

Capstone Project Phase B

**House Gan ++**

**Project number -23-2-R-14**

Prof. Zeev Volkovich

Dr. Renata Avros

Dorin Beery - dorinbeery@gmail.com  
Mosa Hadish - moseshadish@gmail.com

# 

# Table of Contents

[Table of Contents 2](#_3yytzu4jed1k)

[1. Abstract 4](#_7upwltcsz4g0)

[1.1 Introduction 4](#_155uamm7ueic)

[1.2 Background and Related Work 5](#_bs6a0uoe9tsy)

[1.2.1 Neural Network 5](#_o8qnsepzurvl)

[1.2.2 Convolutional Neural Network(CNN) 5](#_bya0phydp634)

[1.2.3 Generative Adversarial Network (GAN) 6](#_ac1iro8lt99m)

[1.2.4 Conditional GAN (CGAN) 7](#_2m333cdqr0y)

[1.2.5 Relational GAN (RelGAN) 8](#_16mdwvdni3gn)

[2 Project Review and Assessment 9](#_33eyuiopa7ku)

[2.1 Project Accomplishment Description 9](#_96iacrolp5kz)

[2.2 Architecture 10](#_fdnxo5dp2jgs)

[2.2.1 CI/CD 10](#_r9axa456fqrj)

[2.2.1.1 Git 10](#_i7lzjwpcj1m6)

[2.2.1.2 Google Cloud Build 10](#_wmmc2979ajqq)

[2.2.1.3 Google Artifact Registry 10](#_sgv4lpxsxijr)

[2.2.2 Google Cloud Storage 10](#_hr6sl73oo5xd)

[2.2.3 Google Cloud Run 10](#_mc70ggr98fr7)

[2.2.4 Agile Development 10](#_xy9u7xj4p8u5)

[2.2.5 GUI and Logic Separation 11](#_2zdz8cf89229)

[3 Challenges and Solutions 11](#_lgzhxxteblg)

[3.1 Experimental Study 11](#_wrbgtwzfjv1w)

[3.2 Selecting Development Tools 11](#_upuxvjnnb0k4)

[3.3 Json bugs 11](#_i7x0utpvrnbh)

[3.4 Running Model With Google Collab 11](#_dei6pcf708ux)

[3.5 Migrating to Google Cloud from Google Collab 12](#_m7xtkvghcm77)

[3.5.1 Google Function 12](#_ywfw2xozdmvu)

[3.5.2 Cloud Run 12](#_o1a77dited20)

[4 Discarded ideas 12](#_3hzlhxljfpbu)

[4.1 CUDA 12](#_3ilpxzp72q0w)

[4.1.1 Google Cloud Function 12](#_vjskf0d72vbh)

[4.1.2 Google Cloud Run 13](#_ohhm7yyrtzk)

[4.1.3 Google VM 13](#_qxe0bdy5mwur)

[4.1.4 Google Kubernetes Engine (GKE) 13](#_a1egste0vtms)

[4.2 Edge Selection in GUI 13](#_efczzzoeiduz)

[5 Diagrams 13](#_a2htdkmbnykr)

[5.1 Generate Floorplan Activity Diagram 14](#_pw9bjeswjeqj)

[5.2 Package Diagram 14](#_amjk4sclyxp6)

[6. Testing Process 15](#_xemm3id1w17i)

[6.1 Model Testing 15](#_xzsms4smds5a)

[6.2 GUI Testing 19](#_qi0vh1tfqgon)

[7 User Documentation 25](#_hao4nnrzvb2d)

[7.1 User Guide 25](#_eci8oewy4gjd)

[7.1.1 Start Screen 25](#_gv7drt3akl6x)

[7.1.2 New Floorplan Screen 26](#_1e5dj8k1rakx)

[7.1.3 Load Json button 26](#_j5gs3ht44m73)

[7.1.3 Add a Door 27](#_n3pcz96s9qbh)

[7.1.3.1 Select Door Type 28](#_famsd1j9ave6)

[7.1.3.2 Fill Room Neighbor Indexes 28](#_s8w5eqv6ov4s)

[7.1.3.3 Fill in Room Neighbor Types 29](#_kfhgt5g8csk1)

[7.1.3.4 Add Horizontal Door or Vertical Door 30](#_sqbvylz7b5os)

[7.1.4 Add a Room 30](#_u2s1j9lrh1t6)

[7.1.4.1 Add a Room - Select Room Type 31](#_xk7cnojsroj4)

[7.1.4.2 Add a Room - Fill Neighbour Room Indexes 31](#_q58uzkzepwhz)

[7.1.4.3. Add a Room - Fill Neighbor Room Types 32](#_fokmbolu0cqa)

[7.1.4.4 Add a Room - Fill Neighbour Door Indexes 33](#_o9zvwsl5qy7a)

[7.1.4.5. Add a Room - Fill Neighbor Door Types 34](#_952cco9nxb)

[7.1.4.6. Add a Room - Click on “Add Room” button 34](#_k54pbe3o6963)

[7.1.5 Moving Rooms, Doors and Edges around the Canvas 35](#_q6mjopl6unqm)

[7.1.5.1 Clicking “Both Selection” 35](#_gkblej4i32y1)

[7.1.5.1.1 Moving with Cursor 36](#_ngr1jz3wdg9d)

[7.1.5.1.2 Moving with Keyboard 36](#_o8f1a672w7h7)

[7.1.5.2 Clicking “Edge Selection” 36](#_qr8rce3x8im6)

[7.1.5.3 Clicking “Save new Json” button 37](#_e6ti3gu5dk9q)

[7.1.5.4 Clicking “Generate” button 37](#_c7ghoxbgvxbr)

[7.2 Maintenance Guide 38](#_fdrx1zv0196s)

[7.2.1 Hardware Requirements 38](#_a6lmge9tw7qj)

[7.2.2 Software Requirements 38](#_9e576ze56g7z)

[7.2.3 Installation 38](#_oe2ju7mrg9iv)

[7.2.4 Maintenance 38](#_s0lz5uu119e3)

[8 References 38](#_4dr982g11qie)

# 

# 1. Abstract

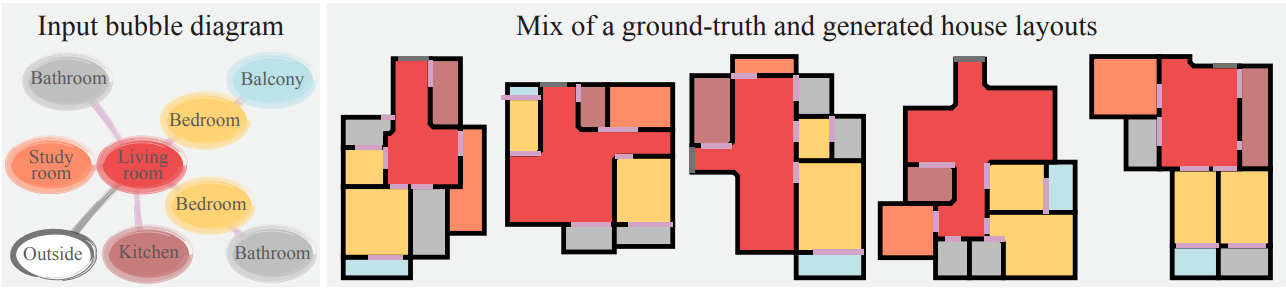
The main purpose of this project is to present a way for generating professional floor plans for houses. It extends and refines the House-GAN [[1]](#_vd6aueajbzmm).

The discovery is a simple non-iterative training process, named component-wise GT-conditioning, that is effective in the learning curve of the generator.

Also, it provides further improvement by meta-optimization techniques.

Assessment is based on metrics used to evaluate, show significant improvements over the original House-GAN program, and even rises above ground-truth floorplans.

## 1.1 Introduction



[Figure 1] A floor plan is a diagram of a house viewed from above. They are a vital part of the house building process as they visualize the house without the need to visit it physically. They help to show the relationships between rooms and spaces and give a feeling of the flow of the house.[[4]](#_unqjsovtvhnx)

The issue being addressed is the creation of house floor plans, a time-consuming process, mostly limited to professional architects which results in huge costs.

The goal is to generate professional floor plans automatically, based on the client’s conditions, which will save a lot of time and money.

This project is based on the state of the art House-GAN, the novelty suggested here allows the model to cope with non-rectangular rooms, and achieve better results from previous models. It is possible due to improved architecture. The architecture includes training the GAN with component-wise ground truth-conditioning, where instead of providing the GAN with a professional ground-truth floor plan each iteration, a floor plan is provided with a probability of 0.5 at each iteration. And also this project suggests meta-optimization refinement-scheme which further improves the results compared to models.

The architecture is a mix of graph-constrained relational GAN and conditional GAN.

The term GAN [[103]](#_adzgyyo0tq2d), refers to a machine learning framework. Given a training set of ground truth floor plans designed by professionals, it learns to generate floor plans with the same properties.

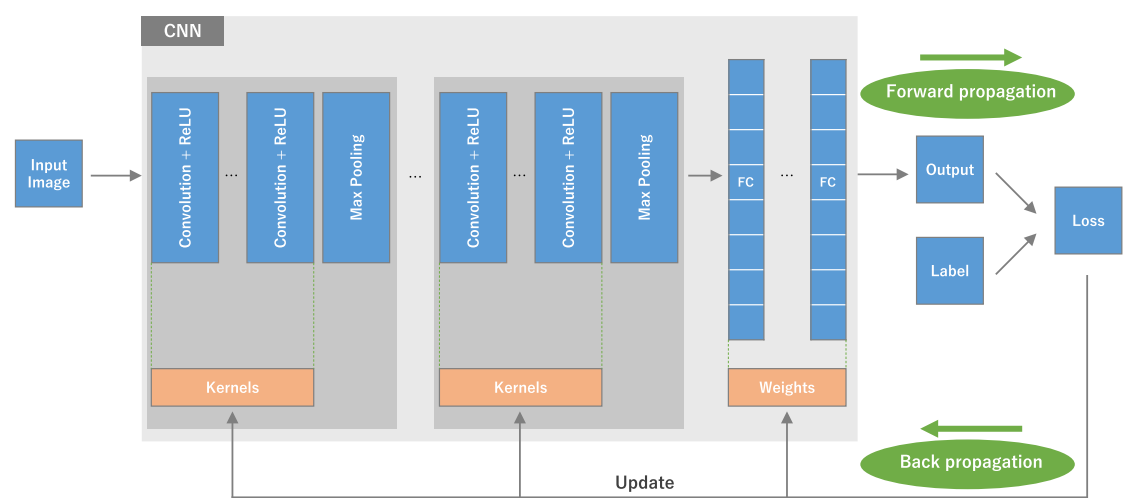
## 1.2 Background and Related Work

Many methods for generating structured data have been seen in the field. Our work revolves around deep learning, convolutional networks and image processing.

### 1.2.1 Neural Network

Neural networks are the backbone of all modern Artificial Intelligence (AI) systems. Their mechanism is similar to that of the human brain thus the name “neural”. The network is a massive amount of components named Neurons which are connected to each other and such that they form a network. The neurons receive input data, perform mathematical operation on it and forward it to the next layer of neurons through an activation function that determines which of the data is further forwarded and which not.

### 1.2.2 Convolutional Neural Network(CNN)

Is a type of deep learning model mainly for computer vision tasks. Their ability is to learn local stationary structures to generate hierarchical patterns without considering the spatial locations or size and it is their advantage against regular neural networks. 

[Figure 2] Illustrates training process of CNN. Input image is a matrix called a tensor and it undergoes linear kernel operation which is a smaller array of numbers. An element wise product between the kernel and tensor and then summation produces an output tensor called feature map. The process repeats on the whole input image to create feature maps each representing a different property of the input image. These are passed to a non linear activation function to simplify the amount of calculations by the model, then, a pooling layer is obtained on these in order to downsample the model. The generated feature maps of the convolutions are joined to a fully connected layer. The last fully connected layer undergoes a last activation function to provide the probability of classification to a class.

Lastly, the model performance is measured by the loss function through forward propagation, and the learned parameters are updated according to it through backpropagation [*[7]*](https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9)[*[9]*](https://proceedings.neurips.cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf).

### 1.2.3 Generative Adversarial Network (GAN)

A GAN consists of a generator and a discriminator, where each one of the mentioned is a neural network. This network is usually trained using adversarial training

The generator and the discriminator have 2 different roles:

The generator’s role is to generate samples that are indistinguishable from real data, using random noise vectors as input. A generator is typically composed of several layers (such as convolutional layers, etc)

The discriminator’s role is to output a probability score, indicating the likelihood of the input being real.

The generator is trained using real data, whereas the discriminator is trained using both real and fake data.

In order to learn a generator’s distribution pg over data x, a prior distribution is defined on input noise variables pz(z), which defines the characteristics of the input noise, which is then mapped to the data space as G(z;θg), where G is a differentiable function which is represented by a multilayer perceptron with parameters θg.

A second perceptron is defined, D(x;θd), which outputs a scalar. D(x) represents the probability that x is real data, not fake. The training goal is for D to maximize its probability of assigning the correct label to both sample types from G (real and fake), while G is simultaneously trained to minimize log(1-D(G(Z)).

This is usually referred to as a two-player minmax game with a value function V(G,D) (often referred to as objective function).

V(D,G) refers to the value function

G(x) refers to the data space generated from x

D(x) refers to the output representing that x came from training data

Ex∼Pdata(x) refers to Expectation function

The generator tries to minimize the value function, while the discriminator tries to maximize it.

In practice, the game is implemented using an iterative, numerical approach. The optimization of D to completion in the inner loop of training is prohibitive, and on finite datasets it would result in overfitting. The inner loop refers to the part of the algorithm that D is trained in to discriminate samples from data. In order to avoid that, the optimization of D and is alternated between k steps, and one step of optimizing G. As long as G changes slowly enough, D is maintained near its optimal solution.

After several steps of training, G and D will both reach a point at which they both cannot improve, meaning the generator outputs fake samples that are almost indistinguishable from real data, causing the discriminator to be unable to differentiate between fake and real data, i.e

The generator tries to deceive the discriminator, meaning that the generator passes on data, some of which is generated by the generator, while the rest is real data, to the discriminator, and the discriminator tries to differentiate between real and fake data. The network updates its parameters by using a process called Backpropagation.

Backpropagation is a process that enables the network to update its parameters for further optimization. There are two key steps in the backpropagation process:

Forward Pass - the process of feeding an input to the generator, as it travels each layer in the network, generating an output sample.

Backward Pass - the gradients of the loss function of the network’s output are computed, and propagated backwards through each layer, which are then used to update the parameters of the network using an optimization algorithm.

