

Floor Plan Generation as an Optimization problem

Joel Morisset de Perdigo
ICA-6627803

Department of Information and Computing Sciences
Utrecht University

Supervised by: M.J. van Kreveld & J.L. Vermeulen

January 2021

Abstract

Floor-plan generation is the process of generating room layouts that represent potential house or building floor-plans. The generation of floor-plans is a topic that has many publications. Although some papers were designed to assist an architect in its work, no architectural software currently incorporate procedural techniques in their workflow. We propose a method that assists architects during the early design stages of a residential project, by generating a diverse set of floor-plans based on the basic requirements of the residence. Our method uses a plot area and some basic information about the number of rooms and their connections. These input requirements are then used by a Simulated annealing (SA) algorithm to explore many possibilities and present them to the architect. Our method is simple with only three move-types, which makes it easy to incorporate and further adjust to a project's needs. By using SA we allow the plot area to be fully explored, giving us good diversity but with the drawbacks of being relatively slow and slightly reducing the resulting quality.

1 Introduction

In the last 20 years, there has been an increase in the need for procedural content generation, simply due to the increase in size and scope of virtual worlds. PCGs have been applied to a wide range of topics in the creation of virtual worlds and floor-plans are not an exception. Although most papers started with the objective of creating tools to help architects in their projects, recently the focus has mostly been around the creation of virtual worlds.

Research in the generation of floor-plans started around 1970s with the focus on space allocation [10] where the authors would assign different activities to specific floors to reduce costs. The focus then changed towards either the generation of full buildings (specifically facades) or the generation of floor-plans, sometimes coupled with the generation of 3D models. The facade generation led to some important tools that can generate procedural cities (Descensor Engine, City Engine [4] and others) without interiors. More recently papers focused on the generation of floor-plans, and applied different methods either taken from other fields or designed for this purpose. Starting with grammars, usually used for facade generation, a paper published in 2006 [7] generates office buildings that can be generated and explored in real-time while allowing the users to make changes to the generation that persisted between loads. Then many methods use subdivision techniques such as the squarified tree-map algorithm [2] to generate floor-plan layouts with rooms having a close to 1 height to width ratio [14]. Other methods started using the idea of growth where rooms would be initialized as small squares and then grew using different algorithms. The first of these methods [15] can generate floor-plans for residential houses. Other methods would start using grids to simplify and reduce the odd placement of walls and would restrict the method to a predefined outer-shape, which became a common restriction.

Newer research used more complex methods like genetic algorithms [22] and agent-based systems [9]. These generations managed to create multiple floor buildings with decent layouts and made use of some optimization algorithms to reach their final results.

An early paper [17] used an optimization algorithm, similar to ours, to generate floor-plans, although, similarly to others would restrict the method to a predefined rectangular outer-shape making the methods more reliable but achieving less variability.

Still we noticed that currently, architects do not incorporate procedural generation in their day-to-day workflow, possibly due to the complexity of some methods. Although some softwares, like Grasshopper3D [8], and Revit [21] are starting to propose the use of some PCGs, it is mostly restricted to the generation of crude shapes for design ideas.

We propose a method that will focus on the delivery of options to an architect for the early stages of a project. More specifically for the sketching and early design phase of a residential project, by presenting architects with a diverse range of design ideas for a floor-plan. Allowing them to easily explore ideas and interesting layouts from only the core project schematics.

By aiming at as much diversity as possible and by keeping the method simple we easily allow the architects to add further restrictions and considerations that some specific projects may require. To create this diversity we did not want to restrict the method to a predefined outer-shape and we decided to use a plot area instead which would also be more coherent with what an architect usually has to work with. This would also allow the generation to work for a more restrictive space, by simply reducing the size of the plot to the desired area. For this reason we opted to use an optimization algorithm which would work in a similar manner to growth-based systems but would achieve higher diversity since it would allow us to easily explore different states while being oblivious to the amount of space available. And to enhance this we specifically choose Simulated Annealing (SA) as by allowing worse states makes the exploration more thorough with its know drawback of being a slow method. We additionally decided to represent the space in a modular grid which will further simplify the method and allow us to fully explore the space with only 3 moves types.

Compared to previous methods we manage to get more diversity in our results while still producing usable floor-plan layouts. The method is easily adjustable to different needs which makes it quite versatile and easily expandable. As of now the method only generates single floor houses and the generation is decently slow, but we believe that further work could tackle this limitations.

