earth with a wooden pile.
3. From the clay and sand, the bricks are produced. The bricks are pressed in the machine. The mixture, according to previous reports and the book Raw Earth form LEVS architects, is as follows: Clay, Sand and Gravel in a proportion of 3:3:4. In addition to this volume, 10% of water is added and 10% of straw to increase its resistance to tension.
4. The bricks are placed in a partial overlap to make the bricks work as one.
5. The corners of the module are made with bricks in partial overlap.
6. In between the corners a wooden mould is placed. This mould is created to form the arched shape.
7. In between these wooden moulds, a diagonal wooden slab is placed. From the middle point of this wooden slab, a wire is spanned towards the corner of the module. This functions as a guide to place the bricks.
8. When the walls are made, on top layer by layer, bricks are placed in a circle to finish the dome.
9. If relevant the walls are continued in a vertical way, to build a horizontal roof. The space in between the shell and roof is filled with bricks as well.
10. By this horizontal roof, another module can be placed on top.

Figure 141: Rammed earth (Project Sudan, 2020)

Figure 143: Compresse earth bricks making (DiStasio, 2015).

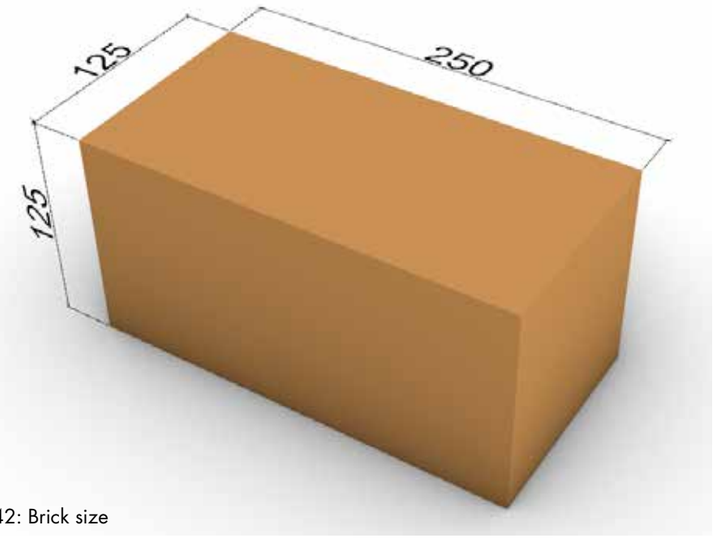


Figure 142: Brick size

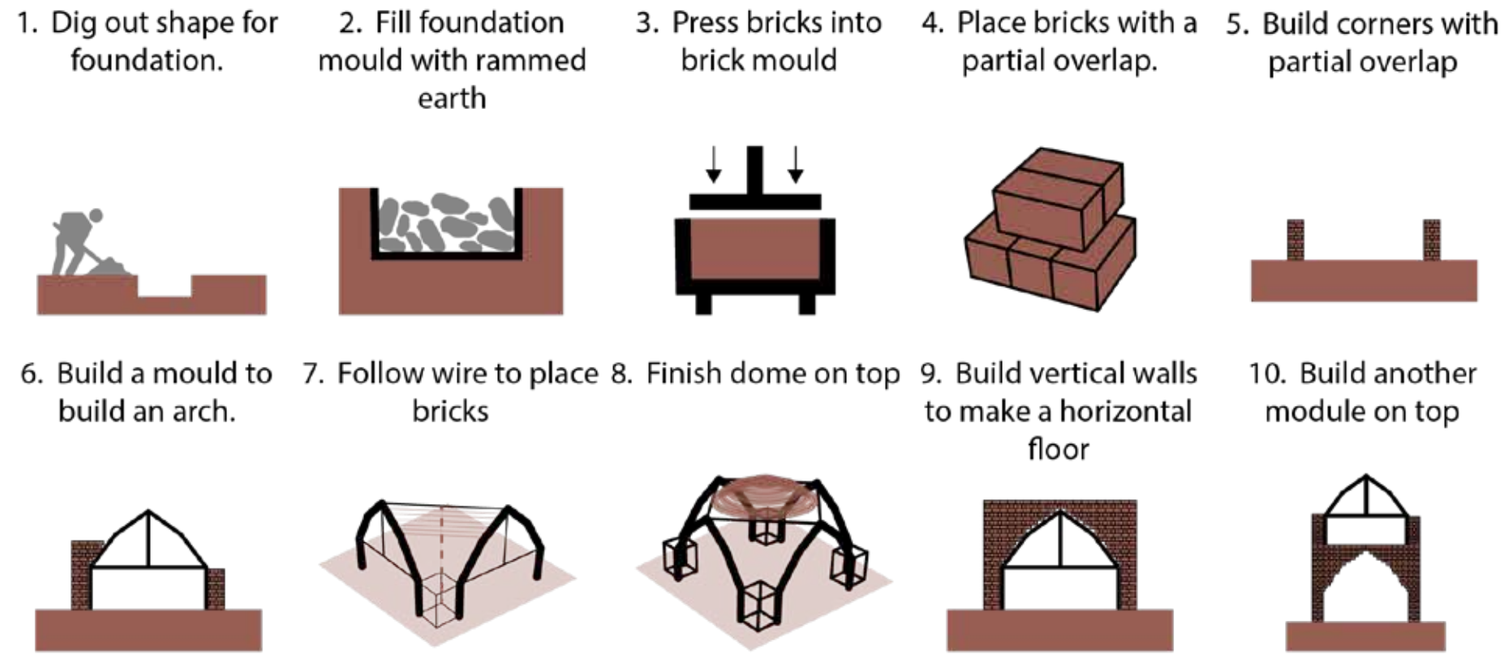


Figure 144: Construction Manual

Figure 145: QR of the construction video

Organization method

For the organisation of the construction, the floor plan is used in an efficient way. For the construction a local approach is used. Within the floor plan six different districts are defined, and three squares.
Each square is responsible for two districts. So district 1 and 2 together share the square for the construction. District 3 and 4 as well and finally district 5 and 6.

Within these square the earth and clay stored for the production of the bricks. Every square has its own container with building equipment. This container contains the following building equipment:

- Earth brick machine
- Wooden Mould
- Measurement equipment (including wires)
- Required drawings
- Ladder
- Digging material
- Wheelbarrow
- Basic tools for repairing the equipment

With this equipment and the manual provided earlier in the report the future inhabitants and shop owners are able to build the building themselves. Because the plot is split up in different districts the plan can be executed phased. An important note is that the roads remain open for transportations of people and their equipment.

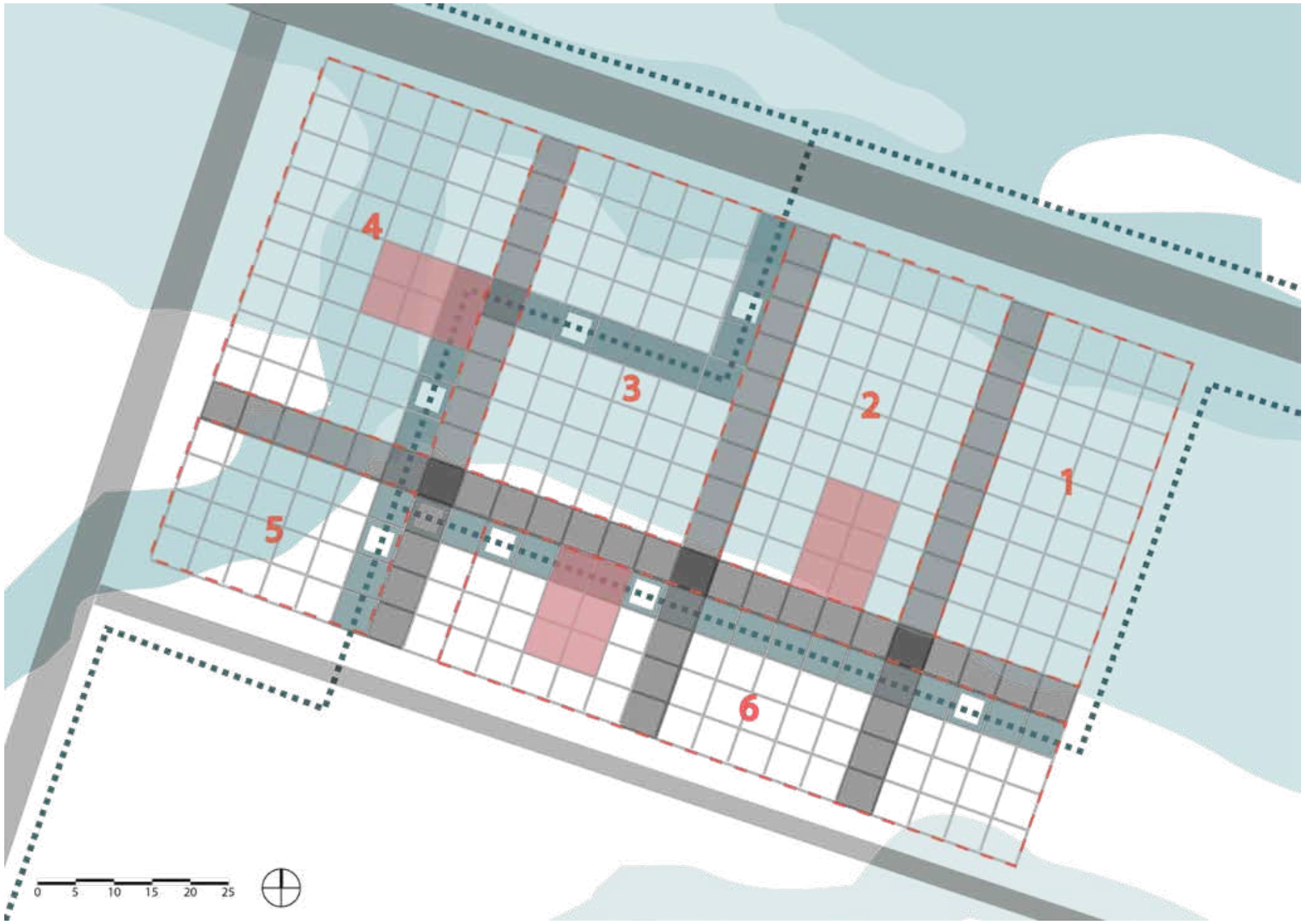


Figure 146: Organisation method

6_Reflection

Group reflection
For the midterm presentation we focussed on creating a floor plan by dynamically relaxing a bubble diagram in Python. After the midterm presentation we discovered that this wasn't the right direction. The feedback on our mid term presentation and the presentations of other groups made us realise that our focus had to be adjusted. Trying to write code for the entire design was not the solution because we had to act more like designers instead of programmers. The cause of this problem was the way we work: if we have an idea we like to work it out, not rethink it until it's perfect.

From this point on we took a step back, made our program of requirements more specific and started designing more in line with the vision to store water. We set goals to work in a different way and we managed to hold on to those goals. The goals we set for ourselves were to brainstorm a lot more and also focus on how to realise the connection between storing water and creating a legal economy.

In the end we were still struggling to find the right balance between being the architect and the programmer, but it did work out. It also took a lot of time to understand the software and programming language we were working with. Because of this we were not able to finish all the work we set out to do. Translating thoughts into code a computer can understand can be quite difficult. If we had more time, we would've liked to make a script in Python with the rules we created in the manual game. Another improvement is to give scores to the rules and configurations originating from them. This would make the player of the game more eager to come up with better rules.

Looking at the structural component of the project, there are some challenges and points of improvement. The first challenge we came across was making a span that can only take on compressional forces. This is a completely new approach. We were used to designing constructions where either materials are used that can take tensional forces, or we used a combination of materials where one can take compression and the other can take tension.

The second challenge was the brick size. Currently, the brick size is very useful for the roof module but not as useful for the stairs. The height of the brick is not ideal for the height of one step in the stair module nor is it half of a step. A new height for the brick needs to be developed and tested for the normal module.

As mentioned before in the structural chapter, the method used for the structural analyses is something that could use further development. Our method didn't work very well for all of the scenarios and had some clear limitations. Further development in the method discussed with C. Andritos could be beneficial for the reliability of the FEM calculations without the need for a discrete element analysis.

Sjoerd
I applied for Earthy because of my interests in computational design. In the course Research and Innovations, I wrote a paper about topology optimization and its possible appliance in wood. This was a difficult topic, as was the course Earthy. I was struggling with constantly being focussed on making computational rules in words and Python. It reminds me of the three different personalities: doer, thinker and speaker. I'm more of a doer, I want to get things done. Because of this I'm easily distracted, and programming is hard for me. Although I really like the project. Nevertheless, I learned the basic skills of programming in Python and making computational rules with words. This was a different approach than I usually have learned at the University. It helps you to better understand how to handle design driven by data, which I'm grateful for that I applied for this course. Although I still have a lot to learn in computing and programming.

Marnix
Generative and parametric design and automation in the built environment are subjects that I'm very excited about. This is the main reason why I chose EARTHY. The course challenges students to work with logic in order to get to configurations, shapes and structures. It means that the designer is in charge of the motivation behind design choices and the computer does all of the heavy lifting and bulky work. It leaves a lot of valuable time to think about what is important. In reality though, making the computer do what you'd like it to do can take up a lot more time than expected. Especially when you just start programming without really knowing how everything is connected, like I did. Luckily, the learning curve always starts slow before it takes off. At this point I am at the foot of the hill and intend to continue climbing until I have the view I desire.