### 1.2.4 Conditional GAN (CGAN)

In this variant of a GAN, both the generator and discriminator receive additional input (denoted y) relevant to the task. This allows the user to influence the generated output by the conditioning of the GAN, passing as input the desired characteristics, and the generation is more controlled.

This results in more targeted and controlled sample generation

In HouseGAN++, the cGAN takes in as input the requirements of the layout, such as room types, number of rooms, etc..

However, cGANs don’t train easily for context prediction tasks as the discriminator easily exploits the generated regions and the original to easily classify fake versus real samples.

In the generator (denoted as G), the random input noise vector (denoted as z) and y are combined in a joint hidden representation, and the framework allows for flexibility in how this representation is composed. In HouseGAN++, z and y are concatenated. In the discriminator (denoted as D), the data and y are presented as inputs.

The value function of a cGAN is as follows:

minG maxD V(D, G) = Ex∼pdata(x) [log D(x|y)] + Ez∼Pz(z) [log(1 − D(G(z|y)))]

## 

[Figure 3] Illustrates the flow process of a CGAN, 2 inputs; conditions, random noise vector for the discriminator and generator. Concatenate the 2 inputs and pass it through the network

Batch normalization is a training technique that allows deep neural networks to standardize the inputs to a layer for each mini-batch. This stabilizes the learning process and drastically reduces the number of training epochs required to train the networks, which speeds up the training process.

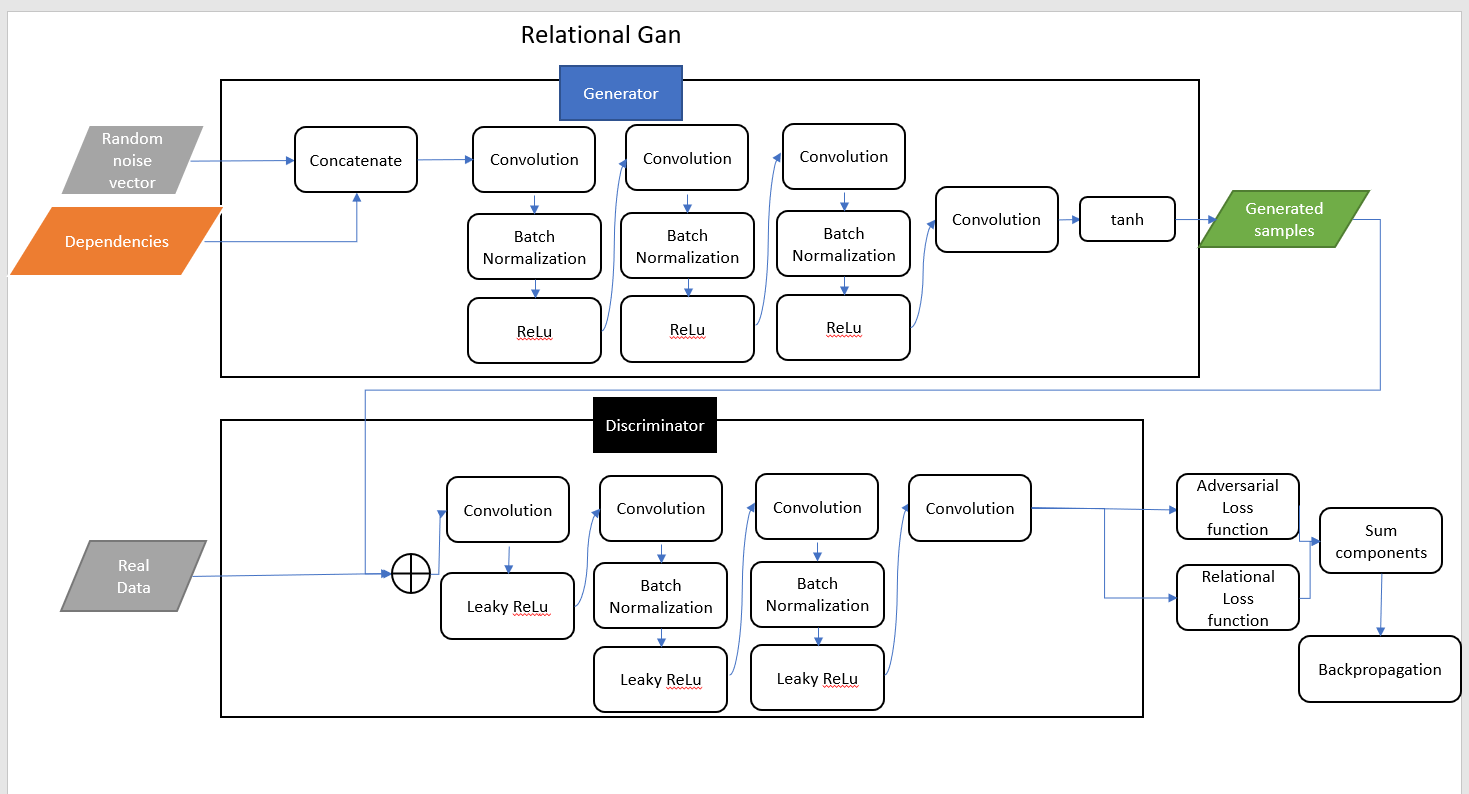
This is done by adding new extra layers. Those new layers perform the standardizing and normalizing operations on the input coming from a previous layer.

### 1.2.5 Relational GAN (RelGAN)

The idea behind a Relational Gan is to enhance the generation process by considering the dependencies and interactions between elements of the generated output. The generator has an additional input, which refers to the relational information or spatial dependencies, to guide the generation process. This can be useful in various tasks such as generating images in specific spatial arrangements, or generating data with complex dependencies and more.

This enables a Relational GAN to output more complex and realistic samples compared to standard GANs.

In HouseGAN++, the RelGAN’s generator takes as input the relationships between different rooms in the layout, doors, etc..



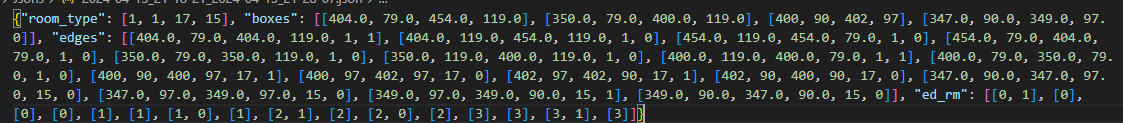
[Figure 4] Illustrates the flow process of a RelGAN, 2 inputs for the generator; conditions, random noise vector. Concatenate the 2 inputs and pass it through the network. Discriminator’s input is the same as normal GAN.

# 2 Project Review and Assessment

## 2.1 Project Accomplishment Description

The primary aim of our project was to test the models’ weakness and post train it, to deliver a better model. We already found weaknesses with the model's ability to recognize nested rooms as it does not make the difference. To achieve the post training we should have at least 1000 json files.

To start with explaining why achieving 1000 jsons files is hard we’ll first provide with a picture of a json depicting two rooms and two doors:

The place of each room/door corresponds to each key, the keys:   
“**boxes**” are pixels on screen for each room/door.  
“**edges**” are the edges of each box corresponding to the room\_type list, normally each box will have 4 edges but also none rectangle rooms are included so there are possibly 6 edges. The 5th and 6th indexes are the room\_types of that specific edge and its neighbor with that edge in the order: [left,top,right,bottom].  
“**ed\_rm**” are describing the neighbors of each room and door index, also corresponding to the room\_type list.   
To achieve a good model, each of the json files should be unique and different, with combinations of different room types, different rooms layout, different doors layout, and each room/box.

So we started building a simple GUI which allows such editing, this escalated quickly and our main goal changed to developing a GUI application which is able to edit a floorplan’s json, save it, and thus, create a dataset containing correct floorplans jsons.

## 2.2 Architecture

### 2.2.1 CI/CD

#### 2.2.1.1 Git

First step for CI/CD is choosing a version control system. It helps track changes to files, who made the changes and when, and revert changes. It also allows for branching and merging which makes collaborating a lot easier, distributing tasks and very appropriate for AGILE development. We chose git as both of us already know it and it is well documented and also integrated with google cloud.

#### 2.2.1.2 Google Cloud Build

Google Cloud Build is a service allowing CI/CD. When integrating Git repository to Cloud Build, Cloud Build responds to push requests from git by compiling the code, wrapping it as a container, saving the image in “Artifact Registry” and deploying it to Cloud Run.

#### 2.2.1.3 Google Artifact Registry

Google Artifact Registry allows to store, manage and secure container images in private or public repositories.

### 2.2.2 Google Cloud Storage

Google Cloud Storage is a scalable and flexible service for storing and accessing data in a secure manner, from anywhere in the world. We chose to store the pretrained model on google cloud because it is cheap, secure and collaborates very well with other Google Cloud services such Cloud Run.

Google Cloud Storage offers different “storage class” allowing to customize and pick the best package in means of pricing for your use like “Nearline” which is a class for accessing the data once a month. We chose the “Standard” class which is best for short-term storage and frequently accessed data.

### 2.2.3 Google Cloud Run

Google Cloud Run is a service allowing to invoke a container by web requests. It is scalable and serverless, very easy to bring up and maintain. We chose Google Cloud Run because it already abstracts away the server bring up allowing us to focus on the application and communication between it and the model.

### 2.2.4 Agile Development

Agile Development improves the application delivery time. It is an iterative and incremental approach allowing to focus on tasks. Combining between design and production leading to balanced application between achieving on-time product and well-done design.

### 2.2.5 GUI and Logic Separation

Best practice approach when developing desktop applications is to maintain separation between logic and ui, we’ve paid attention to it from the first line we wrote. it leads to cleaner and maintainable code allowing ease on the work for the long-run.

# 3 Challenges and Solutions

## 3.1 Experimental Study

To summarize, the first thing we wanted to achieve is an experimental study, so we needed to have a light version that works on the current hardware. That’s why our project transformed into a development project.  
First we tried to edit the jsons without GUI, this took a lot of time and it was not easy to comprehend whether the model failed because the json failed or whether it failed because it is not properly trained. The jsons' structure is not human-friendly from the first few rooms being added. Because of this situation and knowing that post training a model to get a better version of it requires at least 1000 different jsons, The solution was to build a GUI that serves as a utility to generate dataset of different variety of floorplans.

## 3.2 Selecting Development Tools

We had to pick the correct language to work with so we chose python as it is most fast to develop with and has a lot of user-libraries implementing most of the needed features in our GUI. We chose tkinter because it is well-tested, has thorough documentation and users.

## 3.3 Json bugs

Later on difficulties continued to stack up trying to follow each of the rules that the json was following, the AGILE process was intensive as bugs in our comprehension of the json structure were floating. GUI presents correct floorplan structure but the model outputs wrong data even for simple occasions. Testing were made and deeper research on current jsons were conducted. Eventually we’ve made it, dealing with all the connections and dependencies of the “edges” and “ed\_rms” and the model now corresponds to the output of the GUI. Users can use the GUI and create their own dataset without even downloading the pretrained model because it is already on the cloud.

## 3.4 Running Model With Google Collab

The generation of a floorplan was conducted of these steps:

1. Generate floorplan json with developed GUI.
2. Connect Google Collab to Google Drive
3. Upload .txt and .json file to Google Collab platform.
4. Initialize the Google Collab environment holding the code for testing the model which took several minutes
5. Opening the generated floor plan and graph on the ./dump folder.
6. After some time, we exceeded the allowed amount of usage of google collab GPU and CPU and couldn't continue without opening a new google account.

This process was clumsy so the solution was to make a connection from the GUI to Google Collab, soon we realized there is no API for google collab and we had to migrate to a different cloud platform.

## 3.5 Migrating to Google Cloud from Google Collab

This devops section is new to us and aroused many problems. Luckily we ran the model locally with GPU hardware so we had a good local start, also, we already prepared a notebook that’s running on Google Collab which is another advantage.

### 3.5.1 Google Function

At first we tried using Google Function, this is a service that responds to a trigger. The thought was to save a generated json file on Google Bucket, this will trigger the function (running the model on the current saved json) and the result will be sent back to frontend, but something went wrong and the server wasn’t responding as expected.

### 3.5.2 Cloud Run

So the next step was to try Cloud Run which is a stateless container that also serves as a pub/sub to a certain url and port. This required a container, so the next step was to establish a requirements file. We gathered all the packages installed via pip install on the computer and dereferenced them, meaning, we left only those that are used by the project. Trying to deploy these packages brought up new dependency issues with installing LibGL and Graphitz libraries, which are computer programs and not python packages, so we've edited the Dockerfile to include installation of these on the machine the container runs on. Deploying this package included many tries and fails due to code compatibility so we did minor changes so the code of the model will also work on linux environment, and finally the deployment completed successfully. We took the generated URL and sent our first post request which responded very fast that no GPU hardware was available on the machine the container was running on.

# 4 Discarded ideas

## 4.1 CUDA

We managed to run the model locally on my computer as it has the required infrastructure - NVIDIA GPU. Following the AGILE development we desired to make use of google cloud to allow all the computations to be on cloud but failed.

### 4.1.1 Google Cloud Function

First, we tried google function because it enables just to be given with the model and than output result, thought of combining it with triggers from google-storage to trigger an event for cloud function to take that json and feed to the model but came to understanding the that Google Cloud Run was more suitable for the requirements as it is a serverless architecture could make the developing process easier.

### 4.1.2 Google Cloud Run

After some struggle with adjusting the code of the model to the Google Cloud Run platform, Postman received a response from Google Cloud Run that there are lines of code requiring the use of ‘CUDA’ but no ‘CUDA’ is found on the machine. This is where we decided to try Google VM.

### 4.1.3 Google Virtual Machine (Google VM)

We became suspicious to the given GPU problem so we checked before-head whether Google Virtual Machine is able to have a GPU, it sounds obvious but soon realized that the 300$ free of use are not applicable for GPU use on Google VM. Bringing up the GPU infrastructure is for 312$ per month, excluding the premium account fees.