The paper will be structured as follows: section 2 will give an overview of previous work, section 3 will go over the different input requirements that we set for our method, section 4 will explain simulated annealing, including the fitness function and its parameters in the sub-section 4.2, section 5 will explain the different measures that we use to analyse the results, which are then explained in section 6, followed by section 7 that concludes the document and goes over possible future work.

2 Related Work

The generation of layouts has been an important topic in research, with applications in architecture, video games, and virtual/augmented reality. At first, the topic was mostly focused on the creation of design for assisting architects. Starting from space allocation problems for multistory buildings [10] where the objective was to minimize costs when considering the transportation of resources between floors and the positioning of airflow, elevators and other such facilities. Later the focus on floor-plans appeared after a paper on layout optimization of integrated circuits [11] (see Figure 1).

A few years later a paper was published showing the potential use of graphs in the representation and design of floor-plans [19]. This was closely followed by a paper of the organization of data where the authors created the squarified tree-map algorithm [2]. Years later a paper combined both ideas to generate floor-plans [14], where they represent a floor-plan with a tree-map and then use the squarified tree-map algorithm to generate the floor-plan layout. This method works by first dividing the area in 3 rectangles that represent the social,

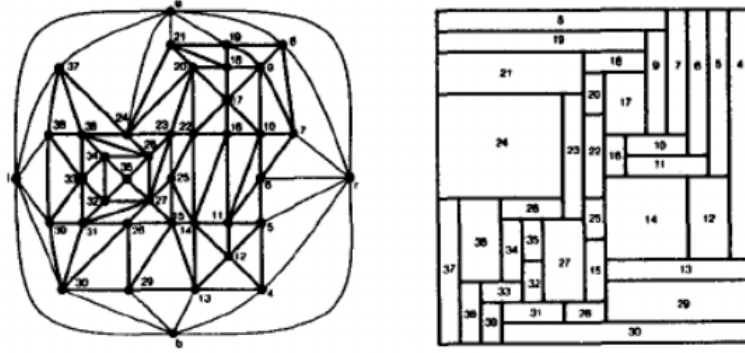


Figure 1: Layout optimization of integrated circuits in the paper [11].

service and private area, using the squarified treemap algorithm basing their dimensions on the required size of each room that constitutes each area. These areas are then subdivided into each room again using the squarified tree-map algorithm. Finally, if the connectivity is not met, the method finds the edge path, composed of inner walls excluding the living room, that will connect the isolated rooms to the living room and pushes the walls creating the corridor.

A few years prior, an overview of potential methods that could be used in the automation and generation of floor-plans [12], including the use of optimization techniques. A paper then was published using optimization techniques for the generation of layouts of multiple apartments in a rectangular floor space [17]. In this paper, they use a mixed optimization technique between simulated annealing and sequential quadratic programming.

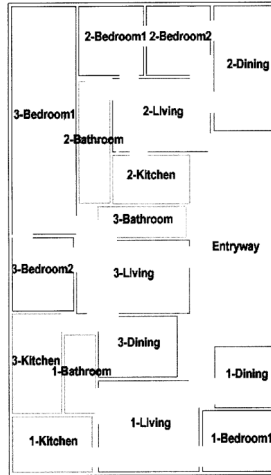


Figure 2: Resulting floor layout from the paper [17]

A different paper focused on generating deterministic fully traversable buildings in real-time [7], uses a method based on shape grammars which is a method most commonly used for facade generation. This method manages to generate interesting office-type buildings that are fully traversable, and by using a lazy generation manage, to make the project run in real-time and be persistent between runs.

Other papers used growth-based methods. The first of which [15] in 2006 Jess Martin published a paper that from a layout graph and using Monte Carlo semi-deformable growth generates a floor plan for a residential house. The method starts by generating an adjacency graph representation, by first adding the public room nodes, attached to those the private rooms and subsequently the stick-on-rooms like pantries and closets. From this graph, it places each row equally spread over the floor area. Then each room is grown using a method that attractively pushes the walls to achieve their required sizes. By using this method they managed to generate floor plans that are not bound to a rectangular area and better fit the possible layout of a single floor residential house, although the rooms themselves are rectangular. Additionally, it is the first method that explicitly generates the input graph, making it a fully procedural method. And in terms of variability, it can generate multiple floor plans given one graph (see Figure 3).