Doris
During my Bachelor I was always surprised at the ease of which design choices were made without being able to reason why this particular direction was chosen. And that you are half expected to come up with an argument afterwards. This felt wrong to me, so when I chose the masters Building Technology I felt more at ease, because it felt right being able to base choices on valid arguments. The course EARTHY takes this three steps further. Making a design that is repeatable in different circumstances.

During the course I struggled with when to design and when to let the code take you into the right direction. I especially enjoyed writing code for the configuration. To base the layout of the building on a set of rules that were made way more explicit than expected, seemed to make a lot of sense. Also, when writing code I felt the urge to continue writing until the entire problem was solved. So I will probably continue learning to write in Python even after this course.

In the end I learned a lot from this course. I learned that for me, parts of the design are just not meant to be done computationally, but other parts I will definitely use in later designs. I also once more learned that having a physical model or game to play with adds so much value to the quality of the design.

Tim
I applied for Earthy because I always had an interest in computational design, programming and modelling. I thought it would be interesting to work with python once and see possibilities of what you can do with it and how you can implement this in a design. All the given lectures were very interesting. It was interesting to learn what you can do with earth, but also what you can't do with it. With the python & mathematical lectures, I struggled a little bit in the beginning to see what we can use it for. This became more and more clear later on once you got started with your design. It was also very interesting to analyze old buildings from the Middle East and other countries around the world, as the methods are still relevant. It was an intense course, but I learned a lot from it. I do not regret choosing this course.

Jordy
During my bachelors and masters i found that parametric and generative design are things that interests me. I choose earthy to further develop skills in this area.

The design part of the project was quite hard from time to time. Going through the process of generativity designing a building while learning about python was quite a lot. Especially because the course is quite free in the way you approach the process. Having an idea, looking for the correct software, learning about it, implementing it requires a lot of time and effort. I can say that I learned a lot. Not only multiple software applications but also how to set up a generative design process and how to keep it parametric during the whole process.

The lectures at the start of the project were very useful. The only thing with them was that it was not always clear what they were useful for at that time. Because there is no clear path through the course you had to connect the dots yourself. All things considered I enjoyed earthy and had quite a good time with the most international group of earthy 21.

7_Bibliography

- Crop water needs. (n.d.). Irrigation Water Needs. Retrieved 31 October 2021, from https://www.fao.org/3/s2022e/s2022e02.htmHow
- Sustainable Dyeing is Changing the Textile Industry. (n.d.). Plug and Play Tech Center. Retrieved 31 October 2021, from https://www.plugandplaytechcenter.com/resources/how-sustainable-dyeing-changing-textile-industry/
- Karapetyan, A. (2019, August 14). The Carpets of Syria - A Mirror of the Country's History. Arab America. Retrieved 31 October 2021, from https://www.arabamerica.com/the-carpets-of-syria-a-mirror-of-the-countrys-history/
- Maiwa. (2018, June 25). the MAIWA BLOG: Natural Colour A Bengal Story - Part Three. The MAIWA BLOG. Retrieved 1 November 2021, from https://maiwahandprints.blogspot.com/2018/06/natural-colour-bengal-story-part-three.html
- One4Leather. (2021, October 25). Why water is so important for leather tanneries. One4Leather. Retrieved 1 November 2021, from https://www.one4leather.com/article/water-usage-in-leather-tanneries
- Project, I. H. M. O. S. (2021, September 22). The Ink that Lasts Forever: Textile Printing in Syria. Syrian Heritage Archive. Retrieved 31 October 2021, from https://syrian-heritage.org/the-ink-that-lasts-forever-textile-printing-in-syria/
- UNHCR. (2017, November). A ssets of Refugees in Zataari camp: A Profile of S kills. https://reliefweb.int/report/jordan/assets-refugees-zataari-camp-profile-skills
- Ghanavati, M. (sd). Ab anbar. Opgehaald van iranindepth: https://iranindepth.com/ab-anbar-a-masterpiece-of-ancient-water-reservoir-architecture/
- Kruijt, R. (2014, December). Rightful landscape. Retrieved from wur: https://www.wur.nl/upload_mm/a/2/0/bb0167df-9349-4b13-8e45-a80ba35cb63d_RKruijt-report-s.pdf
- McFadden, C. (2020, November 2). The Engineering Behind Chand Baori and Other Famous Stepwells. Retrieved from interestingengineering: https://interestingengineering.com/the-engineering-behind-chand-baori-and-other-famous-stepwells
- Society, N. G. (2019, August 2). The Caravanserai. Retrieved from nationalgeographic: https://www.nationalgeographic.org/media/where-worlds-and-ideas-connect-caravanserai/
- Vilsteren, M. v. (2019, December 19). Al'Zaatari Reconnecting the landscape. Retrieved from Issuu: https://issuu.com/studiolandmark/docs/180716_mvvilsteren_alzaatari_reconnecting_the_land
- Wereldwonderen. (sd). Hangende tuinen van babylon. Retrieved from Wereldwonderen: https://www.wereldwonderen.com/oude-wereldwonderen/hangende
- Photo's bazaar of Aleppo. Retrieved from google.nl/maps
- Raw materials, raw clays, crushed or clay mixtures - Fontes refractories. (2021). Fontes Refractories. Retrieved froom: van https://www.fontes-refractories.com/en/products/raw-materials
- Lawn Care Tips: Get to Know Your Soil Texture. (2021). Trugreen. Retrieved from: https://www.trugreen.com/lawn-care-101/learning-center/grass-basics/dig-deeper/soil-texture
- Cornel, L. (2017). Cornell Cob in Cornwall BLOG. Leslie Cornell Cob and Lime in Cornwall. Retrieved from: https://cornellcob.co.uk/about-us/blog/120-how-to-make-a-cob-brick.html
- Silva, R., Oliveira, D., Schueremans, L., Lourenço, P., & Miranda, T. (2014). Modelling the Structural Behaviour of Rammed Earth Components. Proceedings of the Twelfth International Conference on Computational Structures Technology. Published. https://doi.org/10.4203/ccp.106.112
- Compressed earth block – EARTH ARCHITECTURE. (2021). Earth Architecture. Retrieved from: http://eartharchitecture.org/?tag=compressed-earth-block+Infill
- Home. (2020). Poured Earth. Retrieved from: https://www.pouredearth.net/
- Hueto Escobar, A., Mileto, C., Vegas López-Manzanares, F., & Diodato, M. (2021). Preliminary Analysis of Material Degradation Processes in Half-Timbered Walls with Earth Infill in Spain. Sustainability, 13(2), 772. https://doi.org/10.3390/su13020772
- Project Sudan 2. (2020). UK Rammed Earth. Retrieved from: http://rammed-earth.org/project-sudan-2/
- DiStasio, B. C. (2015). Ajartech Liberator compressed earth bricks. Inhabitat. Retrieved from: https://inhabitat.com/the-liberator-turns-dirt-into-bulletproof-fireproof-building-bricks/ajartech-liberator-compressed-earth-bricks-2/
- Wikipedia contributors. (2021, September 13). Windcatcher. Wikipedia. Retrieved 2 November 2021, from https://en.m.wikipedia.org/wiki/Windcatcher#

8_Appendix

Table of contents

Appendix A: Amount of modules

The ratio between interest and skills per function are calculated in the upper table. The bottom table shows how this translates to the amount of modules that are placed on the plot.

	Amount of people living on the plot					
	Men	17440		Total men	233	%
	Women	17520		Total women	215	%

Category	Skill	Percentage men	Percentage women	Percentage men corrected	Percentage women corrected	Total
Carpet	Pre-processing	4	1	1,7%	0,5%	1,03%
	Dyeing	4	1	1,7%	0,5%	1,03%
	Weaving	5	1	2,1%	0,5%	1,24%
Fabric	Pre-processing fabric	4	0	1,7%	0,0%	0,83%
	Dyeing	4	0	1,7%	0,0%	0,83%
	Tailor	4	0	1,7%	0,0%	0,83%
Food	Field?	18	12	7,7%	5,6%	6,20%
	Food processing	18	11	7,7%	5,1%	5,99%
	Carpet shop	6	3	2,6%	1,4%	1,86%
	Clothing store	6	4	2,6%	1,9%	2,07%
	Greengrocers	7	3	3,0%	1,4%	2,07%
	Restaurants	2	3	0,9%	1,4%	1,03%
Total		82	39			121

	Amount of people per industry	Men	Women
Carpet	Pre-processing	299	81
	Dyeing	299	81
	Weaving	374	81
Fabric	Pre-processing fabric	299	0
	Dyeing	299	0
	Tailor	299	0
Food	Field?	1347	978
	Food processing	1347	896
	Carpet shop	449	244
	Clothing store	449	326
	Greengrocers	524	244
	Restaurants	150	244

Tim

Percentages to numbers of persons

Persons per house	=	5
Amount of people working	=	200

Plot size		
Length	=	120
Width	=	70
Area	=	8400
Grid size:		
Length	=	5
Width	=	5
Area	=	25
Pixels:		
Length	=	24
Width	=	14
Total	=	336
Infrastructure		
% of road	=	7%
% of souks	=	15%

Category	Cluster	Function	Module size in m2	Size in pixels	Amount of modules	Total of pixels	Total m2's (Rounded to 5)
Category		Road	50	2	24	47	1175
		Souks	25	1	50	50	1260
Carpet	Carpet	Pre-processing	50	2	2	4	105
		Dyeing	50	2	2	4	105
		Weaving	50	2	2	5	125
Shops		Carpet shop	50	2	4	7	185
Fabric	Fabric	Pre-processing	25	1	2	2	40
		Dyeing	25	1	2	2	40
		Tailor	25	1	2	2	40
Shops		Clothing store	50	2	4	8	205
Food	Food	Field	25	1	336	336	8400
		Food processing	50	2	12	24	600
Shops		Greengrocers	50	2	4	8	205
Shops		Restaurants	50	2	2	4	105
Essentials		Market square	150	6	3	18	450
		House	50	2	40	80	2000
		Wells	25	1	4	4	100

Grid and module properties

Appendix B: Rules in Python

From rules to floorplan

Date: 19-10-2021 Author: Marnix van den Assum

This code is made to go from written rules to a configuration. In the first part libraries are imported, a grid is made and the canals are added manually. Later the roads are added and wells are placed.

1. Import libraries

```
# importing NumPy for using matrices
import numpy as np

# importing Pandas and display for clearer representation of table data and matrices
import pandas as pd
from IPython.display import display
```

2.1 Define grid size

```
# Define the grid size
len_vertical = 14
len_horizontal = 24
```

2.2 Canals

```
# The canals are manual input as a list of coordinates

stream_1= [

    (0, 12),
    (1, 12),
    (2, 12),
    (3, 12),

    (4, 12),
    (4, 11),
    (4, 10),
    (4, 9),
    (4, 8),
    (4, 7),
    (4, 6),

    (4, 5),
    (5, 5),
    (6, 5),
    (7, 5),
    (8, 5),
    (9, 5),
    (10, 5),
    (11, 5),
    (12, 5),
    (13, 5),
]
```

In this part water, roads and wells are located in the grid.

```
stream_2 = [

    (10, 6),
    (10, 7),
    (10, 8),
    (10, 9),
    (10, 10),
    (10, 11),
    (10, 12),
    (10, 13),
    (10, 14),
    (10, 15),
    (10, 16),
    (10, 17),
    (10, 18),
    (10, 19),
    (10, 20),
    (10, 21),
    (10, 22),
    (10, 23),
]

# Combine the streams
water = (stream_1 + stream_2)

print(water)
```

Python

```
[(0, 12), (1, 12), (2, 12), (3, 12), (4, 12), (4, 11), (4, 10), (4, 9), (4, 8), (4, 7), (4, 6), (4, 5), (5, 5), (6, 5), (7, 5), (8, 5), (9, 5), (10, 5), (11, 5), (12, 5), (13, 5), (10, 6), (10, 7), (10, 8), (10, 9), (10, 10), (10, 11), (10, 12), (10, 13), (10, 14), (10, 15), (10, 16), (10, 17), (10, 18), (10, 19), (10, 20), (10, 21), (10, 22), (10, 23)]
```

2.3 Place canals on the map

```
# VV is a matrix with this shape: number of nodes by number of nodes
len_level_0 = len_vertical, len_horizontal
# Initialize the empty VV
level_0 = np.zeros(len_level_0, dtype=int)

# Iterate over the egdes.
for n1, n2 in water:
    level_0[n1, n2] = 1 # Water pixels are shown as 1

# display as pandas dataframe
map_display = display(pd.DataFrame(level_0))
map_display
```

Python

3. Merchant roads

3.1 Find row and column with most water

```
# Find the row and column with the most water
canal_row = level_0.sum(axis=1).argmax()
canal_column = np.argmax(np.max(level_0, axis=0))

# Count the amount of water modules
water_modules_in_row = np.count_nonzero(level_0[canal_row])
water_modules_in_column = np.count_nonzero(level_0[:,canal_column])

print("Max water row:",canal_row, "Max water column:", canal_column)
print("Water modules in row:", water_modules_in_row, ". Water modules in column:", water_modules_in_column)
```

```
Max water row: 10 Max water column: 5
Water modules in row: 19 . Water modules in column: 10
```