### 4.1.4 Google Kubernetes Engine (GKE)

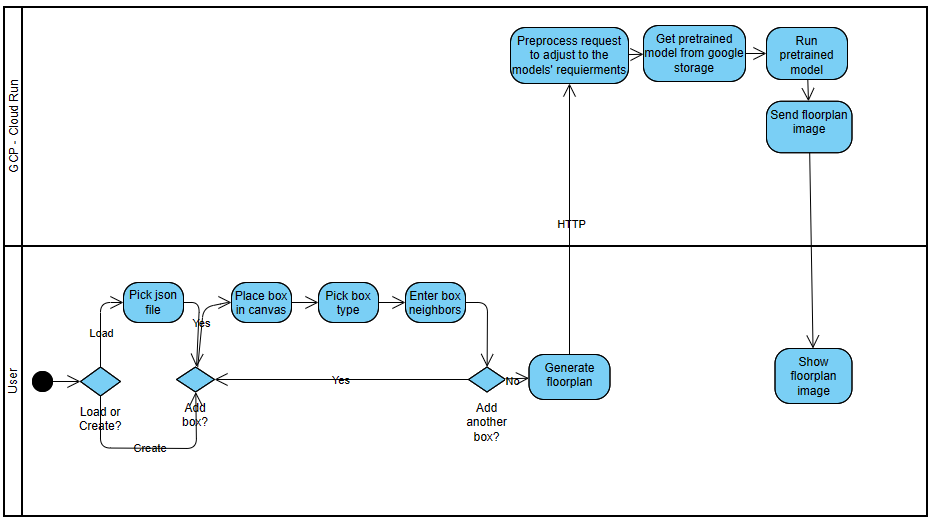
Last resort was using the GKE. We have tried the fast approach called “AutoPilot” which is quite straight-foward but after the build completed we looked for the GPU option but on the “AutoPilot” mechanism it is not possible. There is also an option to bring GKE without “AutoPilot” but it consumes more time, so we decided to check if the model can run on CPU only before diving into another devops task. Indeed we were capable of adjusting the code to work only with CPU, without the use of CUDA, so discarded the cuda option and kept with “Google Cloud Run” service.

## 4.2 Edge Selection in GUI

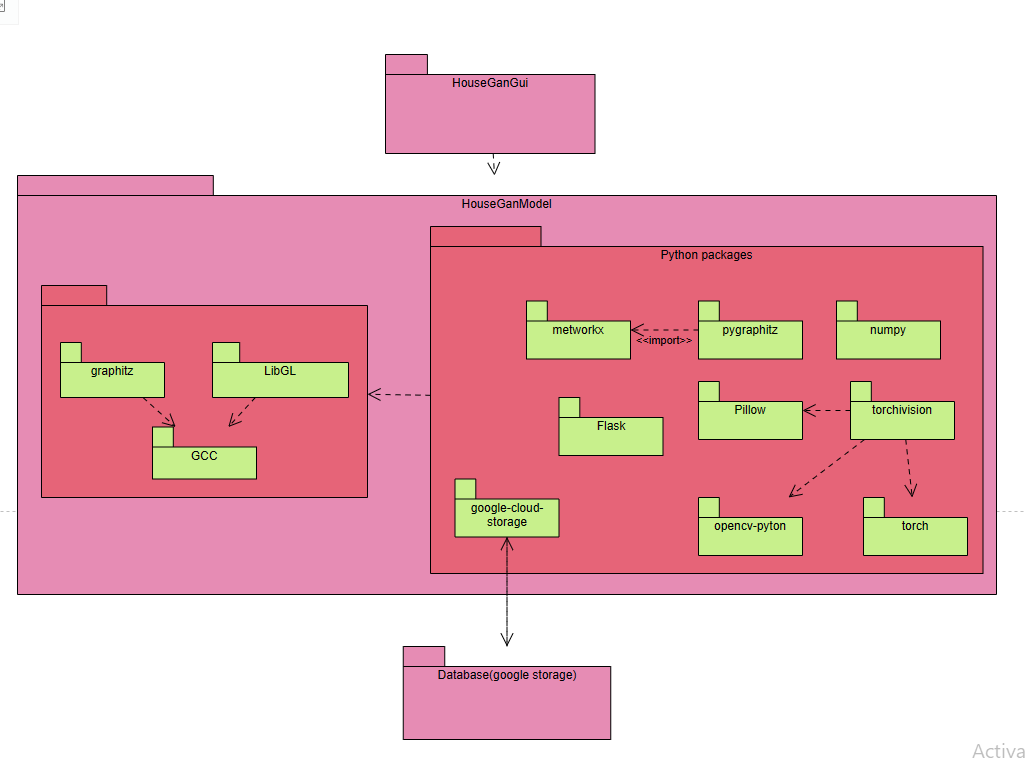
In the GUI we’ve allowed edge selection, but it isn’t precise enough and there are some modifications that need to be added in order for the model to be capable of getting it as input. The problem is that the edges are most of the time not “overlapping”. Meaning, if one edge x2 is at 104.0, then the other edge x1 needs to start exactly from 104.0 and this isn’t archived well while the user drags the edge around the canvas. We’ve tried to modify it and force the backend to fix non overlapping edges but it requires more engineering time.

# 5 Diagrams

### 5.1 Generate Floorplan Activity Diagram

[Figure 5] Generating Floorplan Activity Diagram

### 5.2 Package Diagram



# 6. Testing Process

### 6.1 Model Testing

**Test how the model handles different incomplete JSONs**

| Test #1 | Extra Box | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add an Extra box in the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Added 1 extra room | | | | |
| assets: |  | |  | | |

| Test #2 | Extra Edge | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add an Extra edge in the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Combined rooms even tho no room type was given | | | | |
| assets: |  | |  | | |

| Test #3 | Extra EDRM | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add an Extra EDRM in the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Error message, as expected | | | | |
| assets: |  | |  | | |

| Test #4 | Extra Room Type | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add an Extra Room Type in the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Error message, as expected | | | | |
| assets: |  | |  | | |

| Test #5 | Missing Box | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Remove 1 box in the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Output image is missing 2 rooms (instead of 1?) | | | | |
| assets: |  | |  | | |

| Test #6 | Missing Edge | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Remove 1 edge from the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Error message as expected, we can tell from it that it’s related to edges | | | | |
| assets: |  | |  | | |

| Test #7 | Missing EDRM | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Remove 1 EDRM from the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Output image with 5 rooms (input has 6 different room types) | | | | |
| assets: |  | |  | | |

| Test #8 | Missing Room Type | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Remove 1 Room Type from the dictionary in the JSON | | | | |
| Expected Result (in words): | Some sort of error message or invalid graph | | | | |
| Result (in words): | Front Door turned into a room? | | | | |
| assets: |  | |  | | |

| Test #9 | Box with Negative size | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Change 1 box size from the dictionary in the JSON to negative size | | | | |
| Expected Result (in words): | Some sort of error message, warning or invalid graph | | | | |
| Result (in words): | A Front Door was added, 1 room removed (is it maybe the new door) | | | | |
| assets: |  | |  | | |

| Test #10 | Edge with negative values | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Changed an edge so that its values are negative (length is negative) | | | | |
| Expected Result (in words): | Error message or a warning | | | | |
| Result (in words): | A graph with 4 rooms (input had 6 rooms), front door missing | | | | |
| assets: |  | |  | | |

| Test #11 | EDRM with negative values | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Changed EDRM values to negatives | | | | |
| Expected Result (in words): | Error message or a warning | | | | |
| Result (in words): | A graph with 5 rooms (input had 6 rooms), front door missing, and an empty space between rooms which was not present. | | | | |
| assets: |  | |  | | |

| Test #12 | Invalid room type | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Added an invalid room type to the JSON | | | | |
| Expected Result (in words): | The model has an “Unknown” room type, expected result was to add such a room | | | | |
| Result (in words): | A graph with 5 rooms (input had 6 rooms), front door enlarged and is now looking like a room (it’s possible to to see in the graph that it’s a front door, meaning it wasn’t added as a node) | | | | |
| assets: |  | |  | | |

| Test #13 | Box Size 0 | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Changed a room’s size to 0 | | | | |
| Expected Result (in words): | A relevant warning or error message | | | | |
| Result (in words): | As expected. | | | | |
| assets: |  | |  | | |

### 6.2 GUI Testing

Test the validity of the generated json files and basic buttons of the GUI.

| Test #1 | Save an empty json | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Click on clear and press save | | | | |
| Expected Results (in words): | 1. A json with empty keys is created. | | | | |
| Result (in words): | As expected. | | | | |
| Assets: |  | |  | | |

### 

| Test #2 | Add room to empty json | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Start from Clean json and add room, save new json. | | | | |
| Expected Results (in words): | 1. A single room is added to GUI. 2. Selected room type is added to the “room\_type” list in json. 3. Box coordinates are added to the “boxes” list in the json. 4. 4 elements are added to the “edges” list in json. 5. 5th position is the selected room type. 6. 6th position is 0 because the room has no neighbors. 7. 4 elements to the “ed\_rm” list in the json. 8. Each of the elements stands alone because there are no neighbors. 9. The element is 0 because it is the first room added. | | | | |
| Result (in words): | As expected. | | | | |
| assets: |  | | | | |

| Test #3 | Add door to an empty json | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Click on clear and then on add door, save new json. | | | | |
| Expected Results (in words): | 1. A single door is added to the canvas. 2. Selected door type is added to the json. 3. Box coordinates are added to the “boxes” list in the json. 4. 4 elements are added to the “edges” list in json. 5. 5th position is the selected door type. 6. 6th position is 0 because the door has no room neighbors. 7. 4 elements to the “ed\_rm” list in the json. 8. Each of the elements stands alone because there are no neighbors. 9. The element is 0 because it is the first that has been added to the canvas. | | | | |
| Result (in words): | As expected. | | | | |
| assets: |  | | | | |

| Test #4 | Add Room to canvas with single door | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add Room to canvas with single door | | | | |
| Expected Results (in words): | 1. Room type is concatenated to head of room\_type list 2. Coordinates of room are concatenated to head of boxes list 3. Coordinates of room’s edges are concatenated to the head of edges list 4. 5th position of the new elements in ‘edges’ list are ‘1’ as the selected room type (living room) 5. All 6th position of the new elements are 0 because it has no room neighbors (only one door) 6. The 6th position of the left door edge is changed to 1 because a room neighbor of type 1 is added to it’s left (to its left means to the door’s first edge, easily recognized with the 5th position being 15 instead of 1) 7. 4 elements concatenated to the head of ‘ed\_rm’ list each is 0 because the room added is now indexed 0 (instead of the door that is incremented to being index 1) 8. Each of the new elements is 0 because the new room has no room neighbors, only a door (which is disregarded). 9. The 4 edges of the door that were indexed 0 are now incremented to be of index 1 10. The first of those edges is updated to be with a neighbor, the new room. | | | | |
| Result (in words): | As expected. | | | | |
| assets: |  | | | | |

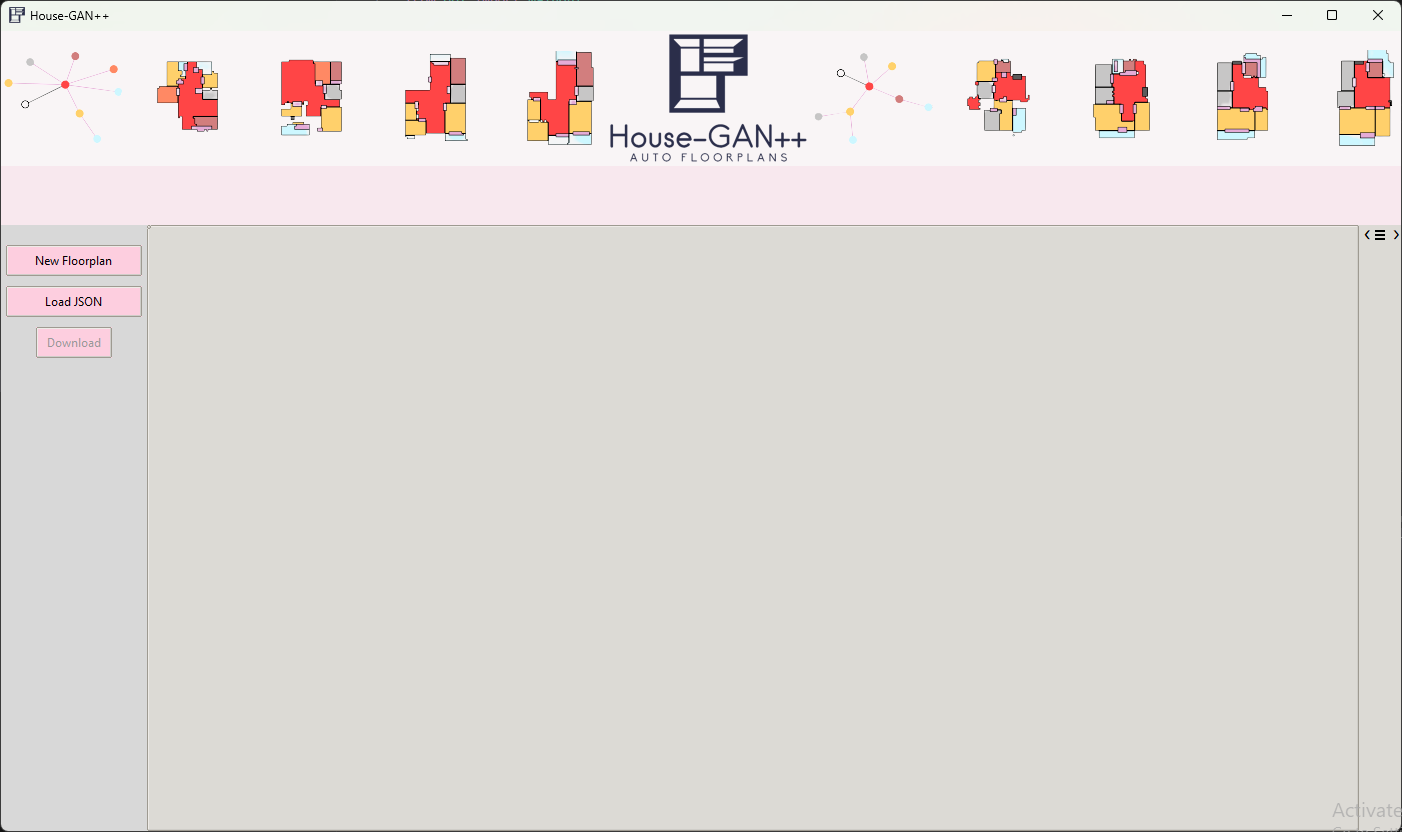
| Test #5 | Add door to canvas with single room | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Add door to canvas with single room | | | | |
| Expected Results (in words): | 1. Selected door type is added to the end of “room\_type” list. 2. Coordinates of door are concatenated to end of boxes list 3. Coordinates of door’s edges are concatenated to the end of edges list 4. 5th position of the new elements in ‘edges’ list are ‘17’ as the selected door type (interior room) 5. The 6th position of the first new edge is 1 because it has a room neighbor of type ‘1’ (living room) to it’s left. all other edges have 0 at this position because they have no neighbor. 6. The 6th position of the left room edge is unchanged because door neighbors are disregarded. 7. 4 elements concatenated to the end of ‘ed\_rm’ list each is 1 because the door added is now indexed 1 after the room which is indexed 0. 8. The first of the new element is indexed [1,0] because it has room 0 as its’ left neighbor. | | | | |
| Result (in words): | As expected. | | | | |
| assets: |  | | | | |

| Test #6 |  | | | | |
| --- | --- | --- | --- | --- | --- |
| Description: | Tkinter uses tags in order for the developer to be able to locate objects on canvas. Manipulating the json requires also manipulating the tags. e.g if the index of a door is changed than also the tags of that room must change. | | | | |
| Expected Results (in words): | To verify, first add a door, then add a room, and check if the tags of the doors are changed correspondingly. In other words, when a door is added than its’ index is 0, than a room is added but it needs to be the first in the list for the model to understand, so the room index will be 0 and doors’ index is updated to 1. | | | | |
| Result (in words): | as expected. | | | | |
| assets: | Green line represents the print of tags after a room is added. | | | | |