Figure 3: Resulting floor-plan generated by the growth method in the paper [15]

Similar to the previous method in 2010 a paper titled "A Constrained Growth Method for Procedural Floor-plan Generation" [13] uses a grid for the growth algorithm and fixes the outer-shape of the floor-plan to some predefined spaces. With the same growth method [3] instead creates the grid based on the outer-shape making the resulting layouts more logical in respect to the walls and Windows.

Also using a grid [5], and more focused on office buildings, this method first creates an axis-aligned grid that fits a given plot area. Using the input topology data and the generated grid, the method creates the building outline choosing one of the following strategies, either subdividing the area into rectangles and merge them until a user-specified percentage of the total space is covered. Or additionally, the method can wait until all the departments are placed and form the building outline from the generated shape. The department placement then depends on the form strategy followed. Once the departments are placed, the method subdivides each department to place all the required rooms. Finally, the circulation is computed and the corridors are generated using the edges between departments (see Figure 4).

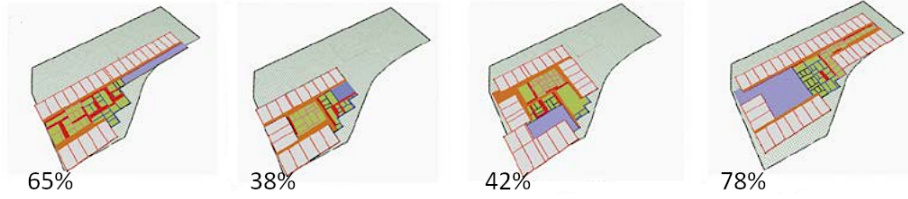


Figure 4: Example generations from the method in [5] with different coverage values.

More similar to our method [22] use a mix of genetic algorithms with a hill-climbing optimization. This method uses a very complex data representation and has different possible mutations depending on the type of room.

Other methods used genetic algorithms to optimize floor-plan layouts. The first of which [6] uses a genetics algorithm where each genotype is a tree-like graph where each node represent either a vertical or a horizontal subdivision, and the resulting leafs (end nodes) are the different rooms (see Figure 5). These graph are then mutated and crossed, and selected using a weighted sum of different measure for each objective. A crossover is made by selecting a random node used as a break point from two parents and swapping their sub-trees. And a mutations is a regeneration of a sub-tree from a randomly selected break point. This method manages to generate multifloor buildings. The generation works with rectangular rooms, but can modify the end result to add diagonal walls. Many buildings can be generated from the same data. The quality of the layout is difficult to asses since all images represent the whole building.

Only one paper published in 2016 [9] uses agent-based algorithms to generate floor plans. This papers method generates multi-floor buildings with complex room layout. It allows the generation of double ceiling rooms, elevator shaft, stairwells and other complex room types. The agent-based algorithm works by assigning to each room to an agent, that can either be a sphere or a capsule-like shape, where capsule-like agents are used to model stairwells, elevator shafts and double ceiling rooms if placed vertically, or corridors and wide rooms if placed horizontally. At the start of the simulation, the agents are placed randomly in space, and rooms that have to be adjacent have their agent connected. Then

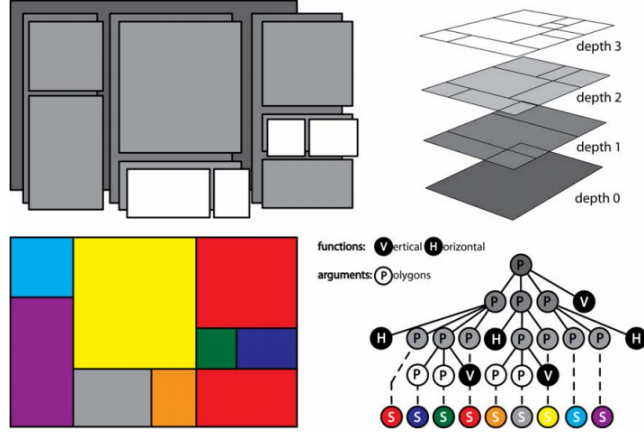


Figure 5: Example of graph representation of a layout used in the paper [6].

during the simulation the agent move towards a preferred location using different measure and once the simulation ends the agents are translated to a voxel space to which they apply an optimization algorithm that mutates the rooms until some constraints are satisfied (see Figure 6).