3.2 Find row and column most central

```
# Find the row and column that are most central
central_row = int(len_vertical // 2)
central_column = int(len_horizontal // 2)

print("The central row is:", central_row, "The central column is:", central_column)
```

```
The central row is: 7 The central column is: 12
```

3.3.1 Horizontal road

```
# Potential rows where the roads can be placed
row_above = canal_row -1
row_below = canal_row + 1

potential_roads_horizontal = [row_above,row_below]
print("The potential rows are:", potential_roads_horizontal)

# Function for picking the right one
def find_nearest(potential_roads_horizontal, central_row):
    array = np.asarray(potential_roads_horizontal)
    idx = (np.abs(array - central_row)).argmin(0)
    return array[idx]

# Call function
horizontal_road = find_nearest(potential_roads_horizontal, central_row)

print("Horizontal road:", horizontal_road)
```

```
The potential rows are: [9, 11]
Horizontal road: 9
```

3.3.2 Vertical road

```
# Potential columns where the roads can be placed
column_above = canal_column -1
column_below = canal_column + 1

potential_roads_vertical = [column_above,column_below]
print("The potential columns are:", potential_roads_vertical)

# Function for picking the right one
def find_nearest(potential_roads_vertical, central_column):
    array = np.asarray(potential_roads_vertical)
    idx = (np.abs(array - central_column)).argmin(0)
    return array[idx]

# Call function
vertical_road = find_nearest(potential_roads_vertical, central_column)

print("vertical road:", vertical_road)
```

```
The potential columns are: [4, 6]
vertical road: 6
```

3.4 Plot roads on the grid

```
# Plot roads on the grid
level_0[ horizontal_road, : ] = [2]
level_0[ :, vertical_road ] = [2]

print(level_0)
```

```
[[0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
 [0 0 0 0 0 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

3.5 List of the coordinates of road

```
# Make a list of merchant roads. The output is used when going to a 3D matrix (distinct between floors)
roads_list = np.argwhere(level_0 == 2)
roads_list
```

4. Sub-roads

+ Code + Markdown

4.1 Define sub-roads and plot on map

```
# Define distance between roads
distance_between_roads = 6

# Find the location of the sub-roads
sub_roads = np.arange(start=vertical_road, stop=len_horizontal, step=distance_between_roads)

# Place the sub-roads in the grid
level_0[:, sub_roads] = [2]

print (level_0)
```

4.1 List of the coordinates of roads and sub-roads

```
# Make a list of sub-roads. The output is used when going to a 3D matrix (distinct between floors)
sub_roads_list = np.argwhere(level_0 == 2)
sub_roads_list
```

5.1 Possible places for well

```
# Make a list of all coordinates where a well can be placed. This means finding the water modules
potential_wells = np.argwhere(level_0 == 1)

# Function that makes tuples that can be used to place wells on the grid
def make_tuples(a):
    try:
        return tuple(make_tuples(i) for i in a)
    except TypeError:
        return a

# Run the function with the potential well spots
potential_wells_list = list(make_tuples(potential_wells))

# Define the starting point of the wells and define the distance
start = 3
distance = 5

# Find and replace water with wells
for i in range(start,len(potential_wells_list),distance):
    if level_0[potential_wells_list[i][0],potential_wells_list[i][1]] == 1:
        level_0[potential_wells_list[i][0],potential_wells_list[i][1]] = 5

for row in level_0:
    print(row)
```

5.2 List of coordinates of wells

+ Code + Markdown

```
# Make a list of wells. The output is used when going to a 3D matrix (distinct between floors)
wells = np.argwhere(level_0 == 5)
wells
```

Explanation script

Doris van Uffelen 31-10-2021

In this script a location for a market square is found in our plot. It uses an example grid and not the actual size of the grid for clarity. First, a base grid, with the functions placed before is used for creating the needed contextual information. Then, the voxels of the square are placed one by one. Each time the script runs, a different possible outcome is generated.

Import and install

All needed libraries are imported and installed.

Plot maps

Consists of functions that can later be used to plot the lattices

Setting the base grid

An example grid is manually filled in, to use as base and information for the rest of the grid. This can be exchanged by input from the actual grid. Also a lattice is made for all free space.

Find neighbours

A neumann stencil is created for finding neighbours. Then a function is written for finding neighbours.

A lattice is made with all voxels next to a road, the function and stencil for finding neighbours are used. A lattice is made with all voxels not next to a road, the function and stencil for finding neighbours are used.

Find location for square

Function for choosing 1 random possible location out of a list. Function for finding direct neighbours.

A location for the square is found by placing each voxel one by one after each other. All voxels have their own conditions.

There are multiple possibilities, each time the plot runs it finds a new location.

In this part squares are located in the grid.

import and install

```
# !python -m pip install -e ../topogenesis
```

```
# importing NumPy and NetworkX for working with graphs
import numpy as np
import networkx as nx

# importing topogenesis for working with lattices / regular-grids
import topogenesis as tg

# importing pyvista for visualizations
import pyvista as pv

# importing Pandas and display for clearer representation of table data and matrices
import pandas as pd
from IPython.display import display

import matplotlib.pyplot as plt
```

Plot maps

```
# function for printing grids with custom color maps

def image(base_lattice, color_map):
    # Change the shape of the lattice so only the groundfloor is plotted
    lattice = np.copy(base_lattice)
    lattice.transpose(1, 0, 2)
    color_lattice = lattice[:, 1, :]

    # Add custom color map
    image = np.array([[color_map[val] for val in row] for row in color_lattice], dtype='int')
    return plt.imshow(image)
```

```
# define color map
color_map = {0: np.array([255, 255, 255]), # white
             2: np.array([152, 141, 96]), # green/grey
             3: np.array([178, 192, 163]), # light grey
             4: np.array([70, 83, 99]), # dark blue
             5: np.array([199, 143, 82])} # orange
```

+ Code + Markdown

```
#  function for printing grids with only 0 and 1

def boolean_image(base_lattice):
    # Change the shape of the lattice so only the groundfloor is plotted
    lattice = np.copy(base_lattice)
    lattice.transpose(1, 0, 2)
    color_lattice = lattice[:, 1, :]

    return plt.imshow(color_lattice, "Greens")
```

setting the base grid

```
# Functions:

Functions = {
    "empty" : 0,
    "water" : 1,
    "road" : 2,
    "souk" : 3,
    "well" : 4,
    "square" : 5,
    "Water workshop" : 6,
    "non-water workshop" : 7,
    "shop" : 8,
    "house" : 9}

```

```
# Define the grid
len_x = 7
len_y = 7
len_z = 3

# length and width of matrix
len_grid = len_y, len_z, len_x
# Initialize the empty matrix
base_grid = np.zeros(len_grid, dtype=int)

# place water
base_grid[4, 2, :] = 1

# place main road
base_grid[3, 1, :] = 2
base_grid[:, 1, 2] = 2

# place souk
base_grid[:3, 1, 6] = 3

# place well
base_grid[4, 1, 4] = 4

original_base_lattice = tg.to_lattice(base_grid, minbound=0, unit=1)
image(original_base_lattice, color_map)
```

```
# Find all voxels without functions and mark them with 1, all locations with functions become 0
bool_base_lattice = (original_base_lattice > 0).astype(int)
free_space_lattice = 1 - bool_base_lattice

boolean_image(free_space_lattice)
```

Find neighbours

create neighbour stencil

```
# creating neuman neighborhood definition
neumann_stencil = tg.create_stencil("von_neumann", 1, 1)
# setting the center and -1 and 1 floor to zero
neumann_stencil[1,:,1] = 0

boolean_image(neumann_stencil)
```

function for finding neighbours

```
# main loop for breath-first traversal
def dist_lattice(index_func,neighs,max_dist, base_lattice):
    func_lattice = base_lattice == index_func
    func_neighs_lattice = 1 - func_lattice
    func_neighs_lattice[func_neighs_lattice == 1] = max_dist
    dist_lattice_flat = func_neighs_lattice.flatten()

    for i in range(1, max_dist):
        # find the neighbours of the previous step
        next_step = neighs[dist_lattice_flat == i - 1]
        # make a copy of the lattice to prevent overwriting in the memory
        next_dist_lattice_flat = np.copy(dist_lattice_flat)
        # set the next step cells to the current distance
        next_dist_lattice_flat[next_step.flatten()] = i
        # find the minimum of the current distance and previous distances to avoid overwriting previous steps
        dist_lattice_flat = np.minimum(dist_lattice_flat, next_dist_lattice_flat)

        # check how many of the cells have not been traversed yet
        filled_check = dist_lattice_flat == max_dist
        # if all the cells have been traversed, break the loop
        if filled_check.sum() == 0:
            print(i)
            break

    # reshape and construct a lattice from the street network distance list
    dist_lattice = dist_lattice_flat.reshape(base_lattice.shape)

    return dist_lattice
```

find voxels next to streets

```
# input neighs function
neighs_streets = original_base_lattice.find_neighbours(neumann_stencil)
max_dist = np.sum(original_base_lattice.shape)
index_street = 2

# find neighbours of the streets
street_neigh_lattice = dist_lattice(index_street,neighs_streets,max_dist, original_base_lattice)
street_neigh_lattice[street_neigh_lattice >= 2] = 0
street_neigh_lattice[street_neigh_lattice == 1] = max_dist

boolean_image(street_neigh_lattice)
```

find voxels not next to streets

```
# Using input from previous street_dist_lattice invert lattice to find all locations not next to a road
street_no_neigh_lattice = np.copy(street_neigh_lattice)
street_no_neigh_lattice[street_no_neigh_lattice == 0] = 1
street_no_neigh_lattice[street_no_neigh_lattice == max_dist] = 0

boolean_image(street_no_neigh_lattice)
```

find location for square

```
def locate_func(func_dist_lattice):
    # Find location
    possible_locations = np.flatnonzero(func_dist_lattice == np.max(func_dist_lattice))
    best_location = np.random.choice(possible_locations , size = None, replace = False, p = None)
    return best_location

def voxel_neigh_lattice(best_location):
    # make empty lattice
    voxel_grid = np.zeros(original_base_lattice.shape, dtype=int)
    voxel_lattice = tg.to_lattice(voxel_grid, minbound=0, unit=1)
    np.put(voxel_lattice, [best_location], [1])

    # find neighbours
    neighs_voxel = voxel_lattice.find_neighbours(neumann_stencil)
    max_dist = np.sum(voxel_lattice.shape)
    voxel_neigh_lattice = dist_lattice(1,neighs_voxel,max_dist,voxel_lattice)
    voxel_neigh_lattice[voxel_neigh_lattice >= 2] = 0
    voxel_neigh_lattice[voxel_neigh_lattice == 1] = 2
    return voxel_neigh_lattice
```

```
# locate first voxel of square
func_dist_lattice = street_neigh_lattice * free_space_lattice
first_voxel = locate_func(func_dist_lattice)

# find neighboours of first voxel
first_voxel_neigh_lattice = voxel_neigh_lattice(first_voxel)


# locate second voxel
func_dist_lattice = first_voxel_neigh_lattice * street_neigh_lattice * free_space_lattice
second_voxel = locate_func(func_dist_lattice)

# find neighboours of second voxel
second_voxel_neigh_lattice = voxel_neigh_lattice(second_voxel)


# locate third voxel
func_dist_lattice  = second_voxel_neigh_lattice * street_no_neigh_lattice * free_space_lattice
third_voxel = locate_func(func_dist_lattice)

# find neighboours of third voxel
third_voxel_neigh_lattice = voxel_neigh_lattice(third_voxel)


# locate fourth voxel
func_dist_lattice = first_voxel_neigh_lattice * third_voxel_neigh_lattice * street_no_neigh_lattice * free_space_lattice
fourth_voxel = locate_func(func_dist_lattice)

# Add voxels to list
indeces_square = []
indeces_square.append(first_voxel)
indeces_square.append(second_voxel)
indeces_square.append(third_voxel)
indeces_square.append(fourth_voxel)

# place squares in lattice
square_lattice = np.copy(original_base_lattice)
np.put(square_lattice, indeces_square, 5)

print(indeces_square)
image(square_lattice, color_map)
```

Appendix C: Bubble diagram in Python

```python
# import needed library's
import pandas as pd
import numpy as np
import networkx as nx
```

```python
#inputs

#number of specific function
num_shop = 24
num_square = 2
num_workshop = num_shop // 2
num_storage = num_shop
num_anchors = 4

#create a list with all of the names of all the functions
functions = ["anchor"] * num_anchors + ["square"] * num_square + ["shop"] * num_shop + ["storage"] * num_storage + ["workshop"] * num_workshop
#calculate the total number of functions
num_functions = len(functions)
print("Total number of functions:",num_functions)

#m2
area = {
"shop" : 30,
"square" : 900,
"workshop" : 60,
"storage" : 10,
"anchor" : 1
}
```

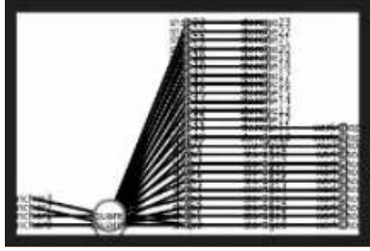
```
Total number of functions: 66
```

```python
#make a table with all the data regarding the functions

last_function = ""
i = 0
function_data = []
for func in functions:
    if last_function != func:
        i = 0
    space_i = func + str(i)
    function_data.append([func, space_i, area[func]])
    last_function = func
    i += 1

#pd.DataFrame(function_data, columns=["function_name", "space_index", "area"])
```

```python
#create a list to store all of the connections
con = []
point = []

#make the node list without other information
for m in range(num_functions):
    point.append(m)

#create some empty lists to keep track of items that are already used
used_shops = []
used_storage = []
used_workshop = []
square_index = []
used_anchor = []
ansq_index = []


#find all of the connections
for b in range(num_functions):
    if function_data[b][0] == 'anchor':
        for l in range(num_functions):
            if function_data[l][0] == 'square':
                if b not in used_anchor:
                    if len(ansq_index) == num_square:
                        ansq_index.clear()
                    if l not in ansq_index:
                        con.append((b,l))
                        ansq_index.append(l)
                        used_anchor.append(b)
    if function_data[b][0] == 'square':
        for l in range(num_functions):
            if function_data[l][0] == 'square':
                if (b,l) not in con and (l,b) not in con and b != l:
                    con.append((b,l))
    st_sh = 0
    sh_ws = 0
    if function_data[b][0] == 'shop':
        for i in range(num_functions):
            if function_data[i][0] == 'storage':
                if st_sh < 1:
                    if i not in used_storage and b != i:
                        con.append((b,i))
                        used_storage.append(i)
                        st_sh = 1
            if function_data[i][0] == 'workshop':
                if sh_ws < 1:
                    if i not in used_workshop and b != i:
                        con.append((b,i))
                        used_workshop.append(i)
                        sh_ws = 1
            if function_data[i][0] == 'square':
                if b not in used_shops:
                    if len(square_index) == num_square:
                        square_index.clear()
                    if i not in square_index:
                        con.append((b,i))
                        square_index.append(i)
                        used_shops.append(b)


#assign some shops to a second square
dbl_shop = []
for b in range(num_functions):...