# 7 User Documentation

## 7.1 User Guide

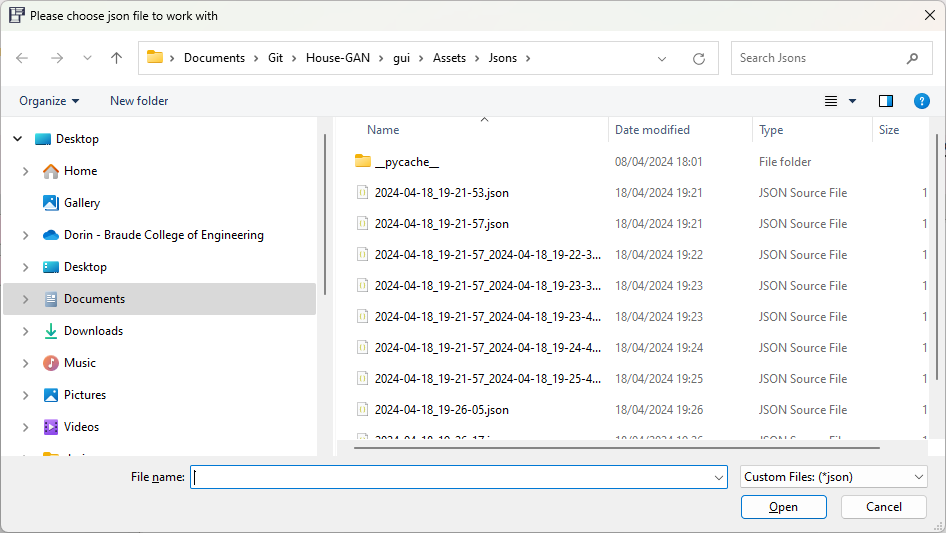
### 7.1.1 Start Screen

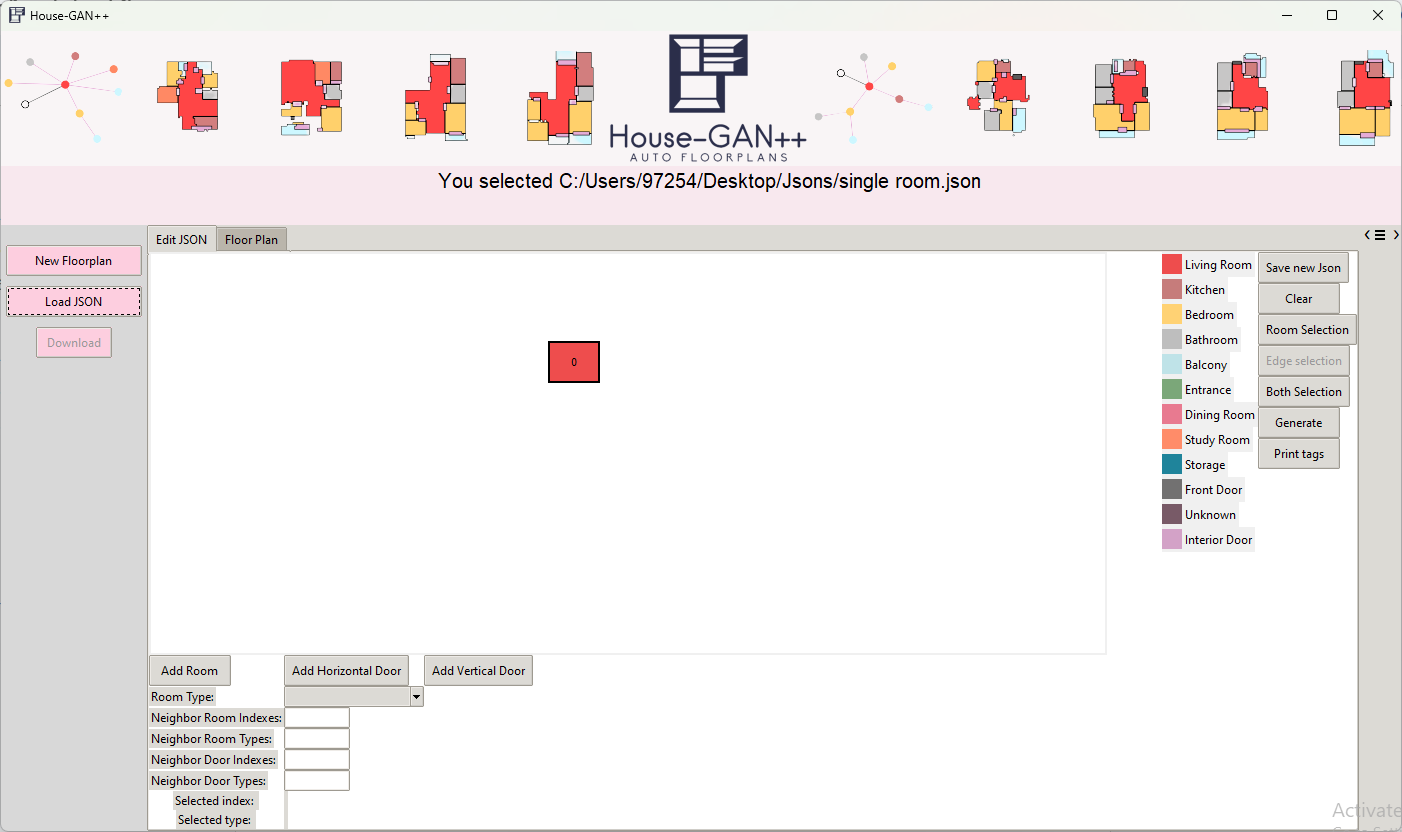
  
[Figure 6] When opening the application this screen is shown. User can choose between creating a new floor plan with the button “New Floorplan” or to load an existing json with the button “Load Json”.

### 7.1.2 New Floorplan Screen

[Figure 7] When clicking the “New Floorplan” button, this screen is opened. Here we can see the legend describing the rooms and doors types.

### 7.1.3 Load Json button



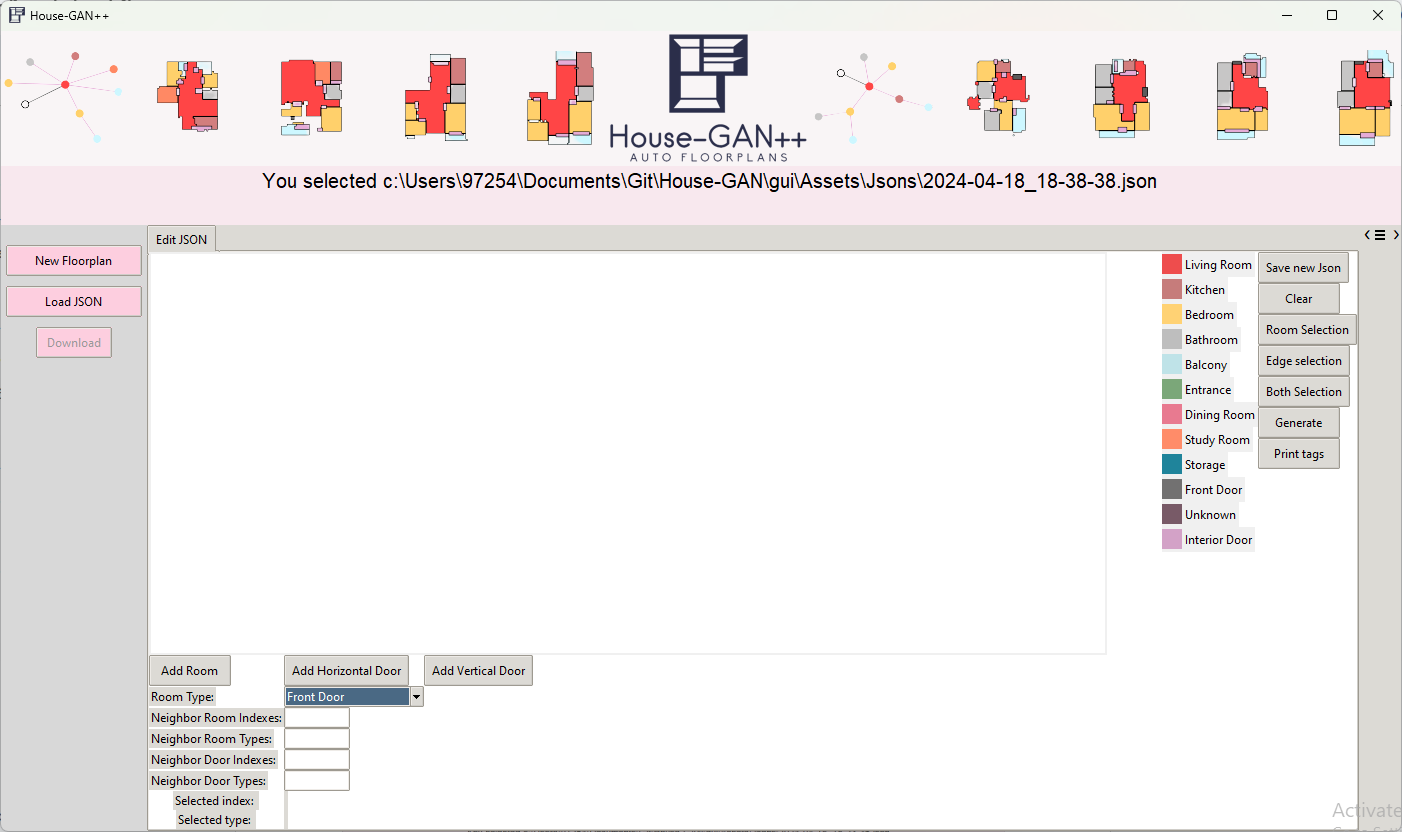
[Figure 8] When clicking the “Load Json” button a screen opens requesting to select the desired json file.  


[Figure 9] After clicking the desired json file, the json is drawn to the canvas.

### 7.1.3 Add a Door

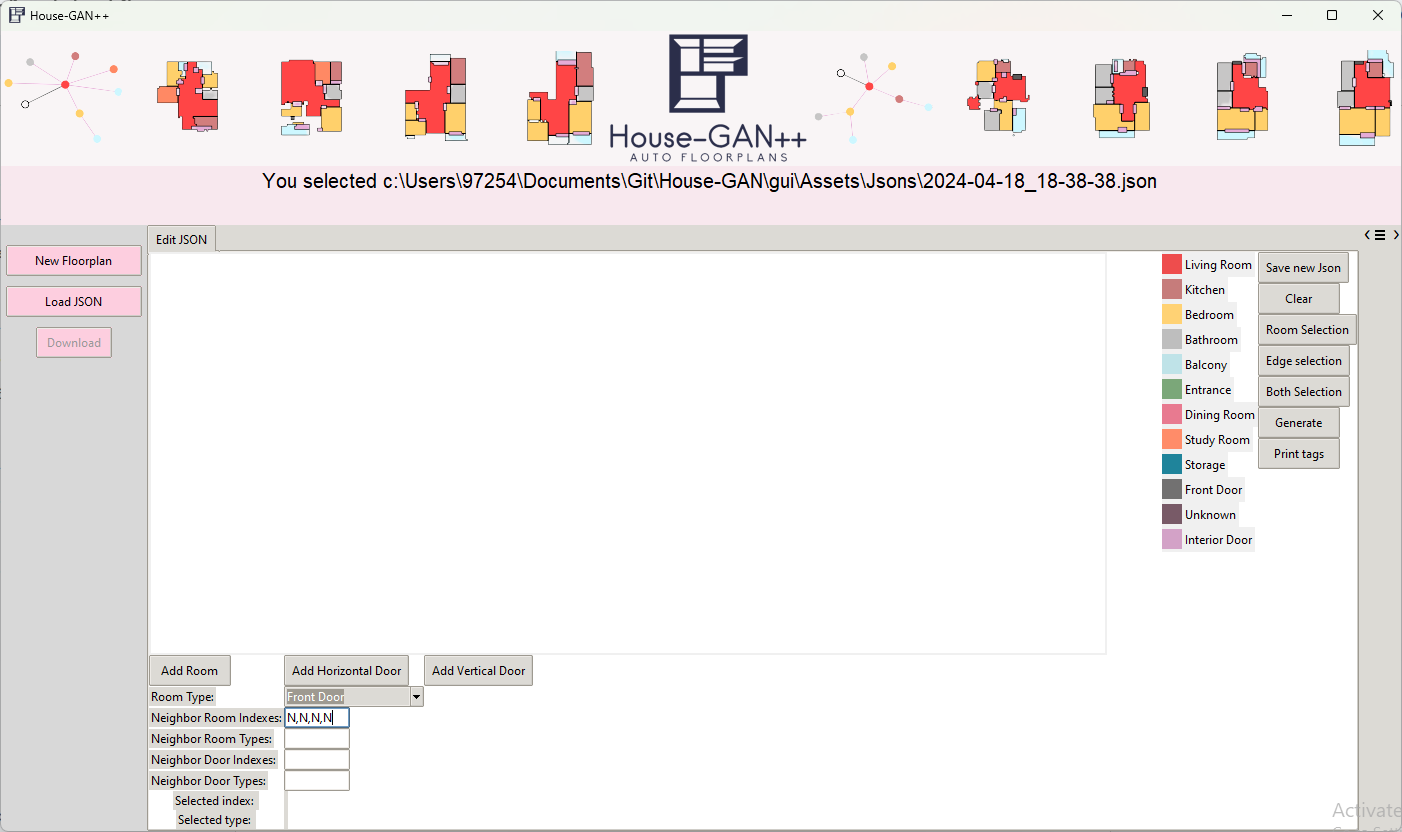
The model input is a json file describing the house layout. When adding a door, the model is expecting to know who are its neighbor rooms, hence, the user must fill the doors’ neighbor room indexes and types. In this section we will add a door to an empty canvas and then add a room to a canvas with a door.