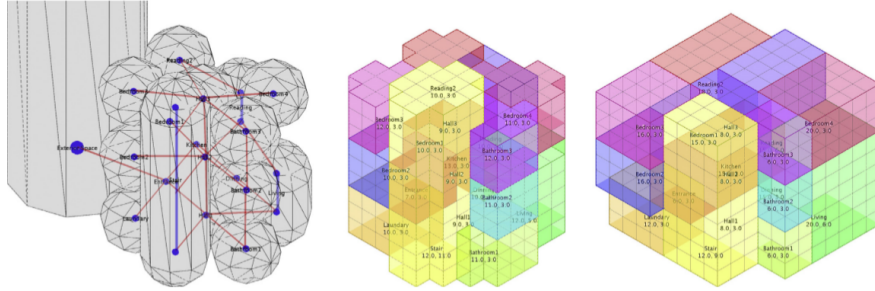


Figure 6: This figure shows how the different steps in the method from [9]

Also using an optimization approach [16] first uses a Bayesian network to generate an adjacency graph. And then optimizes the layout using a metropolis algorithm. Two moves were implemented, Sliding wall, where they adjust a wall position, and swapping rooms, where the labels of two rooms are interchanged. The optimization attempts to minimize a cost function that takes into account accessibility, dimensions, floors area, and shapes (preference towards convex rooms). From the few examples that they show, the method seems to work pretty well and produce some interesting floor-plans.

Additionally many search algorithms exist (see [1], [20] and [18]) and could be interesting to analyse and compare. We chose to use simulated annealing since it should, in theory, allow to more easily reach a higher optimum.

3 Requirements

The requirements for our method will be quite similar to previous works with a connectivity graph, some room requirements and an available space to form the floor-plan in. But in contrast to most methods, our space will be a plot space and not a fixed outer-shape which will allow more freedom to the generation while restricting the available space with a more realistic constraint.

3.1 Connectivity Graph

The connectivity graph is a tree-like undirected graph where the nodes represent the rooms and the edges represent the desired connections between rooms (accessible through a door). Each room can be connected to as many other rooms as desired. These connections will work as objective connections but our method will not force it to happen.

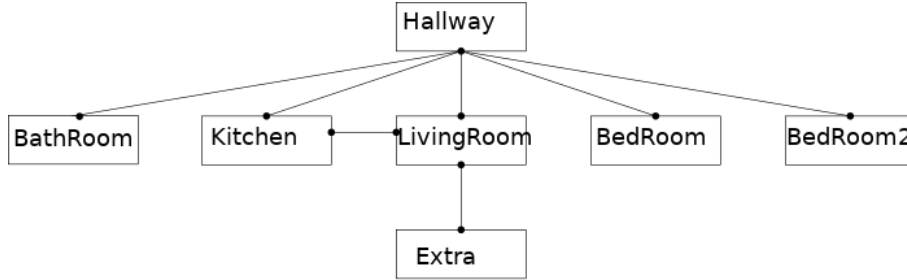


Figure 7: An example of a connectivity graph. This is also the connectivity graph used in most tests referred to a house type 1.

3.2 Room Parameters

All rooms will have a specified desired size and ratio. Represented as a minimum size, a maximum size (presented in cell units) and only a maximum ratio as this value will be calculated to always result in a ratio greater or equal to 1. The method will be drawn towards being in between those values but they are not mandatory restrictions.

	MinSize	MaxSize	MaxRatio
LivingRoom	50	60	1.5

Table 1: Example of room parameters. In this case these are the parameters used for the living room of the house seen in Figure 7.

3.3 Plot Space

The plot is a convex polygon represented by an ordered list of points. A grid will then be fitted to the space by reducing the amount of outside space, by moving the plot on the X and Y axis towards (0,0) in intervals of 5 units until the lowest X values are between 0 and 5 and the lowest Y is also between 0 and 5. Following this the grid is given an offset on the X and Y axis between 0 and 1 to maximize the number of cells inside the plot (see Figure 8) and this way increase the available grid space that the method will have to explore.

A cell is counted as inside if it is at least one unit away from the plot borders counted from the centre of the cells.