#check the connections visualy
print(con)
```

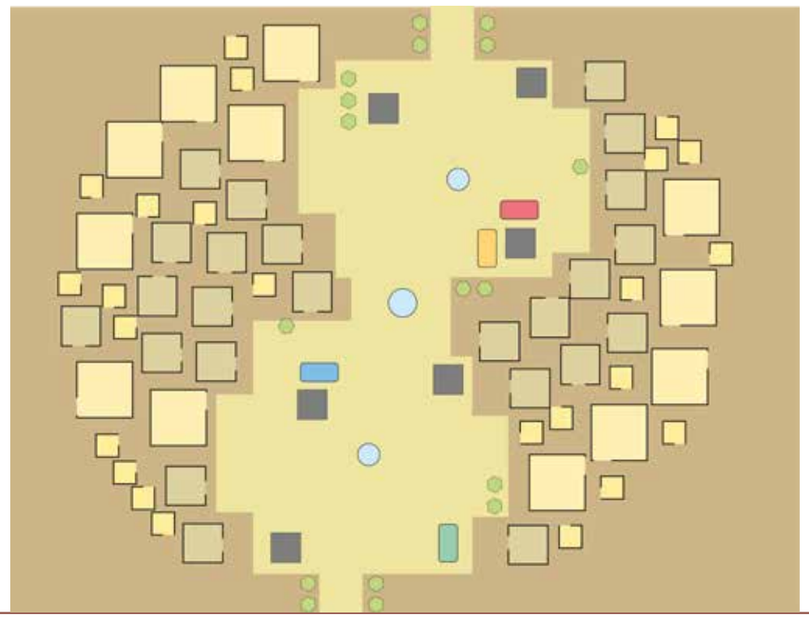
```python
#shervins piece of code from the workshop

G = nx.Graph()
G.add_nodes_from(point)
G.add_edges_from(con)

#find the size of the connection matrix
Matrix_size = (num_functions, num_functions)
# Make the empty connection matrix
Con_matrix = np.zeros(Matrix_size, dtype=int)
# iterate over the egdes
for n1, n2 in con:
    Con_matrix[n1, n2] = 1
    Con_matrix[n2, n1] = 1

#display as pandas dataframe
#display(pd.DataFrame(Con_matrix))
```

```python
#give all of the nodes a position
positions = {}

last_function = ""

x_pos = 0
y_pos = 0

r = 0
for func in functions:
    if last_function != func:
        x_pos = x_pos+500
        y_pos = 0
    positions[r] = (x_pos,y_pos)
    last_function = func
    r += 1
    y_pos = y_pos+500

#make a dictonary with labels for all of the functions
labels = {}
labels_1 =[]
for w in range(num_functions):
    labels[w] = function_data[w][1]
    labels_1.append(function_data[w][1])

#give all the nodes the correct size
function_area = []
for name in functions:
    function_area.append(area[name])

#select the options for the graph
options = {
    "font_size": 10,
    "node_size": function_area,
    "node_color": "white",
    "edgecolors": "grey",
    "edge_color": "black",
    "linewidths": 3,
    "width": 3,
    "labels": labels,
}

#print the graph
nx.draw_networkx(G, positions, with_labels=True, **options)
```



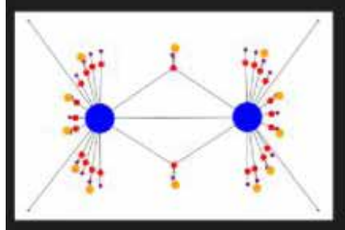
```python
G = nx.Graph()

for b in range(num_functions):...

elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] > 0.5]

fixed_pos = {
    "anchor0": (0, 0),
    "anchor1": (240, 0),
    "anchor2": (0, 70),
    "anchor3": (240, 70)}

fixed_nodes = fixed_pos.keys()

pos_1 = nx.spring_layout(G, pos= fixed_pos, fixed= fixed_nodes, iterations=10000)
```

```python
#move the shops to the square's
for b in range(num_functions):
    if function_data[b][0] == 'shop':
        if b not in dbl_shop:
            for l in range(num_functions):
                if function_data[l][0] == 'square':
                    if (b,l) in con or (l,b) in con:
                        locsh = pos_1[function_data[b][1]]
                        locsq = pos_1[function_data[l][1]]
                        cur_dist = ( (locsq[0] - locsh[0])**2 + (locsq[1] - locsh[1])**2 )**0.5

                        des_dist = (function_data[b][2]/np.pi)**0.5 + (function_data[l][2]/np.pi)**0.5

                        dist_x = abs(locsq[0] - locsh[0])
                        dist_y = abs(locsq[1] - locsh[1])

                        if locsh[0] > locsq[0]:
                            new_x = locsq[0]+ dist_x/(cur_dist/des_dist)
                        if locsh[0] < locsq[0]:
                            new_x = locsq[0]- dist_x/(cur_dist/des_dist)
                        if locsh[1] > locsq[1]:
                            new_y = locsq[1]+ dist_y/(cur_dist/des_dist)
                        if locsh[1] < locsq[1]:
                            new_y = locsq[1]- dist_y/(cur_dist/des_dist)

                        pos_1[function_data[b][1]]=(new_x, new_y)

#move storage to shop
for b in range(num_functions):...

#move workshops to shops
for b in range(num_functions):...
#print(pos_1)
```



```python
pos_keys = [*pos_1]
p = len(pos_keys)
ordered_area = []
ordered_func = []
for i in range(p):
    index = labels_1.index(pos_keys[i])
    ordered_area.append(function_data[index][2])
    ordered_func.append(function_data[index][0])

#colors
color_func = {
"shop" : "red",
"square" : "blue",
"workshop" : "orange",
"storage" : "purple",
"anchor" : "black"
}
color_list = []
for func in ordered_func:
    color_list.append(color_func[func])


# nodes
nx.draw_networkx_nodes(G, pos_1,node_size=ordered_area, node_color=color_list)
# edges
nx.draw_networkx_edges(G, pos_1, edgelist=elarge, width=1, edge_color="grey")
# labels
#nx.draw_networkx_labels(G, pos_1, font_size=10, font_family="sans-serif", font_color="black")
```

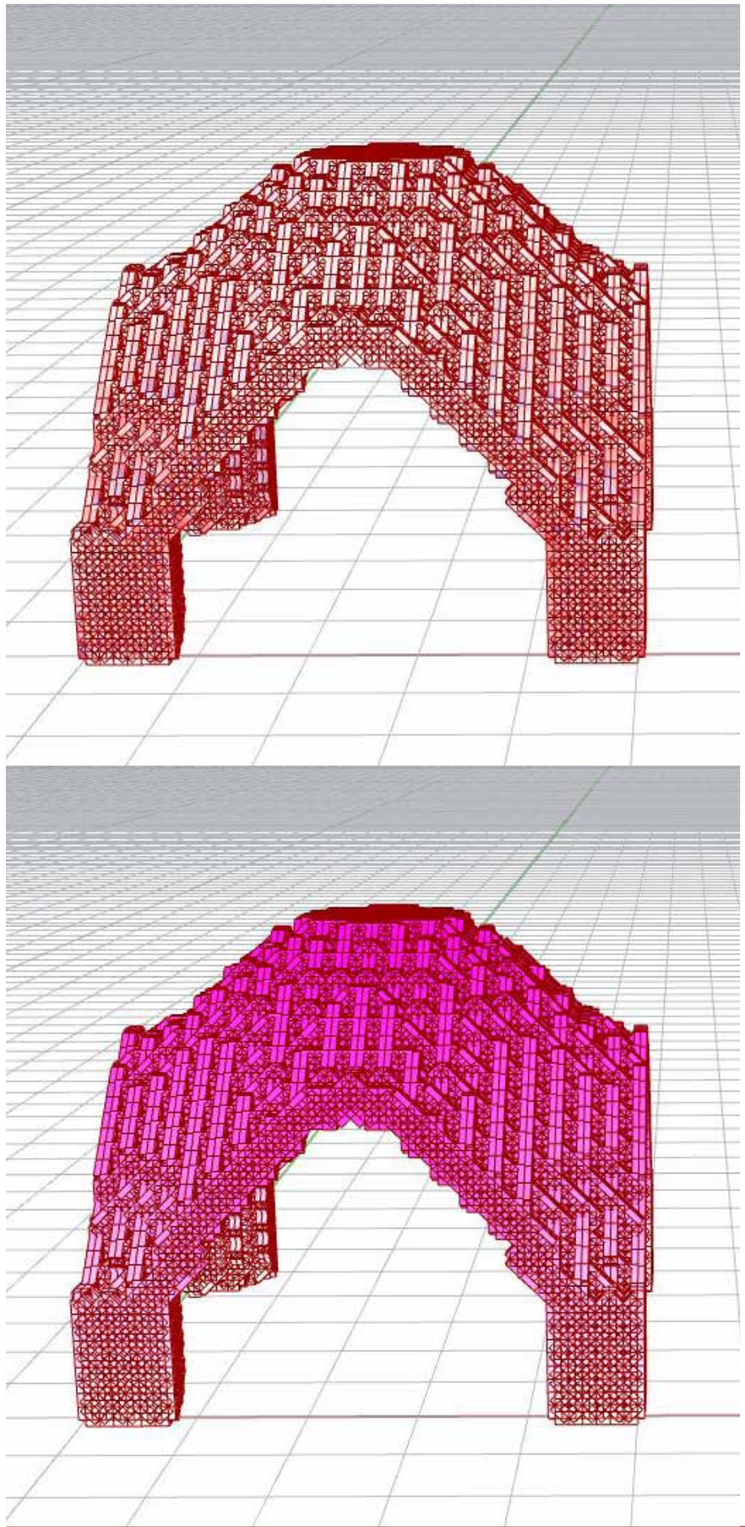
```
<matplotlib.collections.LineCollection at 0x15b22bda2e0>
```

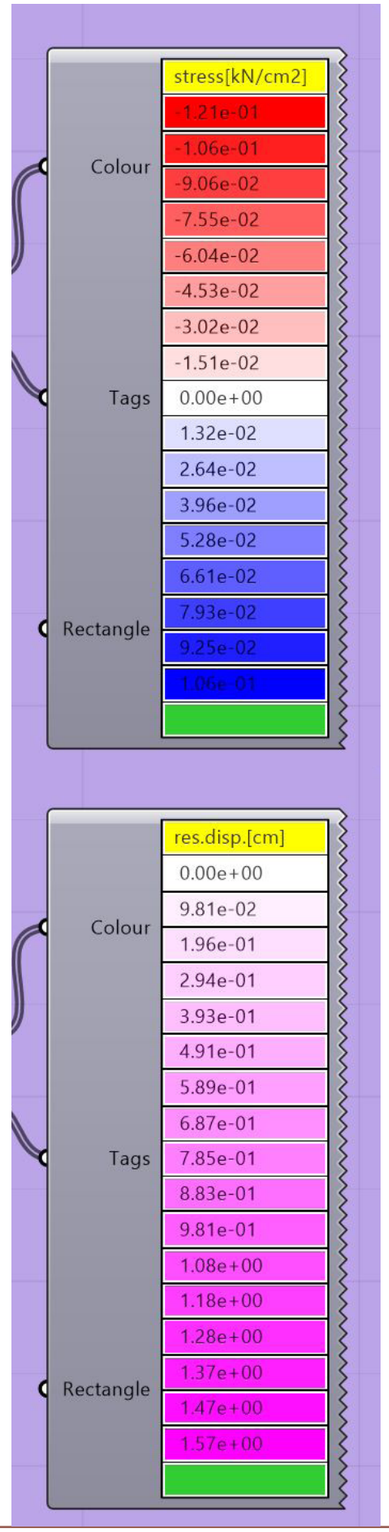