#### 7.1.3.1 Select Door Type



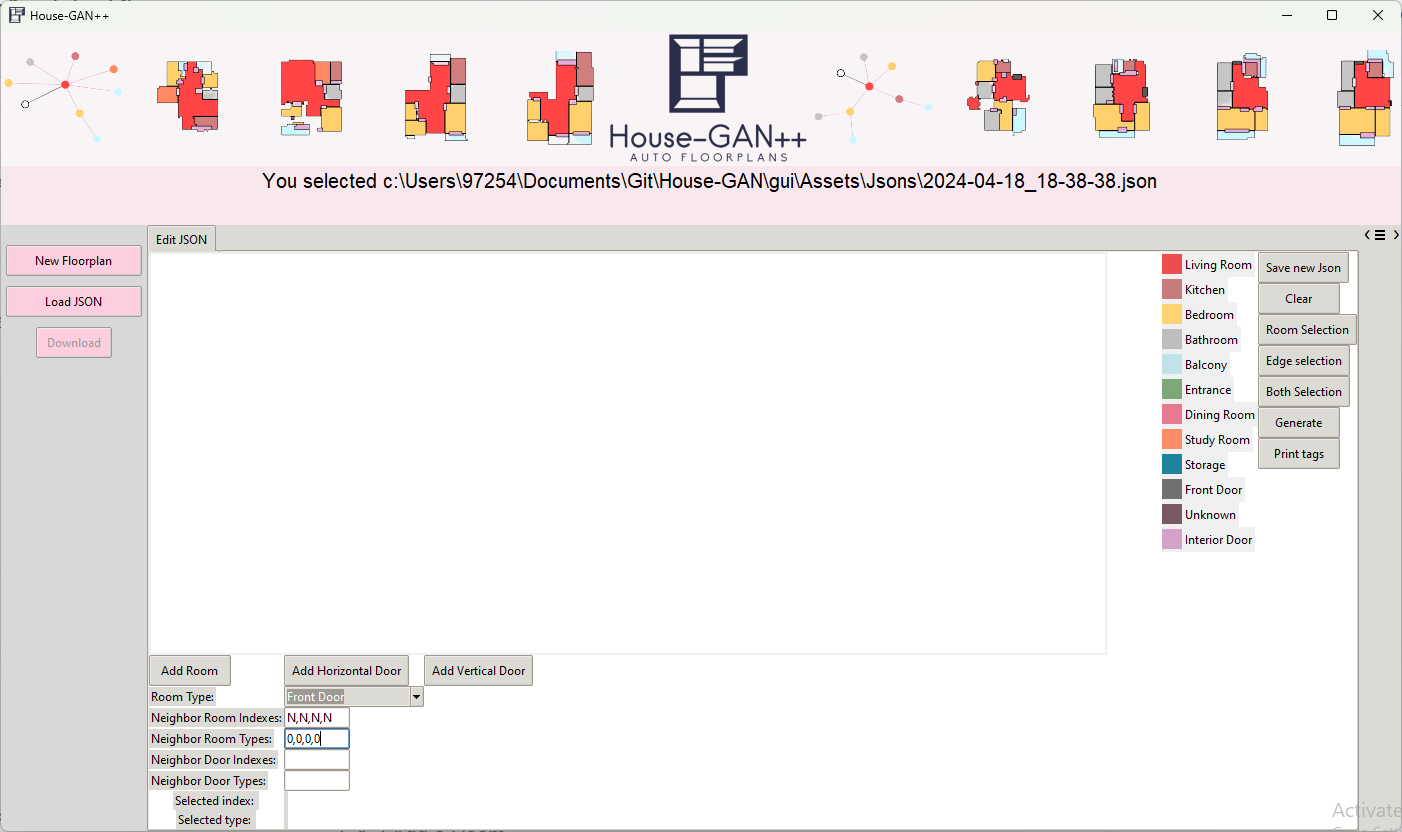
[Figure 10] Selecting a door type according to its job. Lets select “Front Door” in the example.

#### 7.1.3.2 Fill Room Neighbor Indexes

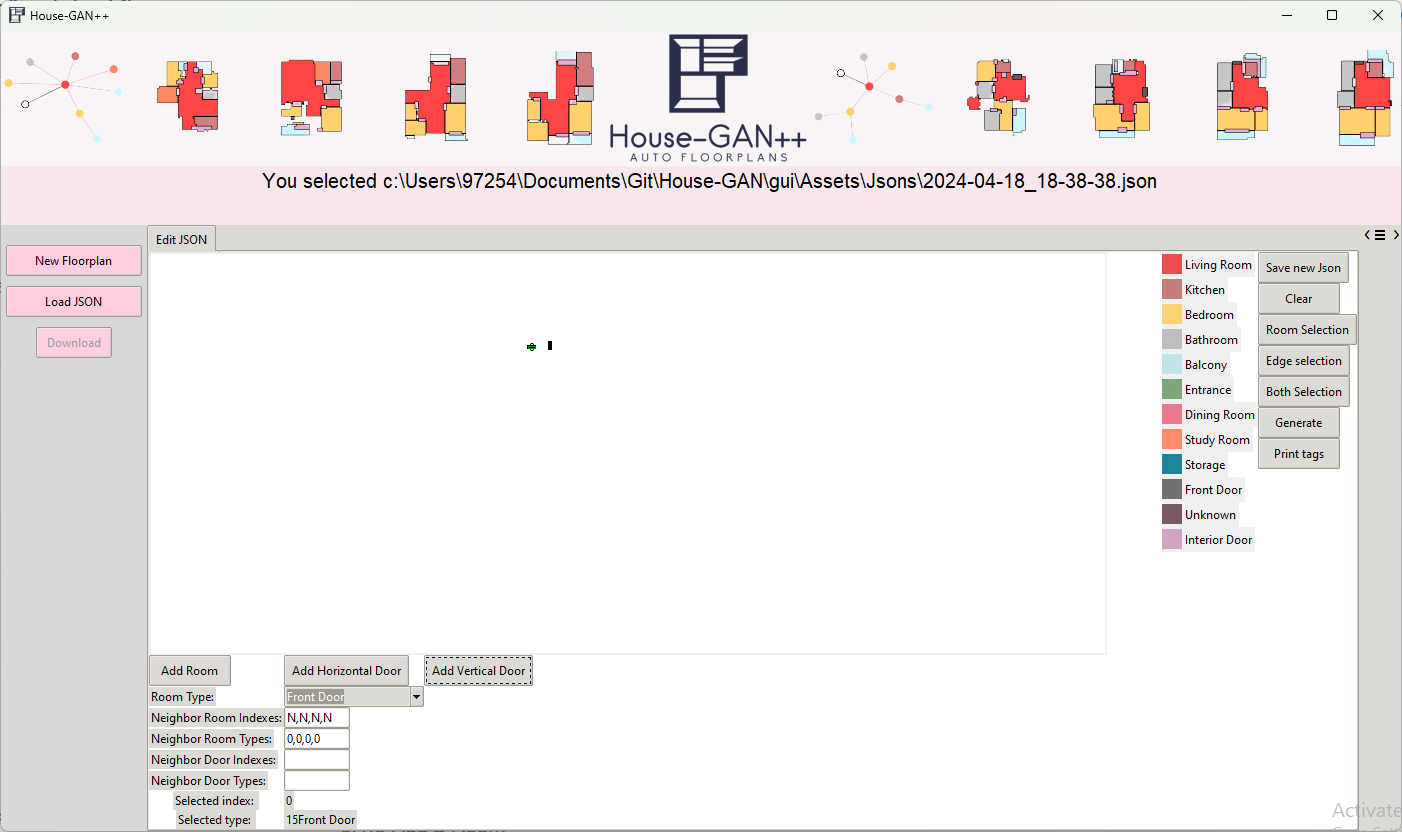


[Figure 11] The input to the model is a Json file, it describes who are the neighbors of each room, so the user must fill in the room neighbors indexes of the added door. In this example there are no neighbors so fill “N” in all four elements. from left to right the positions are left, top, right, bottom. An example of filling neighbor room types and neighbor room indexes when there are neighbors are following this section.

#### 7.1.3.3 Fill in Room Neighbor Types

[Figure 12] The input to the model is a Json file, it describes who are the neighbors of each door, so the user must fill in the room neighbors types of the added door. In this example there are no neighbors so fill “0” in all four elements. from left to right the positions are left, top, right, bottom. An example of filling neighbor room types and neighbor room indexes when there are neighbors are following this section. It is worth noticing that the models don't care about the neighbors' doors of an added door. So we can leave those empty.

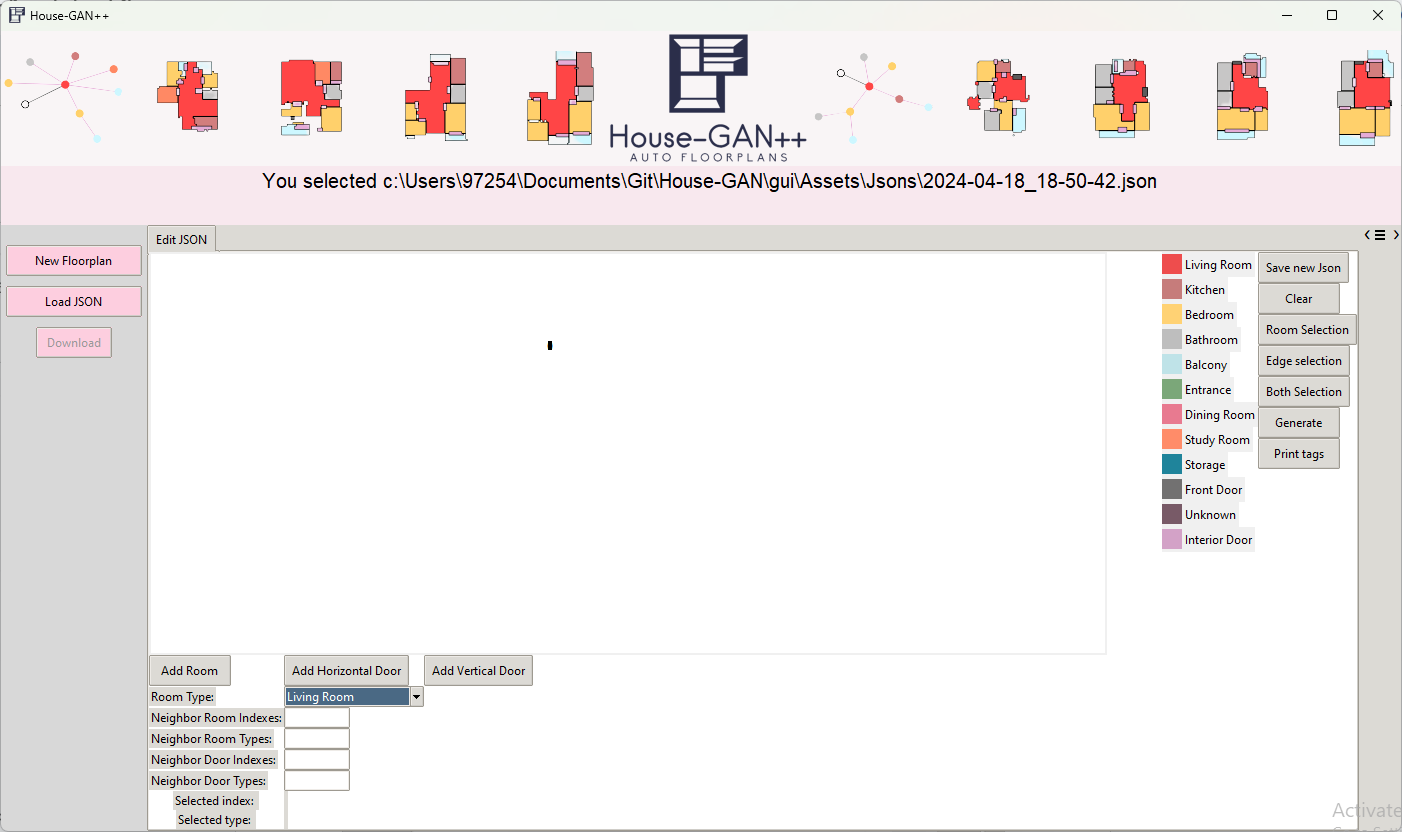
#### 7.1.3.4 Add Horizontal Door or Vertical Door

  
[Figure 13] User can choose between adding a horizontal door or a vertical door, its dimensions are according to what the model recognizes as a door valid door size. Testing has shown that bigger doors are not represented as doors in the output.