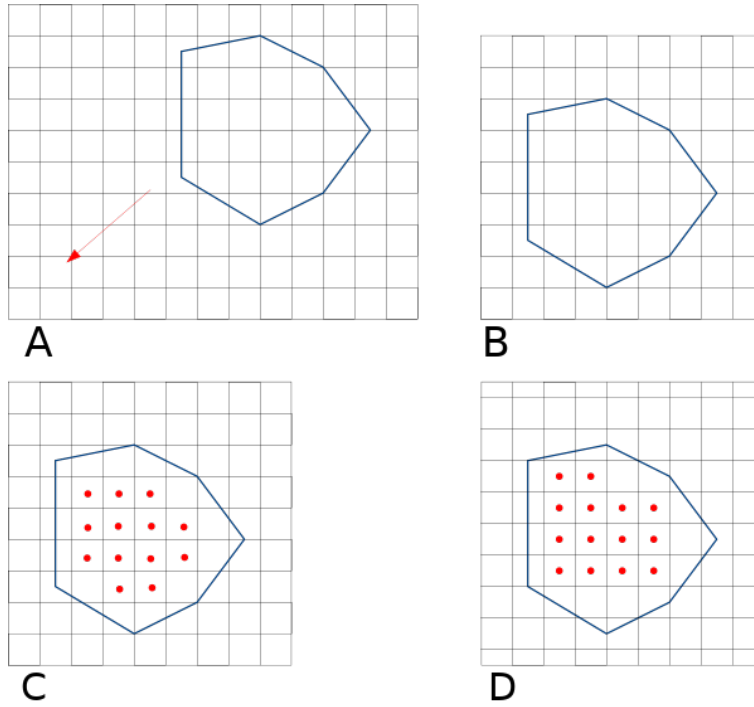


Figure 8: A grid fitting example. **A** shows the initial position of the plot. **B** shows the space after moving the plot towards the point (0,0) for this example we move the points between 0 and 2. **C** shows the counted inner cells. And **D** show the increase in inner cells (from 13 to 14) by offsetting the grid by 0.5 units in the Y axis.

4 Simulated Annealing

Simulated annealing is a probabilistic optimization technique designed to reach global optimums. It has become a popular tool for tackling both discrete and

continuous problems across a broad range of application areas. Is often regarded as an improved version of the older techniques of local search. Broadly, its process has an initial solution that is gradually improved by considering small perturbations or changes, for example, changing the value of a single variable, or swapping the values of two variables.

Since our objective is to generate a variety of floor-plans it seemed adequate to use an optimization algorithm to search the problem space and reach optimums. We then opted for simulated annealing since it should more easily reach better maximums, and by having a random chance of accepting worse results it will manage a better exploration of the space, which will allow to method to reach different local maximums therefore improving the potential diversity, even with the known drawback of this method being slow.

Simulated annealing works by modifying the current state with some randomly selected moves, creating a new state that is then measured by the fitness function also called the energy function. Based on this score we apply a formula that has a chance of accepting a worse state that scales with distance from the previous score (a worse score by 100 will be less likely to be accepted than a worse score by 10). This formula also has the parameter T called temperature which is slowly cooled/reduced over time reducing the chance of accepting worse scored states. The method then ends when it performs a number of iterations without modifying the state (for more information of Simulated Annealing see [23]).

In our method each state is a grid with each cell assigned to a room, the border of the plot or the outside area. These states are then modified by changing the room assignments with the 3 move-types presented in section 4.1.

$$P(S_{new}) = e^{(-|new\ Score - Current\ Score|)/T}$$

This formula will give the probability (P) of accepting a worse state (S_{new}), by returning a value from 0 to 1, which will be compared to a random value also from 0 to 1 and if said result is bigger, then the worse state is accepted.

As for the cooling parameters used in this method, they were selected by trial and error (see Table 2). The cooling is applied by multiplying a factor by the current temperature at the end of each iteration.

Initial Temperature	Cooling Factor	Number of Iterations to stop
40	0.9999995	2000

Table 2: These are the selected values for the SA parameters.

To simplify the search space and the implementation of the moves we base the generation on a modular grid that can be easily adjusted. For the experiments we performed, we opted to make the cells 40cm wide, which is why we then consider that 2 cells are required to possibly have a door, and to fulfil a connection. This value could easily be adjusted to be smaller or larger to fit the desired complexity.

Additionally it is important to mention that all weights, function parameters and the SA values were selected by trial and error and further improvements and refinements can always be made.

4.1 Created Moves

To allow the system to fully explore the space we created 3 moves.

- **Add Cell:** which for a room takes a random bordering cell and expands in the direction of the contained wall (see figure 9). When performing this move the method makes sure that the selected cell is inside the plot and does not belong to any other room.
- **Remove Cell:** which for a room takes a random bordering cell removes it (see figure 10). When performing this move the method makes sure that removing this cell will not split the room, and that the room has more than one cell.
- **Steal Cell:** whenever two rooms are adjacent a room may steal a bordering cell from a different room making it its own (see figure 11). When performing this move the method makes sure that the cell it is stealing will not split the room and that the room it is stealing from has more than one cell.