```python
#prepare the data for exporting to a file
#pos_1_list = []
#for k, v in pos_1.items():
#    pos_1_list.append(tuple(v))
#
#pos_1_arr = np.array(pos_1_list)
#area_arr = np.array(ordered_area)
```

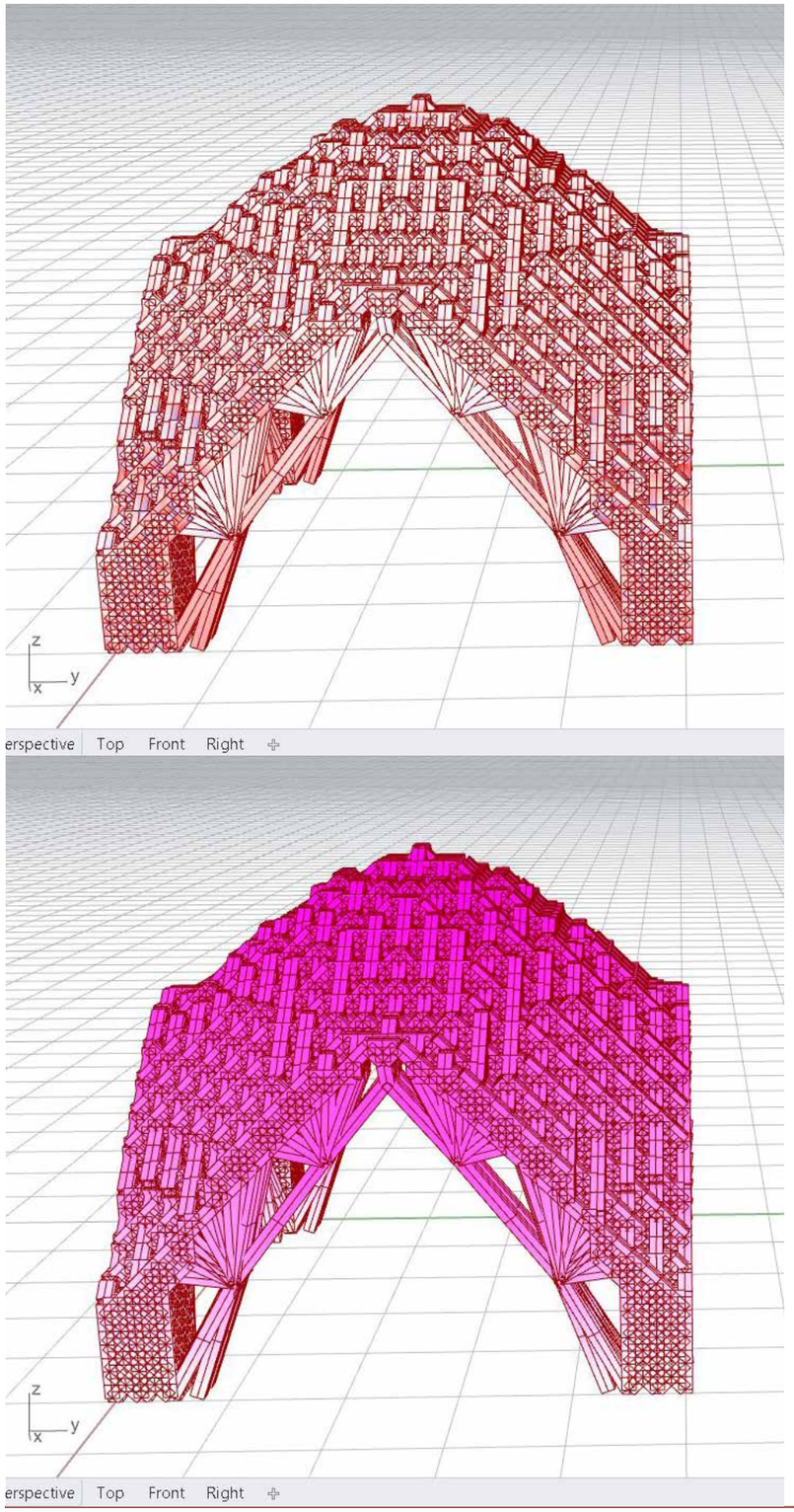
```python
#save the points to a file
#np.savetxt("points", pos_1_arr)
#np.savetxt("areas", area_arr)
```

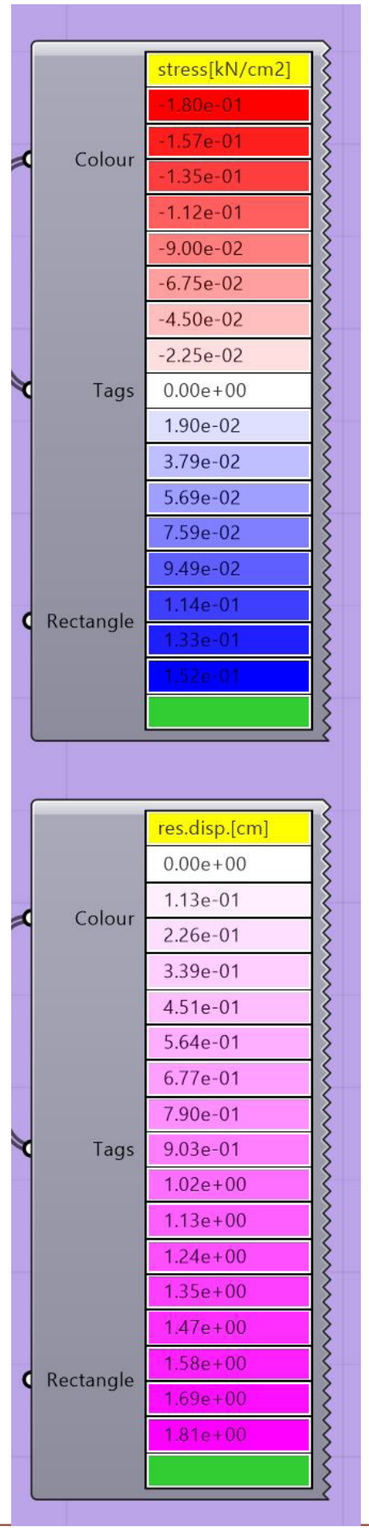