### 7.1.4 Add a Room

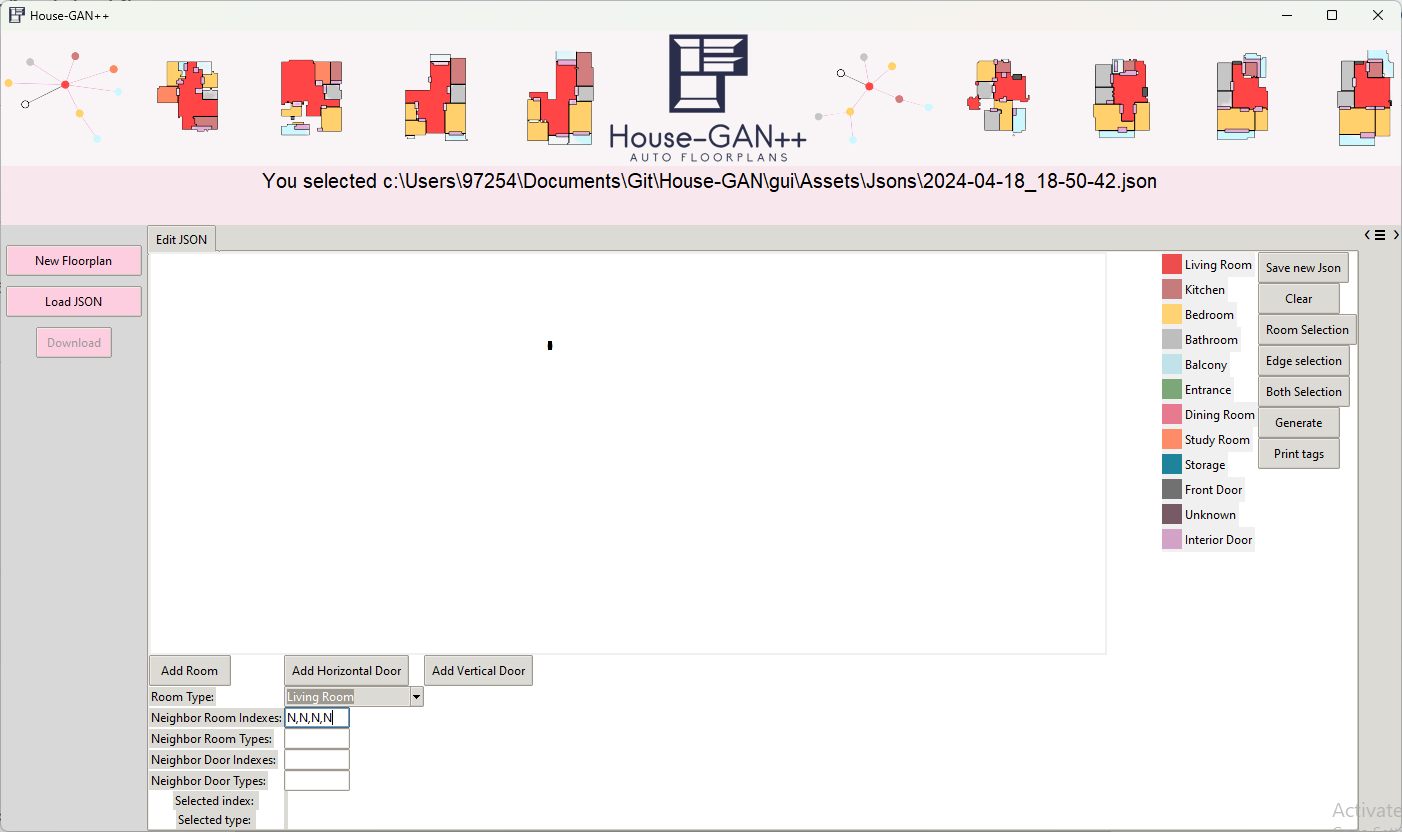
The model input is a json file describing the house layout. When adding a room, the model is expecting to know who are its neighbor rooms and doors are. Hence, the user must fill the rooms’ neighbor rooms and doors indexes and types.

#### 7.1.4.1 Add a Room - Select Room Type



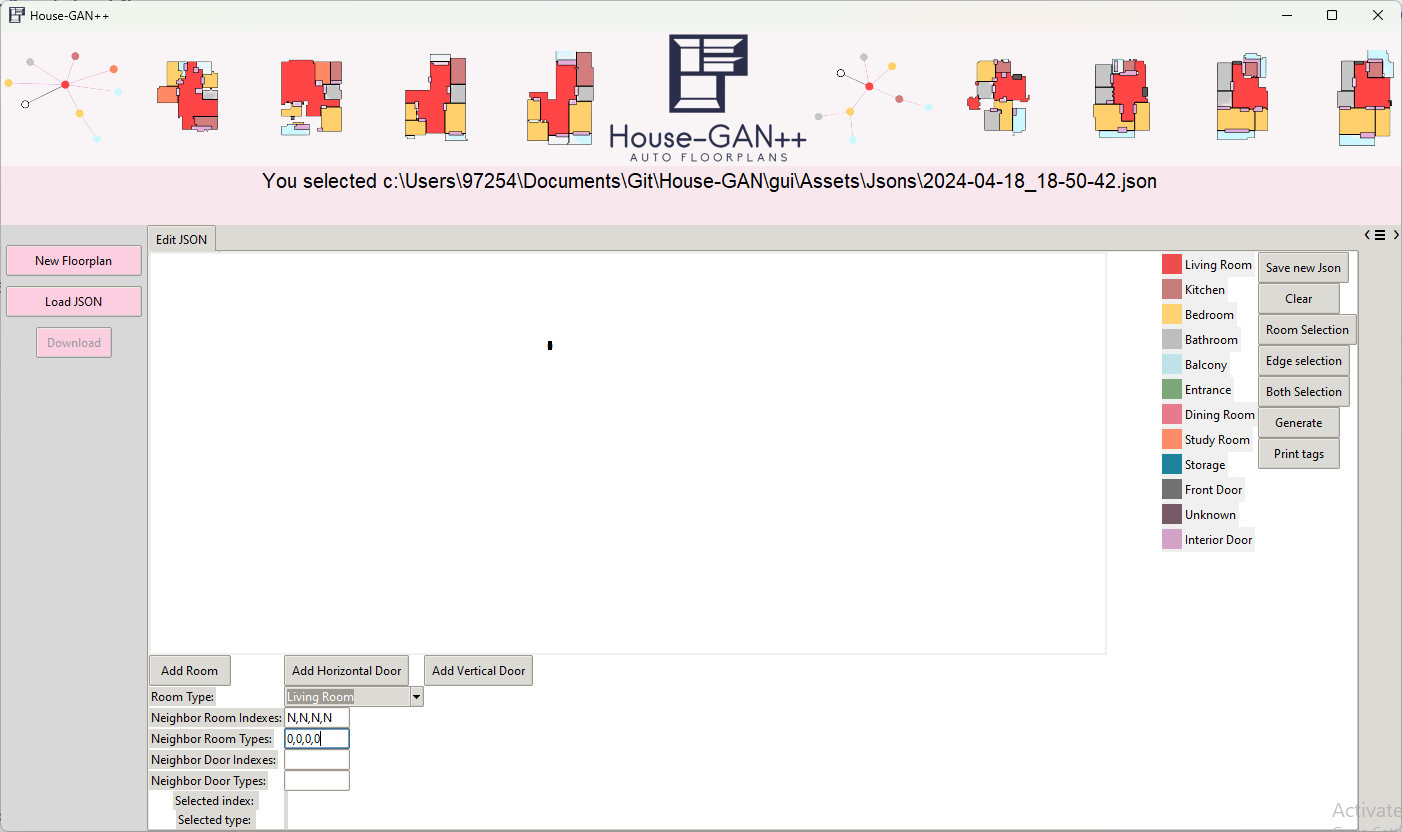
[Figure 14] In order to add a room, the user needs to select the type of room, e.g. “Bathroom” from the combobox.

#### 7.1.4.2 Add a Room - Fill Neighbour Room Indexes



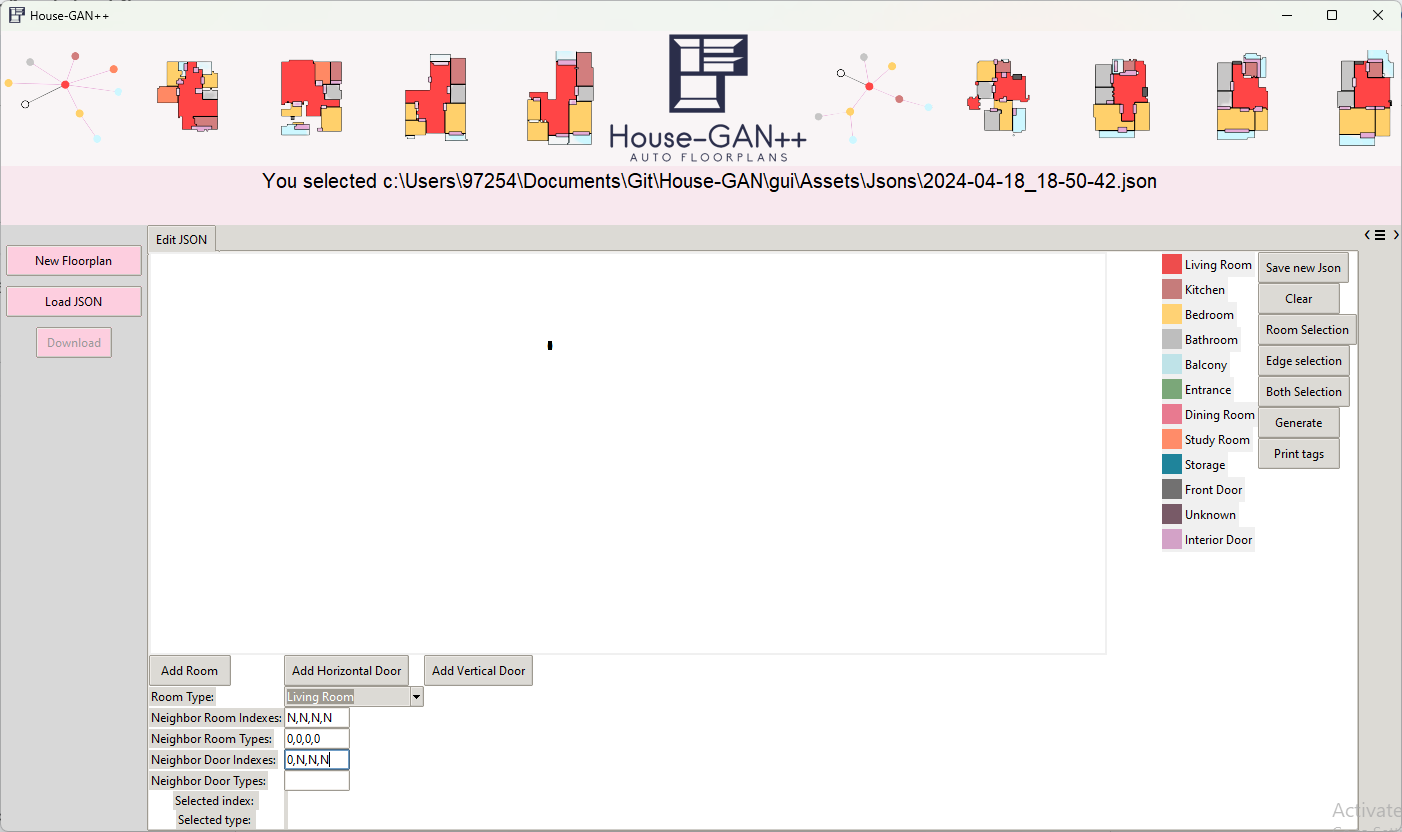
[Figure 15] The input to the model is a Json file, it describes who are the neighbors of each room, so the user must fill in the neighbors indexes of the added room. In this example there are no neighbors so fill “N” in all four elements. from left to right the positions are left, top, right, bottom. An example of filling neighbor room types and neighbor room indexes when there are neighbors are following this section.

#### 7.1.4.3. Add a Room - Fill Neighbor Room Types

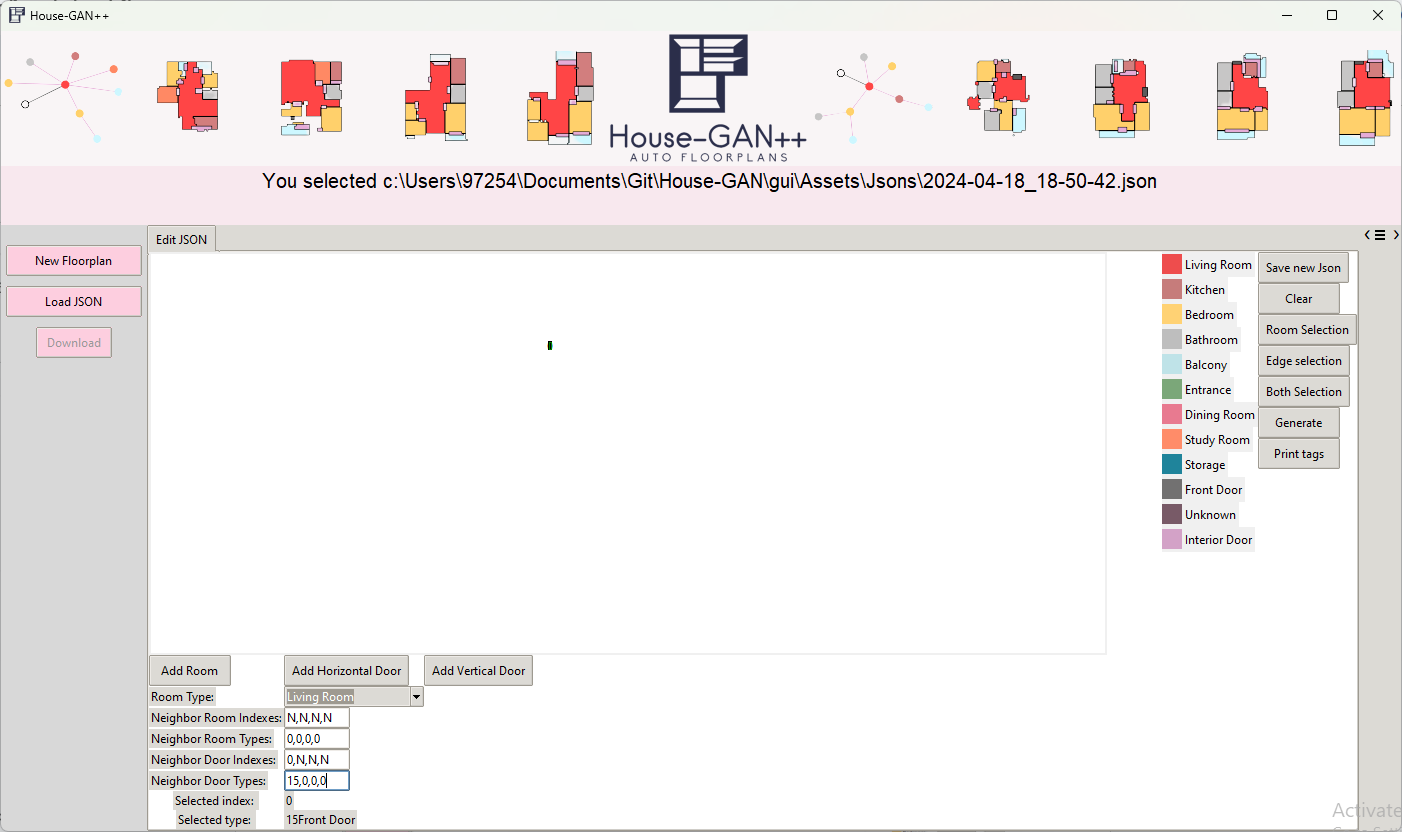


[Figure 16] The input to the model is a Json file, it describes who are the neighbors of each room, so the user must fill in the neighbors types of the added room. In this example there are no neighbors so fill “0” in all four elements. from left to right the positions are left, top, right, bottom.An example of filling neighbor room types and neighbor room indexes when there are neighbors are following this section.