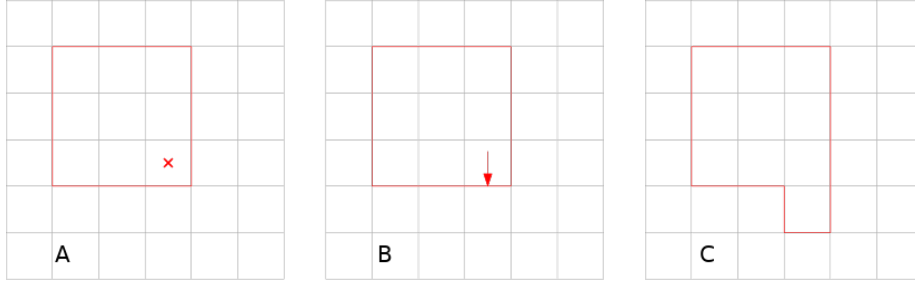


Figure 9: An example of the Add Cell move type. **A** shows initial state with the selected cell. **B** shows the direction chosen for the growth. And **C** shows the new resulting room.

4.2 Fitness Function

The fitness function (also called energy function in SA) is composed of multiple parameters each with their own weight. The weighted sum of these represents the overall fitness of a state.

These parameters were created to attempt to cover the basic aspects of a correct floor-plan. Further parameters could be added and refined to improve or modify the results.

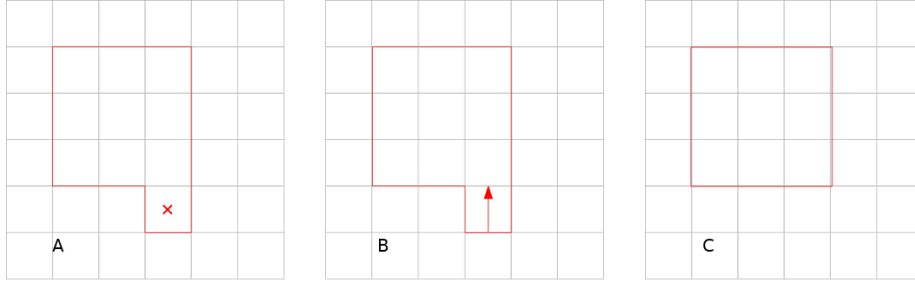


Figure 10: An example of the Remove Cell move type. **A** shows initial state with the selected cell. **B** shows the direction chosen for the shrink. And **C** shows the new resulting room.

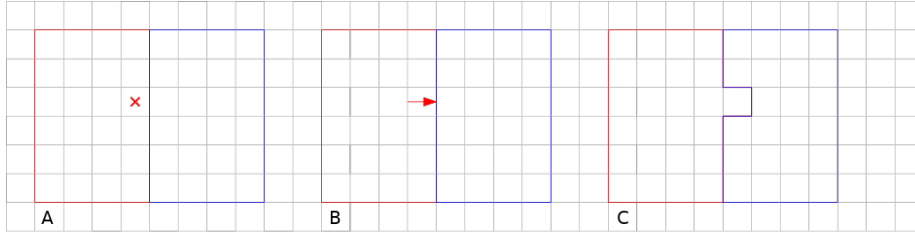


Figure 11: An example of the add Steal Cell move type. **A** shows initial state of both rooms with the selected cell. **B** shows the direction chosen direction for the stealing. And **C** shows the new resulting rooms where the cell is removed from the blue room and added to the red room.

1. Minimum size and Maximum size: based on the desired maximum and minimum size, this parameter adds points when the room's size is in the accepted range and subtracts points if it is below its minimum size or above its maximum size. The value is calculated based on the difference between the current size and the closest desired value.

Below the minimum:

$$MinimumSize += (size - minSize) * 1.5$$

Resulting in a negative value.

Above the maximum:

$$MaximumSize += (maxSize - size) * 1.5$$

Resulting in a negative value.

Between the accepted range:

$$MinimumSize += (size - minSize)$$

$$MaximumSize += (maxSize - size)$$

Resulting in positive values.

The multiplication by 1.5 is done to increase the impact of the value since in such cases it will only be using only one of the size functions. This also accentuates wrong sizes and further penalizes far-off values.

Maximum and minimum size each have their own separate weight when calculating to overall fitness.

The hallway is not affected by this, and instead only has a penalty that increases with its size.

$$HallSize -= size^{1.2}$$

The exponent of 1.2 is added similarly to the added multiplication of the previous formulas to increase the negative impact of bad values and further accentuate it the bigger it becomes.