Appendix D: FEM calculation module



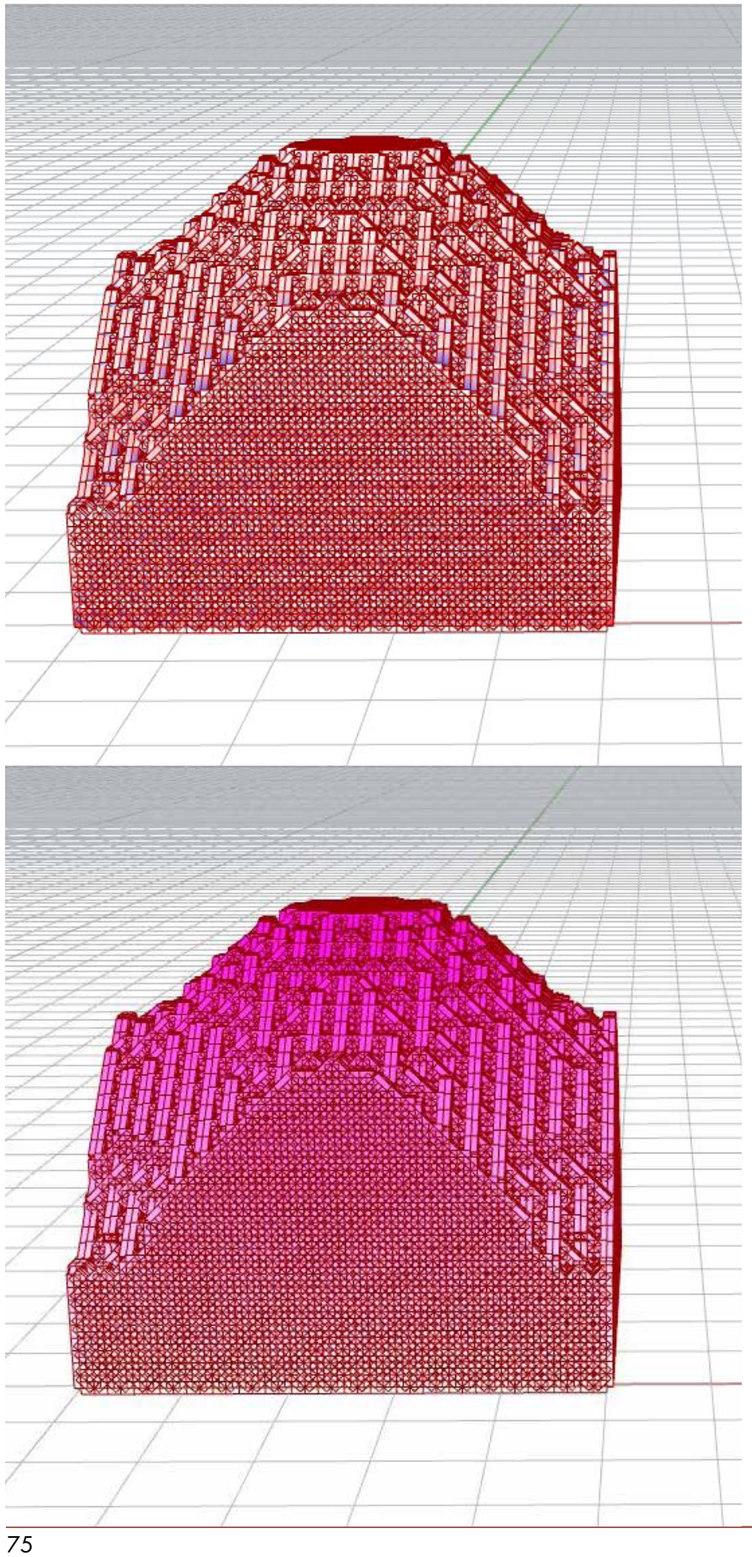


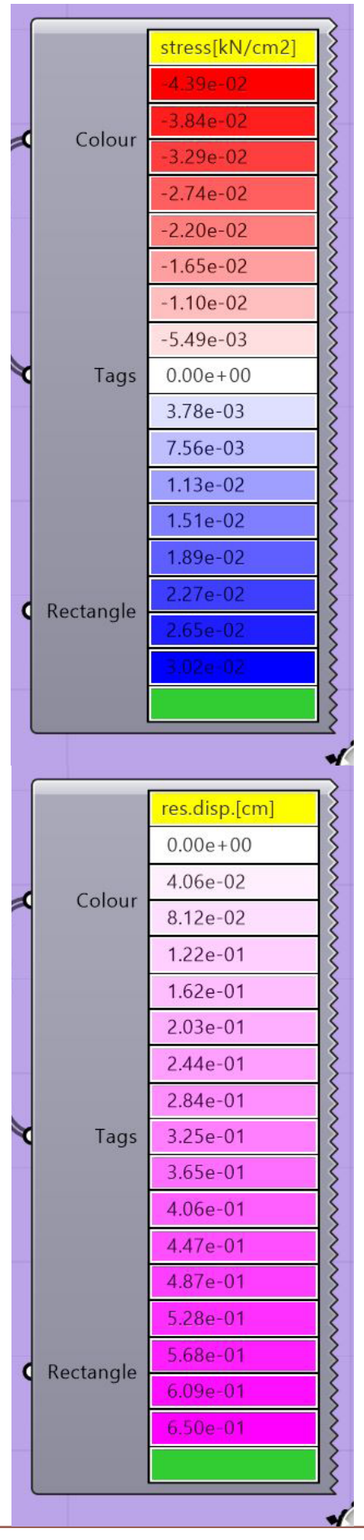
module with arch



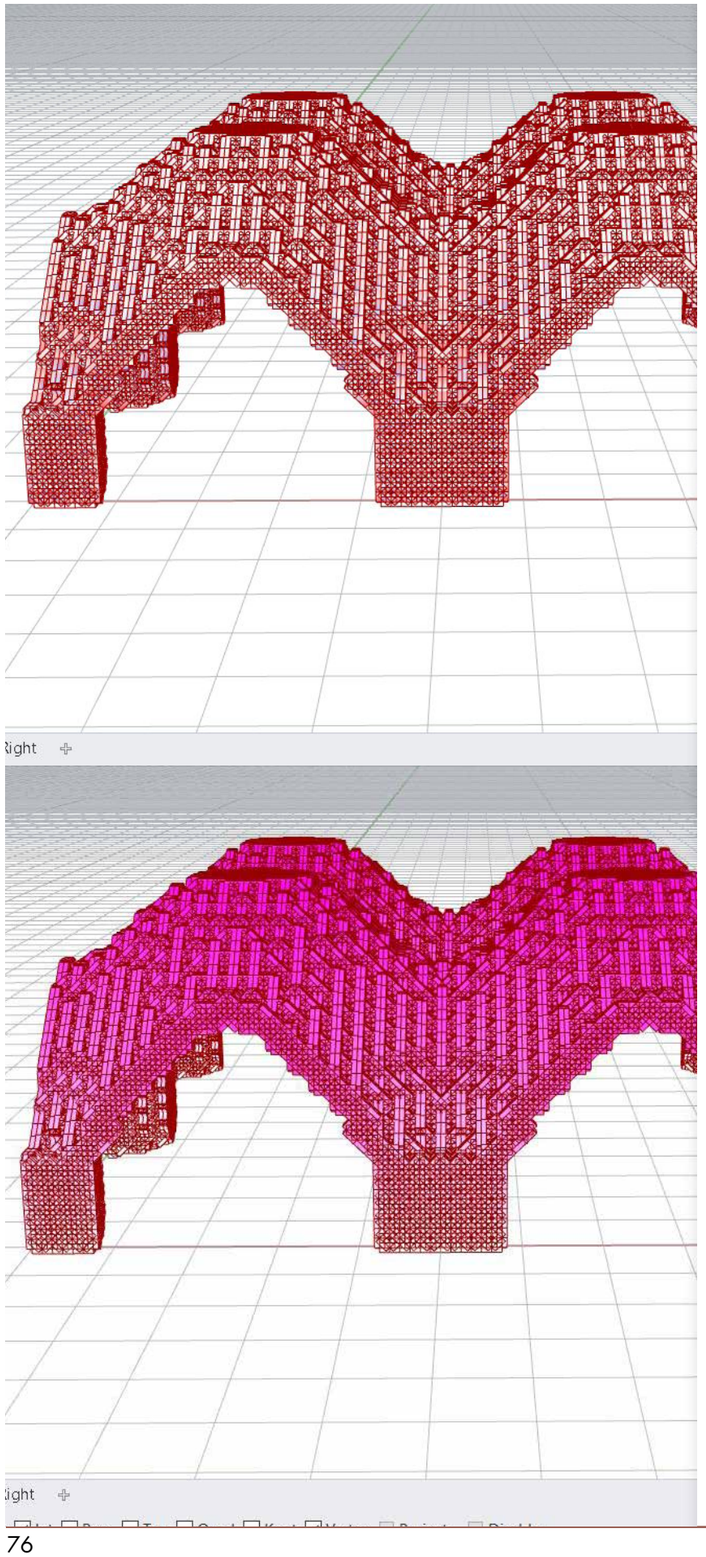


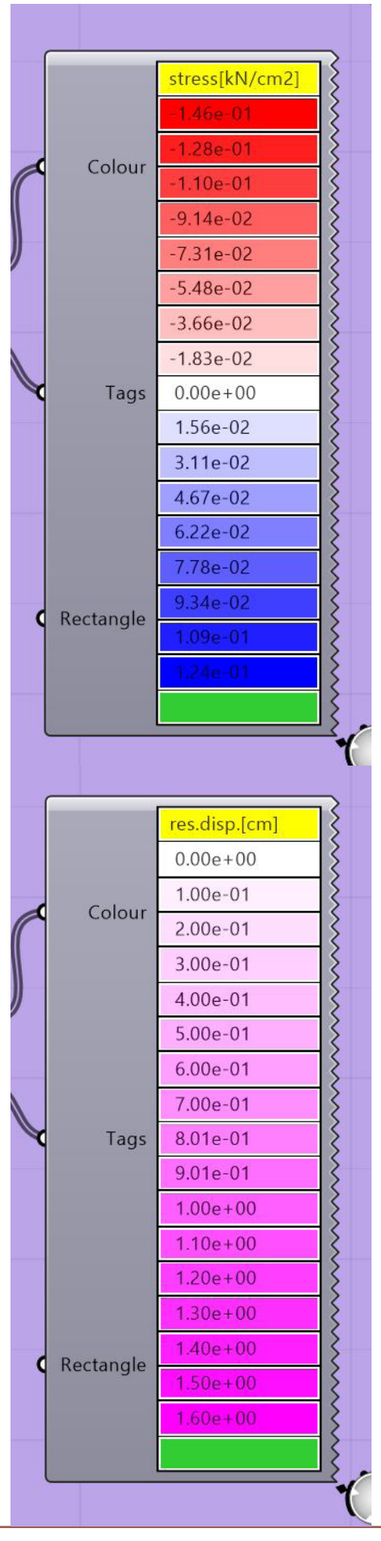
module with walls



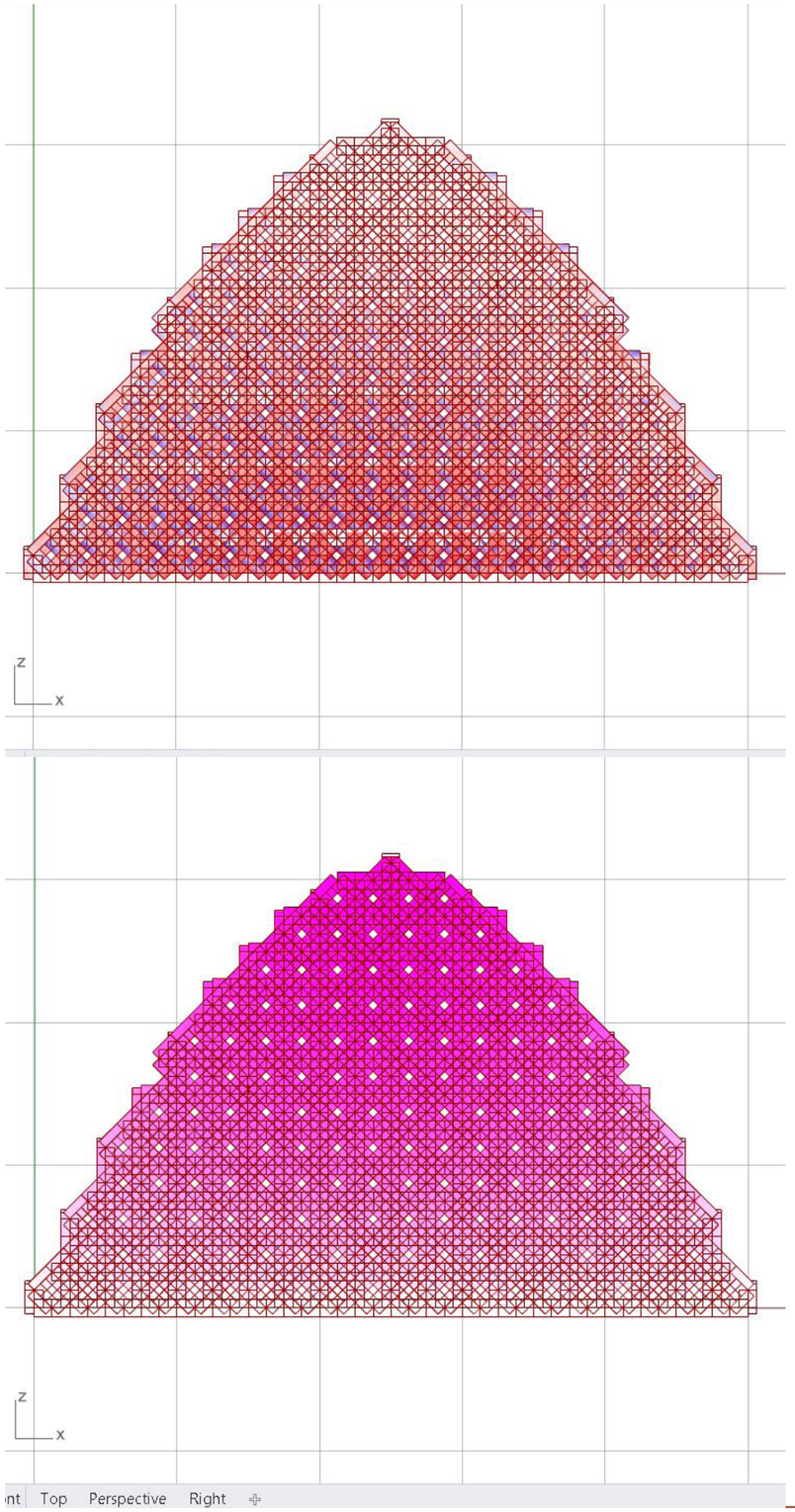


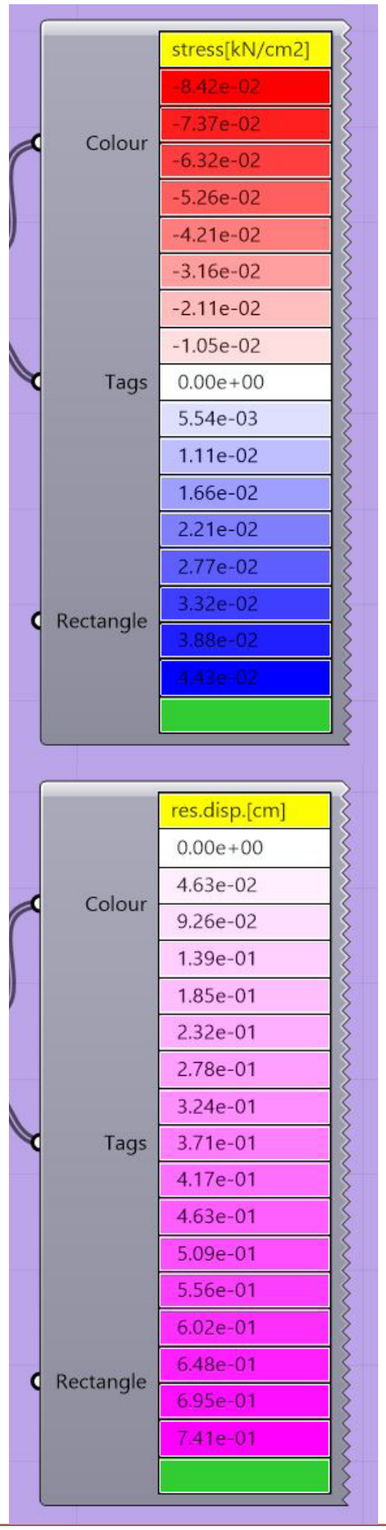
four modules



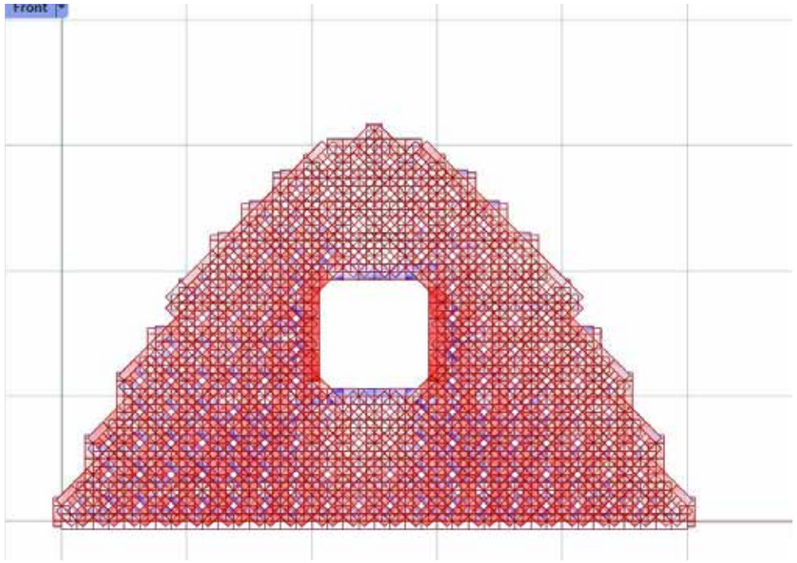