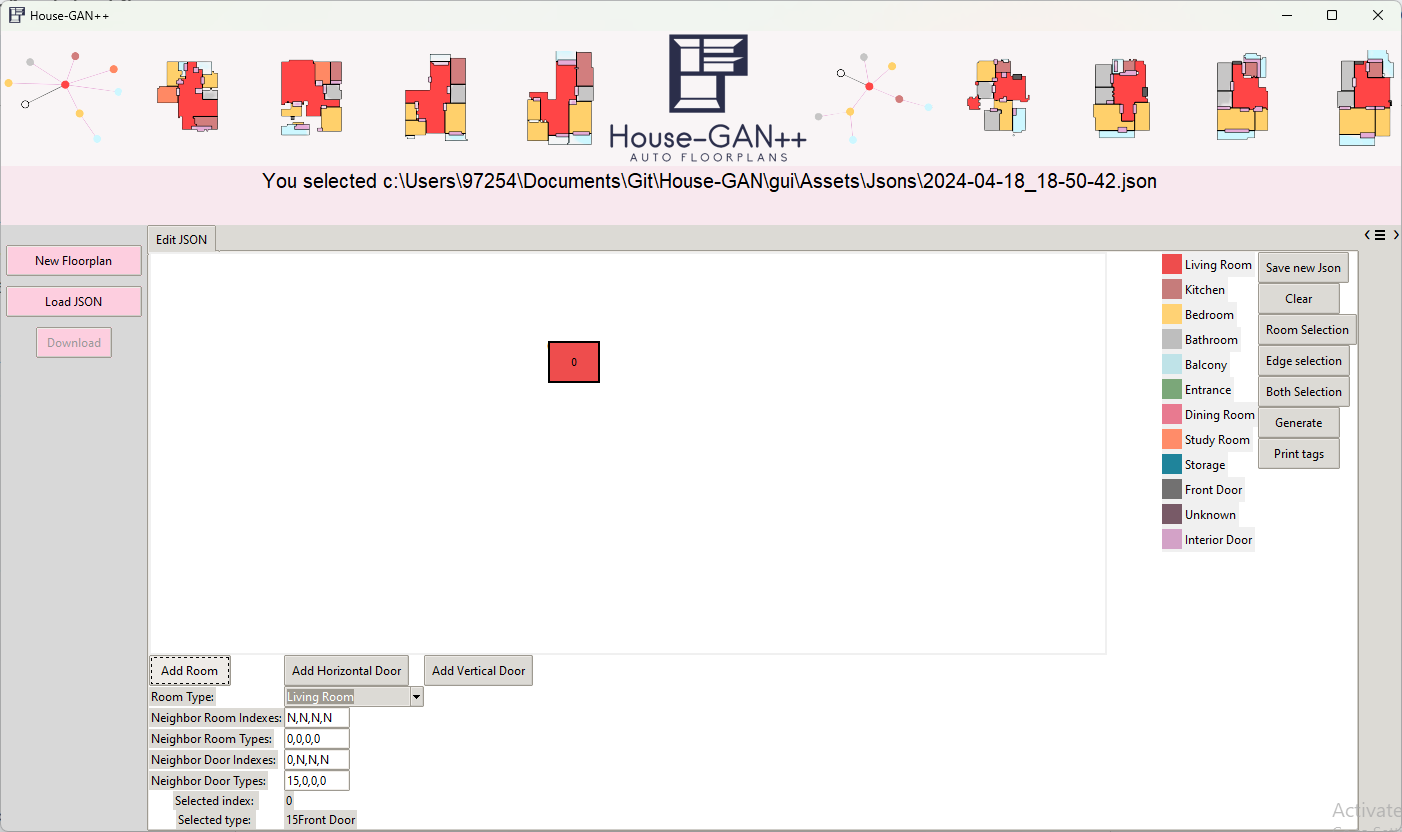
#### 7.1.4.4 Add a Room - Fill Neighbour Door Indexes

[Figure 17] The input to the model is a Json file, it describes who are the neighbors of each room, **including neighbor doors**, so the user must fill in the neighbors door indexes of the added room.   
In this example, assuming we will add the room to the right of the existing door, this means that the room will have a neighbor door to the left of it. So the user fills in “0,N,N,N”. from left to right the positions are left, top, right, bottom. An example of filling neighbor room types and neighbor room indexes when there are neighbors are following this section.

#### 7.1.4.5. Add a Room - Fill Neighbor Door Types

[Figure 18] The input to the model is a Json file, it describes who are the neighbors of each room, **including neighbor doors**, so the user must fill in the neighbors door indexes of the added room.   
In this example, Assuming we will add the room to the right of the existing door, this means that the room will have a neighbor door to the left of it. So the user fills in “15,0,0,0”. from left to right the positions are left, top, right, bottom. Where 15 is the type of “Front Door”.

#### 7.1.4.6. Add a Room - Click on “Add Room” button

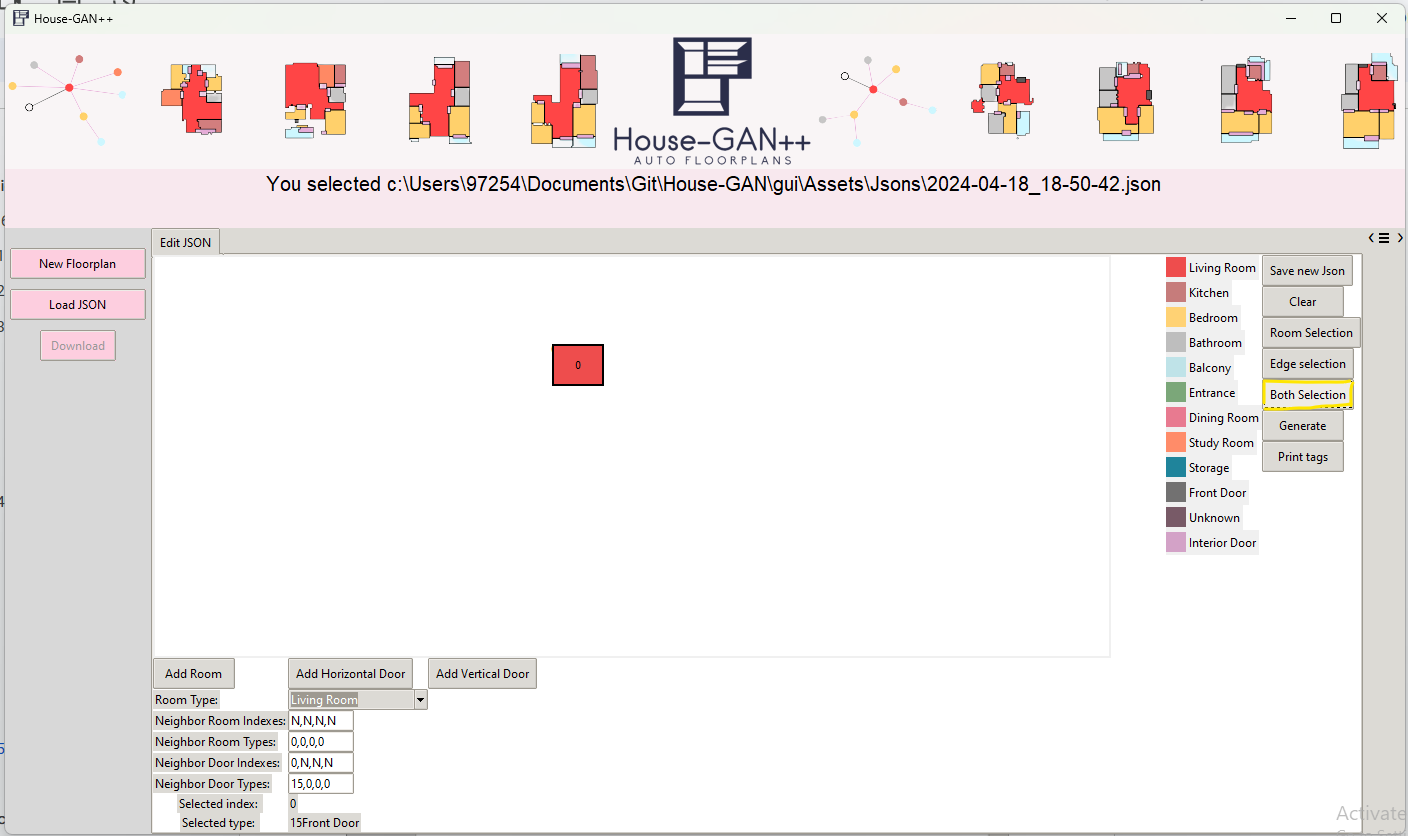


[Figure 19] A box is added to the canvas, its index is 0 and this is because this is the first box in the canvas.

### 7.1.5 Moving Rooms, Doors and Edges around the Canvas

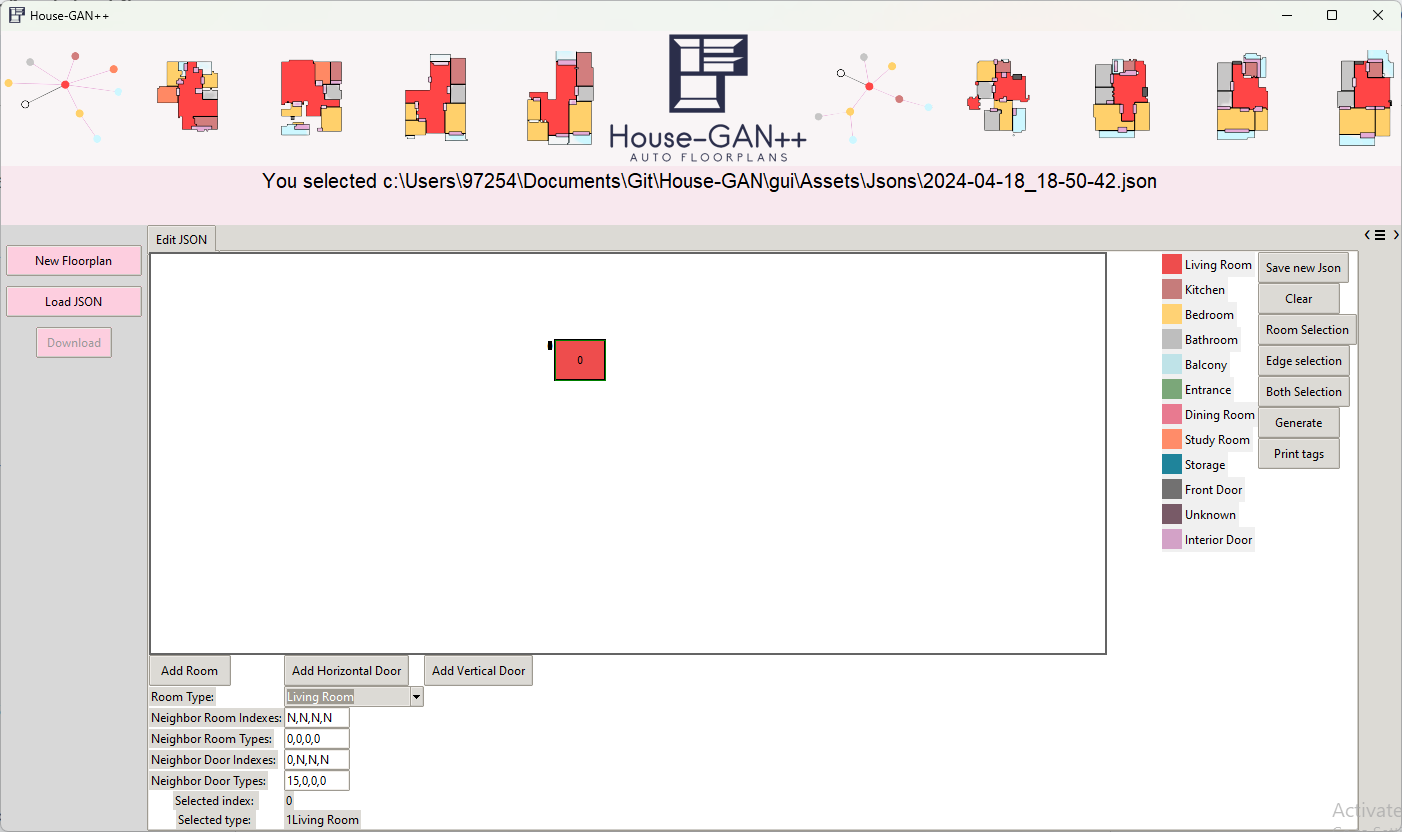
As seen in [Figure 19] the newly added room has overridden the view of the door that was present there. We can move the room box and it’ll change its position (x,y) in the canvas.

#### 7.1.5.1 Clicking “Both Selection”



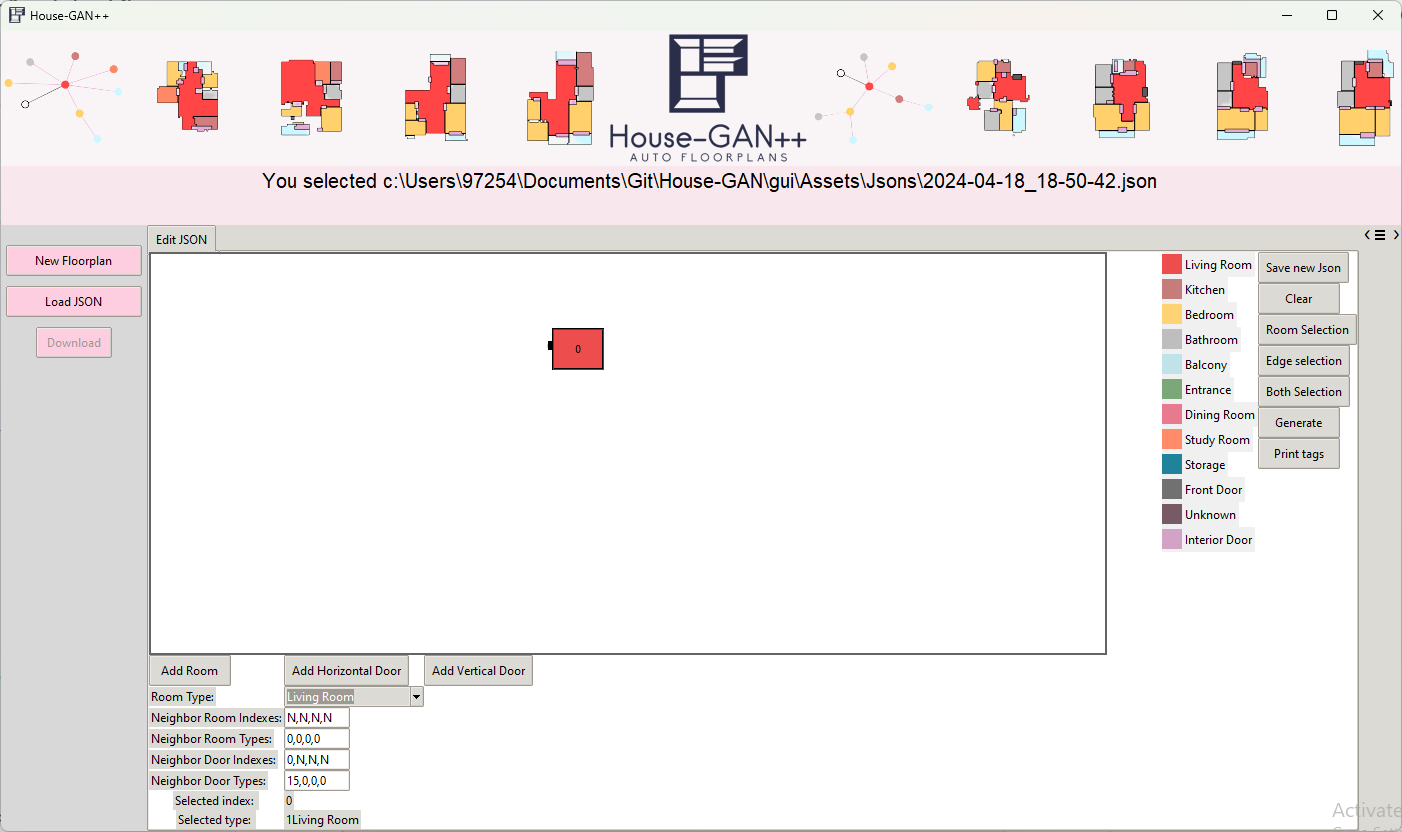
[Figure 20] Clicking on “Both Selection” button in the right pane will cause the box to move its’ box element and also its’ edges elements. Each box is compounded of edges and a box surrounding them to allow control and manufacturing of none rectangular rooms.