2. Ratio: The ratio of the maximum length and width of a room will always be above 1, therefore only a maximum ratio is needed. When the current ratio is below its maximum we give it 1.5 points and when the current ratio is above its maximum we use the following formula:

$$Ratio += (maximumRatio - ratio)^3$$

This will result in a negative value. Again, an exponent is added to further penalize bad values.

3. Average Distance: This parameter is a weighted average of the distance between all rooms. The value is calculated by getting the distance from a room to all other rooms (using their centre point) minus the room's square root of the current room maximum size (This prevents rooms from attempting to reduce their size to increase the distance parameter). When the distance calculated is part of the desired connection its value is multiplied by 1.5. Once all distance values are added they are divided by the total number of distances calculated to get the average.
4. Rectangularity: This parameter will penalize rooms simply based on their number of corners:

$$Rectangularity -= (corners - 4)^2$$

By subtracting 4 we only penalize rooms that are not rectangular. In the case of hallways, the power is 1.2 instead of 2 since it is more common for corridors to have corners.

5. Connectivity: The connectivity parameter is the most complex. For each wall on an outer cell of a room we check whether it is connected to a room, the value of it will then depend on whether it is or not connected to a room and if that room is in its desired connections. The values are then divided by a factor that increases each time a connection is counted, to avoid having room surround others to maximise the connectivity points.

When the room is a desired connection:

$$Connectivity += 2.5 / ConnectedFactor$$

When is adjacent to a non desired room:

$$Connectivity += 1 / ConnectedFactor$$

And when it is not adjacent to anything:

$$Connectivity -= 1 / NonConnectedFactor$$

The connected factor starts at 0.25 and also increases by 0.25 each connection, while the non-connected factor starts at 0.5 and increases by 0.75. This is done to reduce the impact of outside walls. Additionally each wall direction has its own factors.

When a connection is fulfilled an additional 10 points are awarded, meaning that two or more cells have a wall adjacent to the desired room (only awarded once per connection).

6. Bounding Box: For this parameter, we find the current bounding box of the floor plan and calculate its size, then this value is compared to the sum of maximum sizes of all rooms.

When the current bounding box size is above the floor plan calculated maximum:

$$Bounding = 25 - (Size - FloorplanMaximum)$$

If bellow:

$$Bounding = 25 + (FloorplanMaximum - Size)$$

We use a value of 25 chosen by trial and error that is the granted score if the bounding box size equals the floor-plans maximum values.

7. One cell-wide: This parameter was added to penalize one wide corridor or room "arms". And it is calculated by adding -1 for each cell that has walls on opposite sides. This penalization is added for accessibility purposes because of the chosen cell width of 0.4m.

8. Holes: To penalize holes inside the floor plan, we explore the outside area contained inside the bounding box of the floor-plan and subtract the sum of current room sizes, the resulting value gives us the total size of holes.

$$Holes -= (BoundingBoxSize) - (OutsideArea + \sum CurrentRoomSizes)$$

9. One wide hole: Holes penalization only takes into account holes contained inside the layout, this means that as long as an empty cell space opens the area to the exploration it will not be considered as a hole. Therefore to reduce the impact of potential cases where holes are maintained open to increase the fitness score we added an extra penalization. This looks at outer room cells and at two subsequent cells in the direction where the wall is and if it finds that there is an empty cell followed by an occupied cell it gives a penalty of -0.2 which is added for each occurrence.

Each of these parameters are added together in a weighted sum each with their own multiplier, again selected by trial and error.

$$Fitness = \sum (ParameterX * WeightX)$$

<i>MinSize</i>	<i>MaxSize</i>	<i>Ratio</i>	<i>AverageDistance</i>	<i>Rectangularity</i>
3.5	2.4	1.5	12.4	2.5
<i>Connectivity</i>	<i>Bounding</i>	<i>OneCellWide</i>	<i>Holes</i>	<i>OneWideHoles</i>
1.4	0.25	8.2	3.0	1.0

Table 3: Weights used for each parameter.

In the figures 12, 13, 14 and 15, you can see resulting floor-plans each with a missing parameter with the exception of the no sizes where both the minimum and the maximum size parameters were removed, and the addition of an example with all parameters seen in figure 15.

4.3 Initialization

Our system starts by taking the input requirements and the plot and creates the data structures that will be used during the optimization process. Then sets the plot and grid as explained in Section 3.3. Once this is done the system selects a number of cells contained in the inner region of the plot that will be used as potential starting locations for each room. These cells are all cells that lie inside the plot at a distance higher than the width of the plot divided by 5.