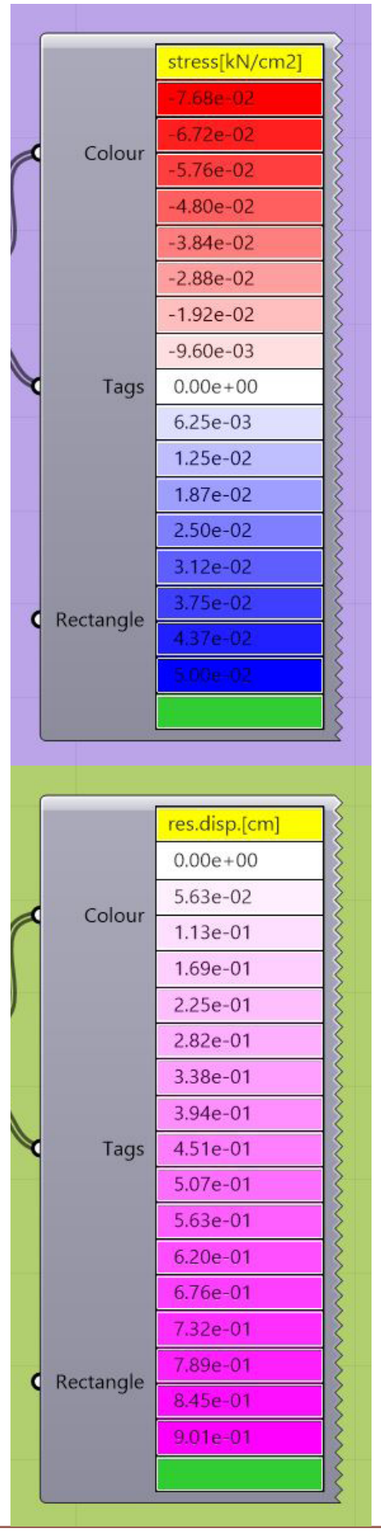
Appendix E: FEM calculation Walls

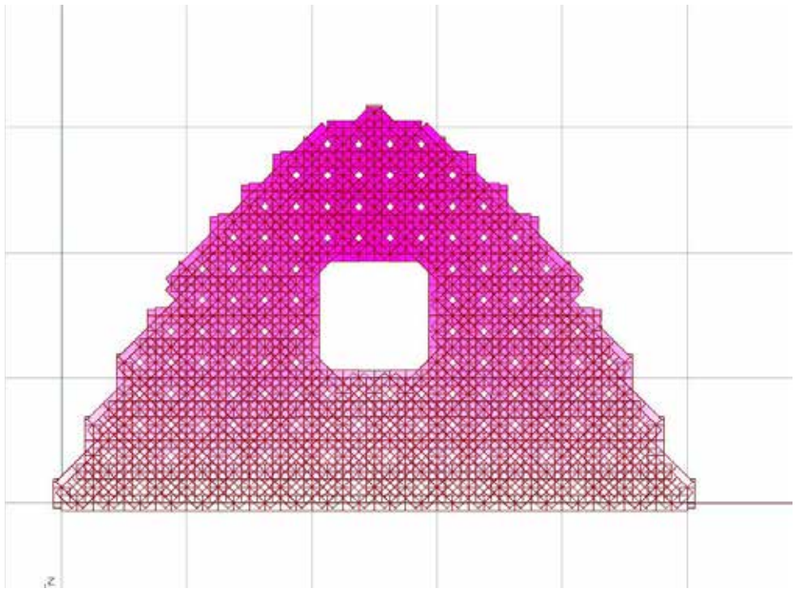




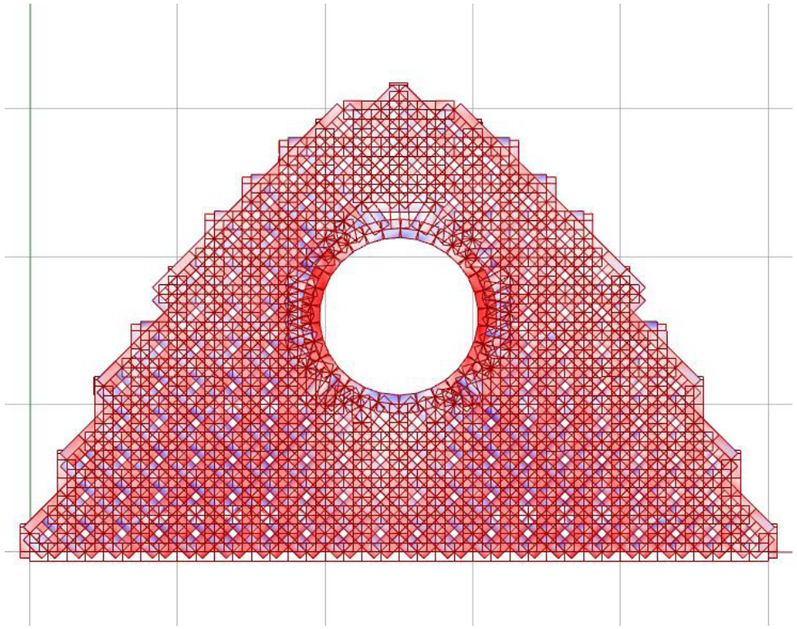
wall square window

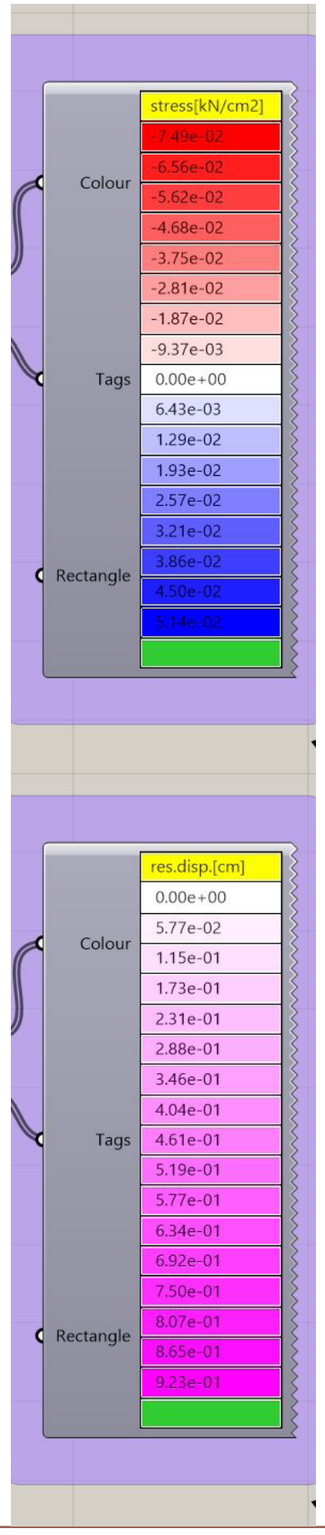


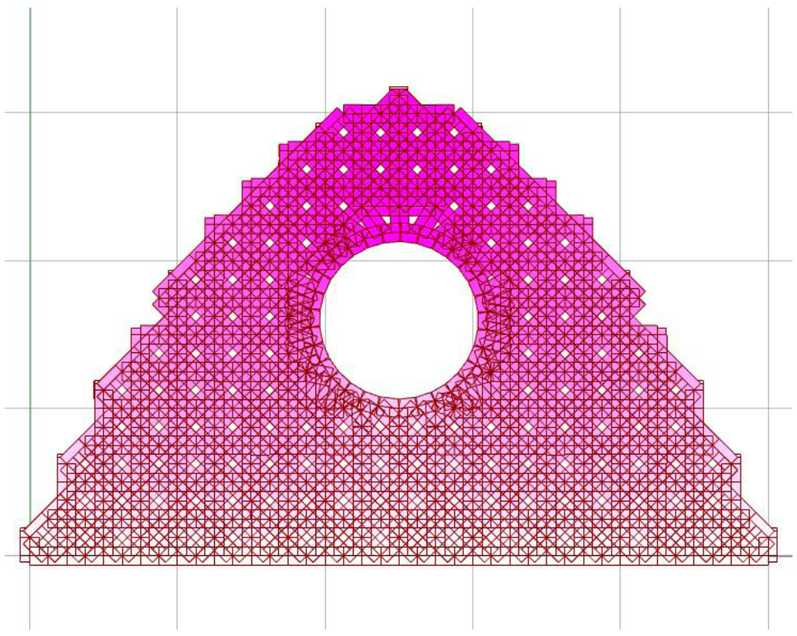




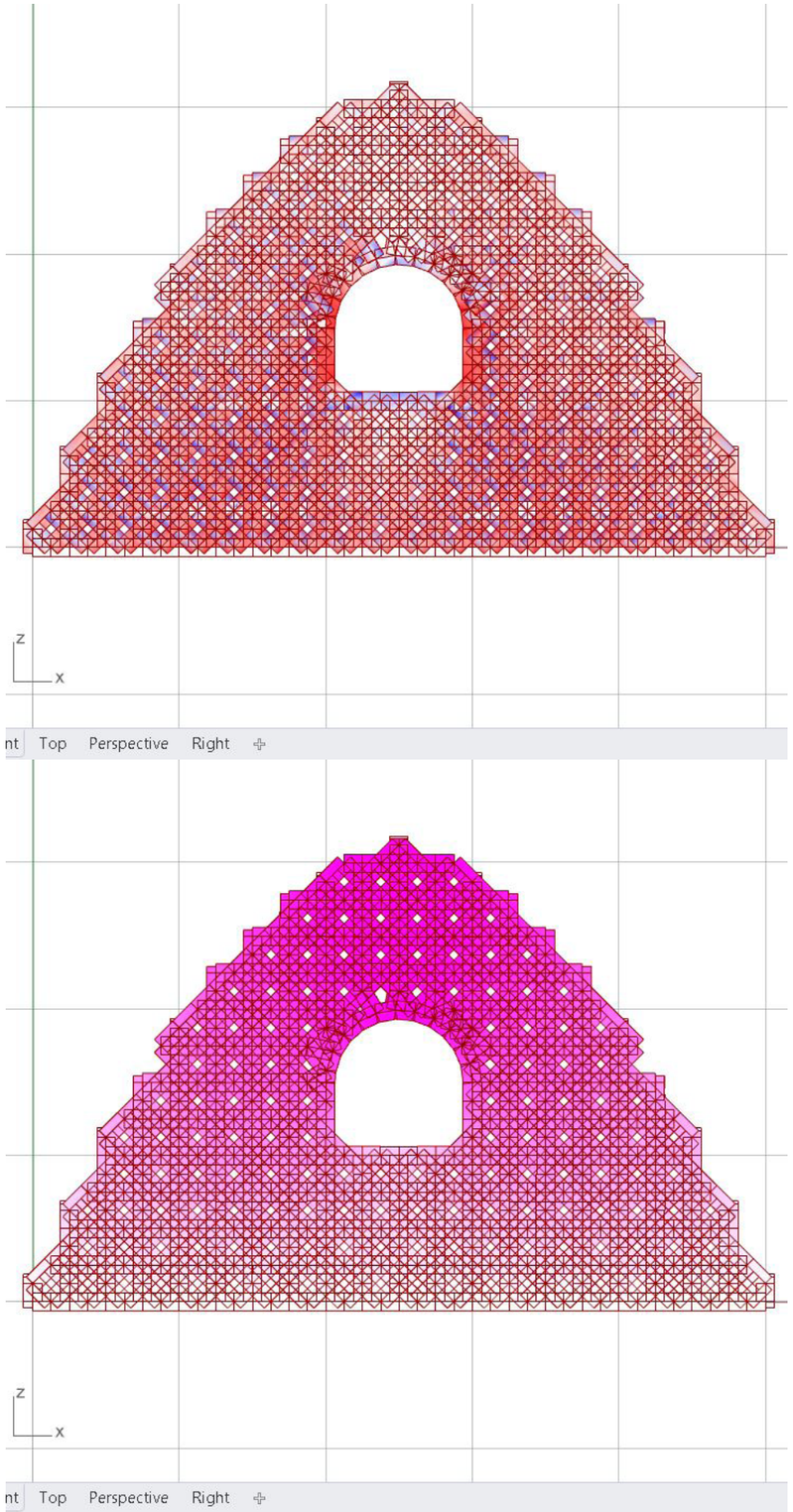
wall round window

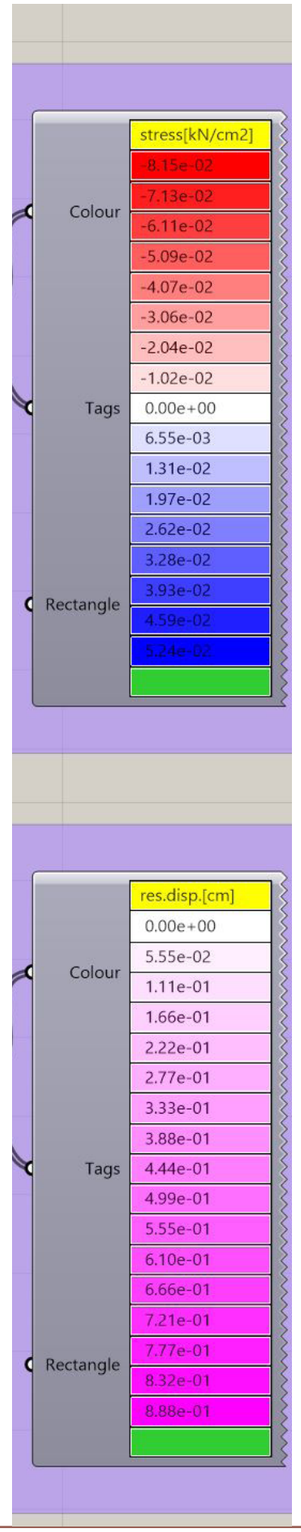




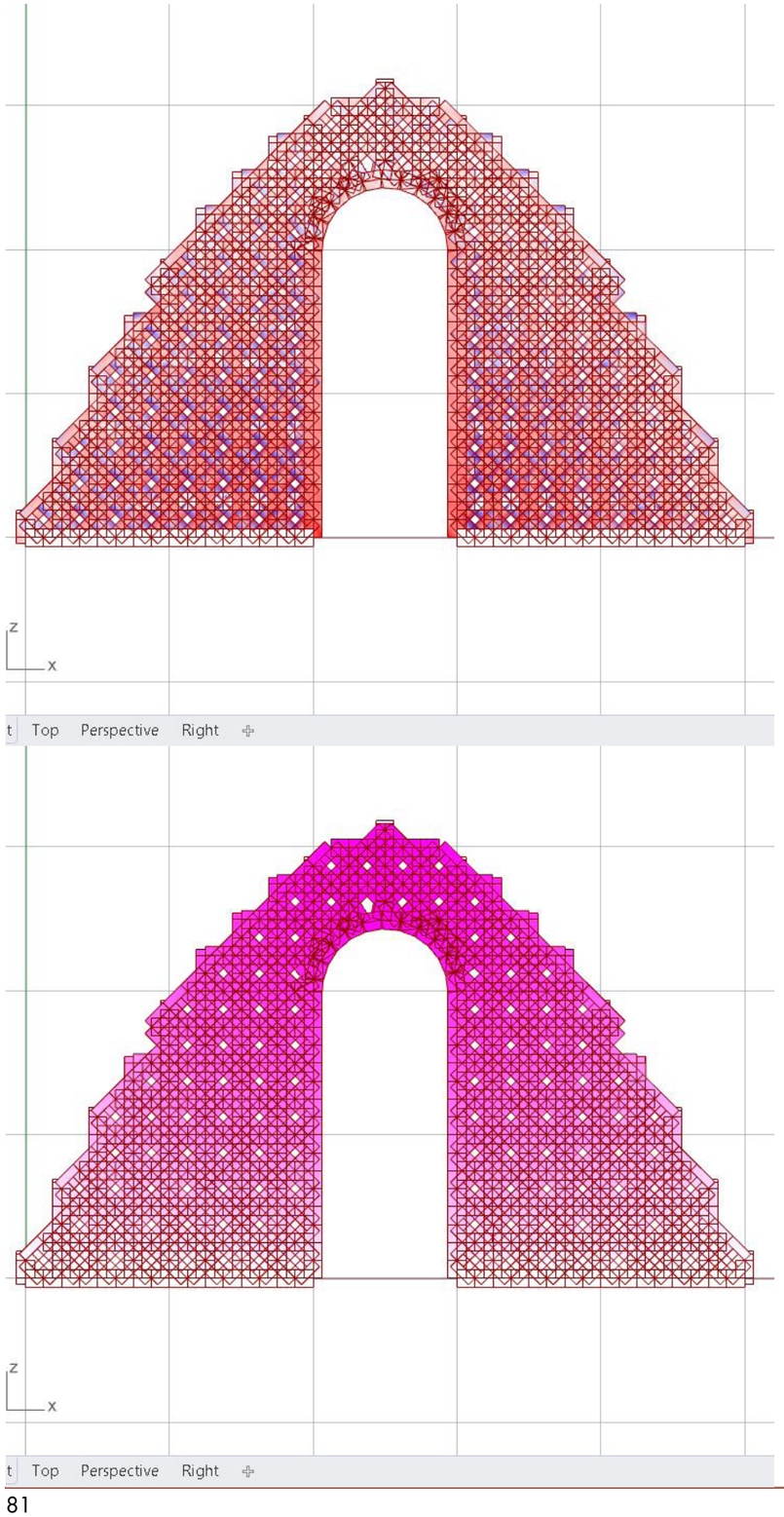