##### 7.1.5.1.1 Moving with Cursor



[Figure 21] Users can click the box and drag it to the desired position.

##### 7.1.5.1.2 Moving with Keyboard



[Figure 22] Users can use the keyboard arrows to move the box more precisely.

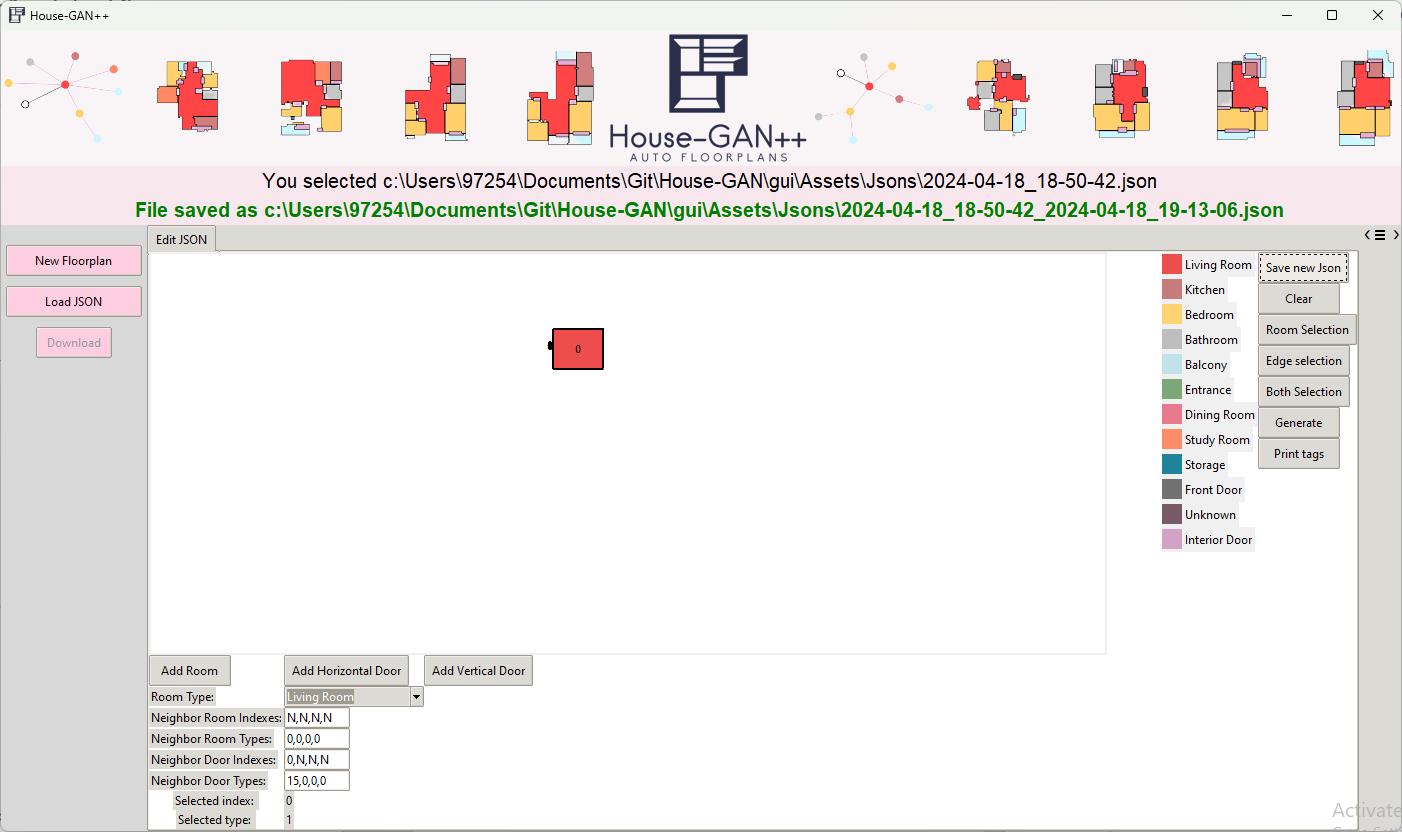
#### 7.1.5.2 Clicking “Edge Selection”

This option needs utilization and currently is disabled. Please check “Discarded Ideas” for more information.

### 7.1.6 Resizing Rooms

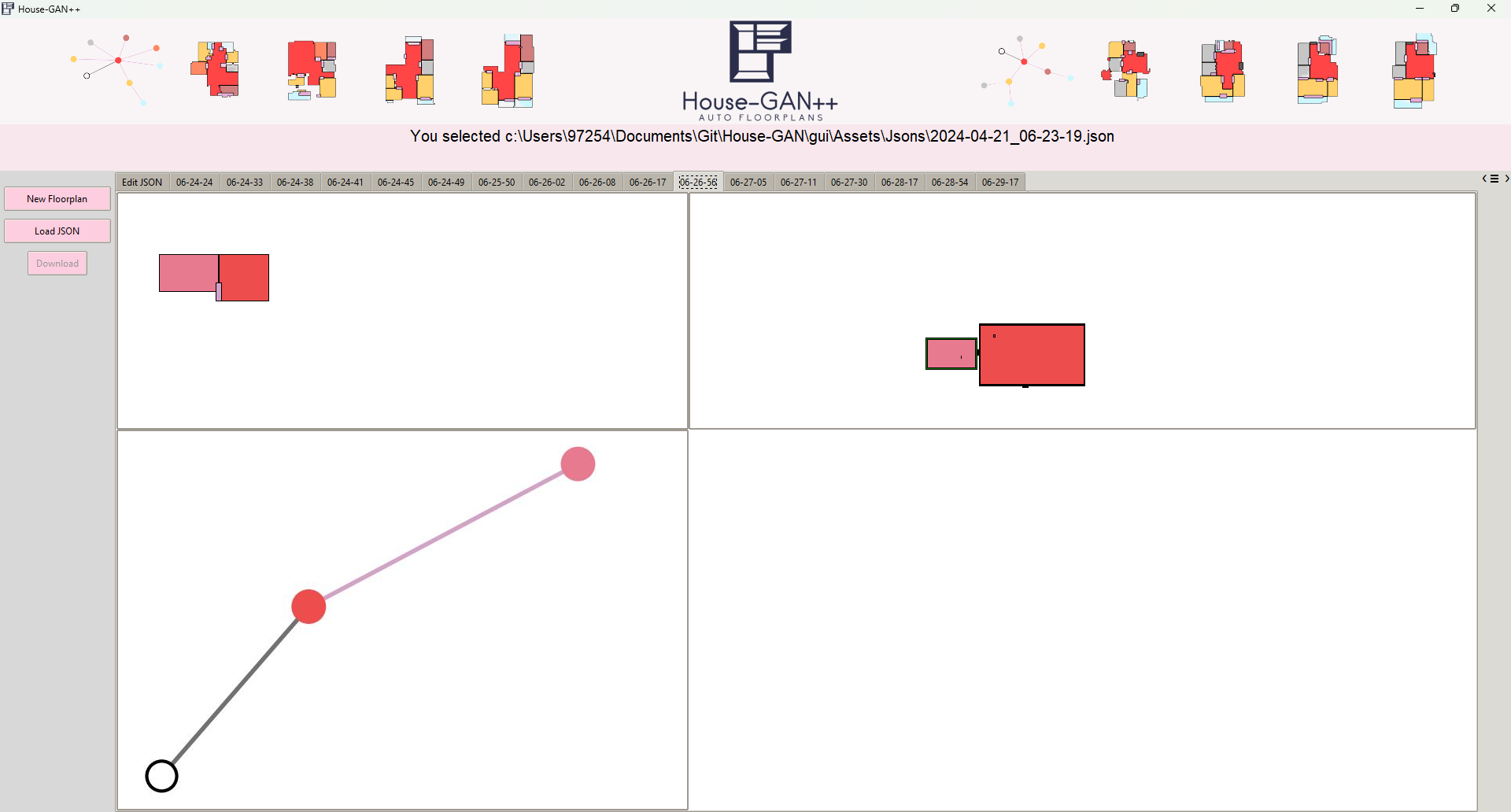
[Figure 23] Rooms can be resized to be bigger or smaller by dragging one of the edges when “Both Selection” is chosen.

### 7.1.7 Clicking “Save new Json” button



[Figure 24] After clicking the button, a path of the location where the json is saved is presented.

### 7.1.8 Clicking “Generate” button



[Figure 25] After clicking the “Generate” button, a new tab is opened containing the results from GCP. Results are the floorplan image on the top left, the graph image on the button left, and to the right the user can view what was requested.

## 7.2 Maintenance Guide

In this section, we will discuss the software and hardware requirements that you need to have in order to continue developing our platform.

### 7.2.1 Hardware Requirements

Most hardware should work since all computations are done over the cloud.

We would recommend a basic computer hardware including:

Processor: 1.8 GHz or faster processor.

RAM: 4 GB RAM or more.

Disk Space: 8 GB or more is recommended.

### 7.2.2 Software Requirements

* Software is tested and working on Windows 11
* Python 3.10.11
* python-dotenv==1.0.1
* Development Tool: Visual Studio Code
* ghostscript for windows 64-bit

### 7.2.3 Installation

install git

install python 3.10.11 with pip

install [ghostscript](https://ghostscript.com/releases/gsdnld.html)

add ghostscript /bin folder to PATH

git clone <https://github.com/DorinBe/House-GAN.git>

open project directory with vscode

pip install -r requirements.txt

add url.env to /gui directory, it is currently hidden since private credit card is used in GCP. Env file should contain URL={url of cloud run}

run main.py

### 7.2.4 Maintenance

Maintenance should be straightforward, with debugging one can comprehend the whole structure of the program since it is structured.

# 8 References

papers:

[*[1]*](https://arxiv.org/pdf/2003.06988.pdf) *Nelson Nauata 1 , Kai-Hung Chang 2 , Chin-Yi Cheng 2 , Greg Mori 1 , and Yasutaka Furukawa 1 House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation.*

[*[2]*](https://dash.harvard.edu/bitstream/handle/1/12561000/Making%20a%20Science%20of%20Model%20Search.pdf?sequence=1&isAllowed=y) *James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning, pages 115–123, 2013. 5*

[*[3]*](https://hal.science/hal-01168816v1/file/Abu-Aisheh%20-%20ICPRAM_2015_71.pdf) *Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. 2015. 3*

[*[4]*](https://arxiv.org/pdf/2103.02574.pdf)[*Nelson Nauata*](http://www.sfu.ca/~nnauata/)*,* [*Sepidehsadat Hosseini*](https://ennauata.github.io/houseganpp/page.html)*,* [*Kai-Hung Chang*](https://ennauata.github.io/houseganpp/page.html) *,* [*Hang Chu*](https://ennauata.github.io/houseganpp/page.html)*,* [*Chin-Yi Cheng*](https://www.autodeskresearch.com/people/chin-yi-cheng/) *and* [*Yasutaka Furukawa*](https://www.cs.sfu.ca/~furukawa/) *House-GAN++ Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects*

[*[5]*](https://arxiv.org/pdf/1406.2661.pdf) *Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio Generative Adversarial Nets*

[*[6]*](https://arxiv.org/pdf/1411.1784.pdf) *Mehdi Mirza Conditional Generative Adversarial Nets*

[*[7]*](https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9) *Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do Kaori Togashi Convolutional neural networks: an overview and application in radiology*

[*[8]*](https://openaccess.thecvf.com/content_CVPR_2020/papers/Zhang_Conv-MPN_Convolutional_Message_Passing_Neural_Network_for_Structured_Outdoor_Architecture_CVPR_2020_paper.pdf) *Fuyang Zhang∗ , Nelson Nauata∗ and Yasutaka Furukawa Simon Fraser, Conv-MPN: Convolutional Message Passing Neural Network for Structured Outdoor Architecture Reconstruction University, BC, Canada*

[*[9]*](https://proceedings.neurips.cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf) *Michaël Defferrard Xavier Bresson Pierre Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*

*other references:*

*[100]* [*KNN implementation*](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

*[101]* [*How Much Does It Cost To Hire A Floor Plan Designer?*](https://www.forbes.com/home-improvement/contractor/floor-plan-designer-cost/#:~:text=Rates%20start%20at%20%2450%20and,anywhere%20from%20%24500%20to%20%242%2C000.)[*Mark Wolfe*](https://www.forbes.com/home-improvement/author/mark-wolfe/)*,*[*Samantha Allen*](https://www.forbes.com/home-improvement/author/samantha-allen/)*, forbes*

*[102]* [*Fréchet inception distance*](https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance)

*[103]* [*Generative adversarial network, Wikipedia*](https://en.wikipedia.org/wiki/Generative_adversarial_network)

*[104]* [*House-GAN++ Github*](https://ennauata.github.io/houseganpp/page.html)