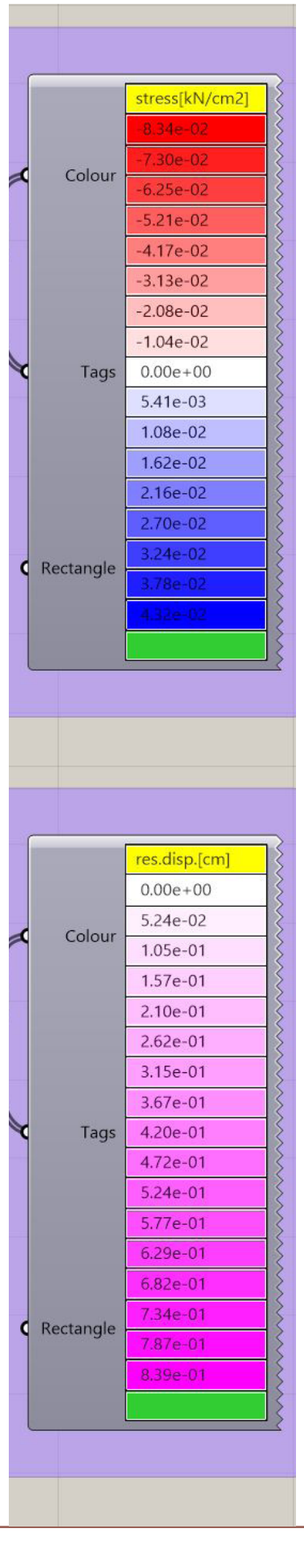
wall semi round window



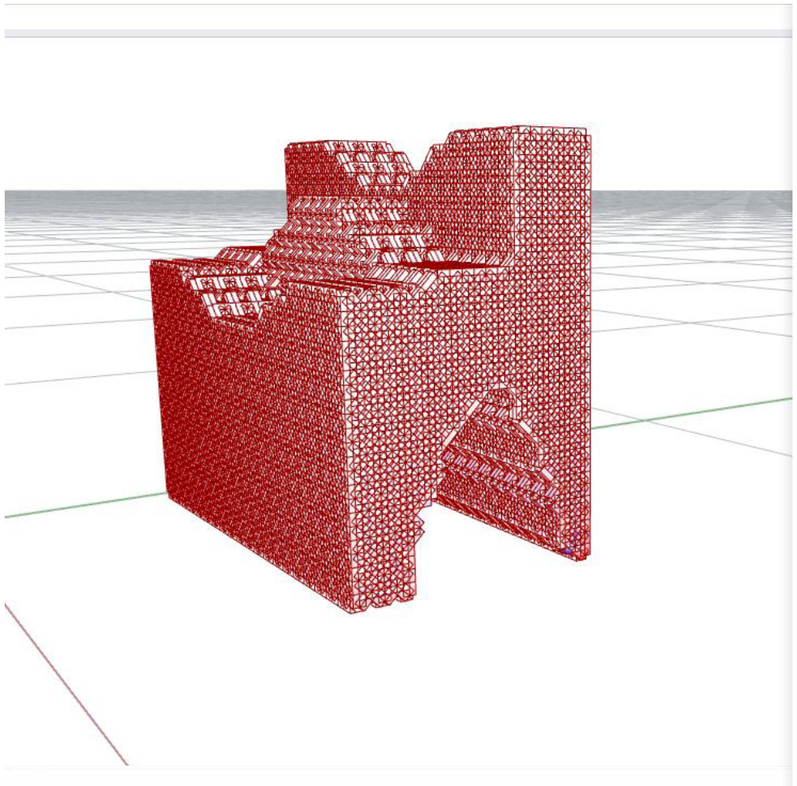


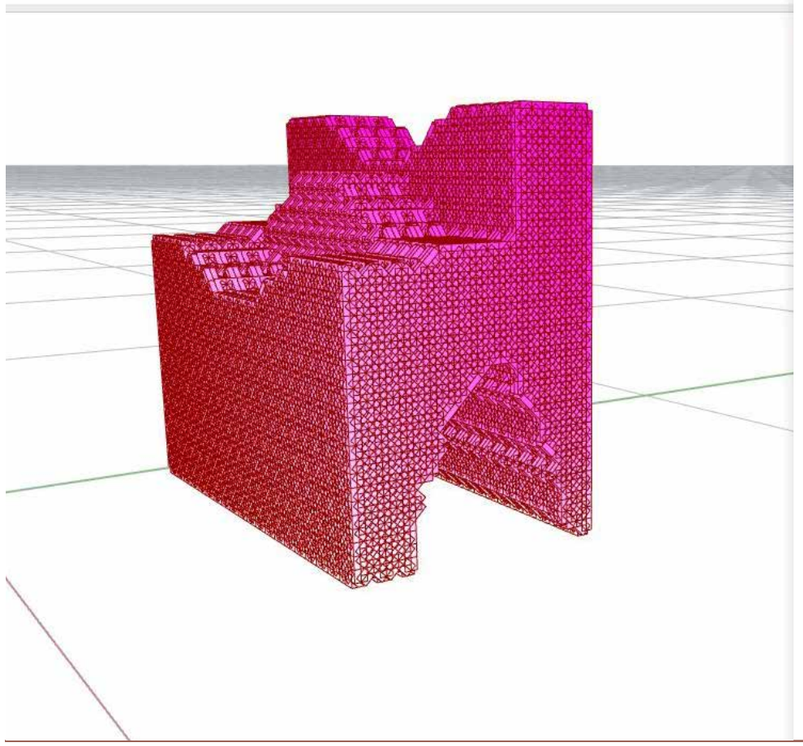
wall with door

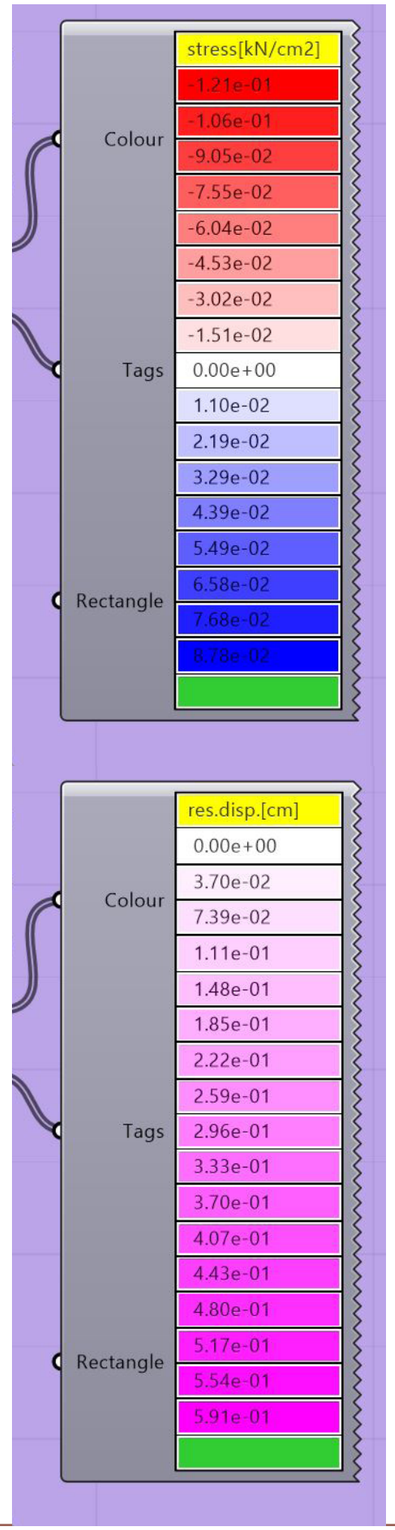




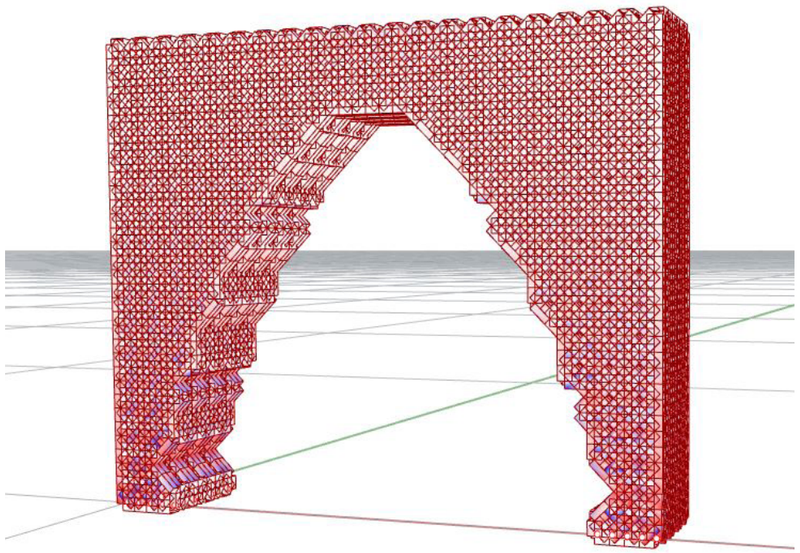
Appendix F: FEM calculations stairs walkway under the



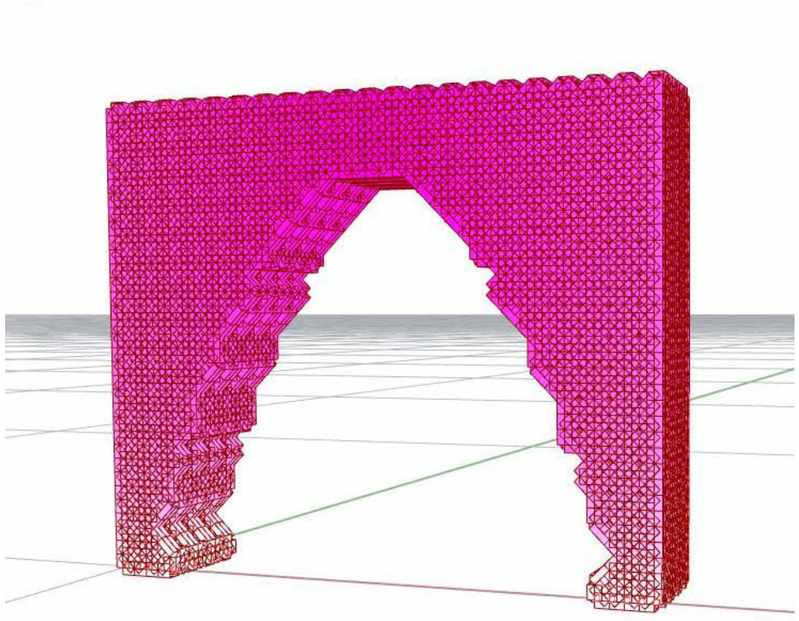




bridge over the stairwell



x



x