Once this setup is done we add each room to the grid. Their starting location is fully random with the only restriction that a room cannot be placed in between two rooms that are supposed to be connected. This is done by verifying every connection and checking the distance between the connected rooms and the

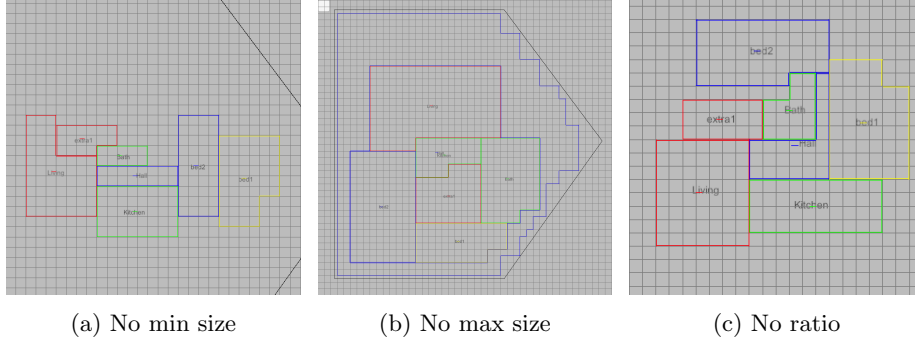


Figure 12: Results without the specified fitness parameter.

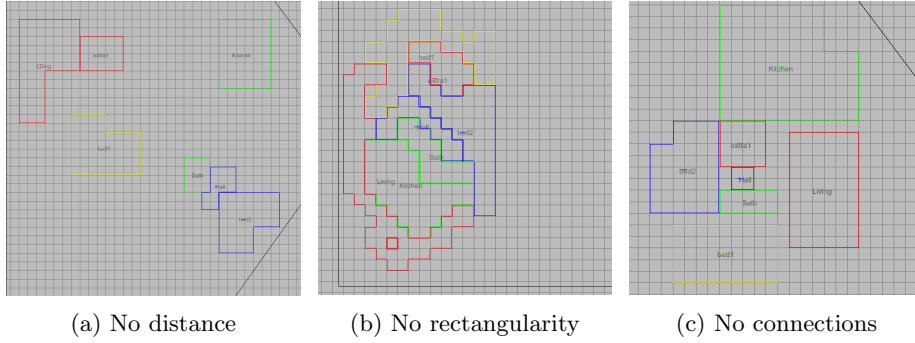


Figure 13: Results without the specified fitness parameter.

distance between the first room of the connection and a third room. If this distance is lower, then we check if the angle of the vector between the two connected rooms and the vector with the third room is lower than 20° . If this happens the rooms are removed and randomly placed again (see Figure 16).

Once all the rooms are placed we start the optimization process with SA as previously explained. Where each iteration the method applies a random move to a room. The rooms are selected by cycling through them in order.

5 Evaluation Method

For the evaluation instead of only analysing the generated floor-plans qualitatively, and since we are not performing a user study, we designed some measures to evaluate the quality and the diversity of the results.

5.1 Requirement Score

The Requirement score was designed to give a score based on how well the method achieved what was specified and this way give somewhat of a quality

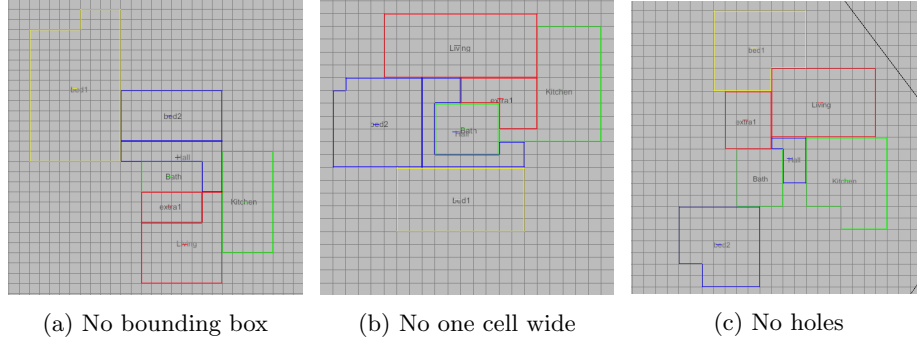


Figure 14: Results without the specified fitness parameter.

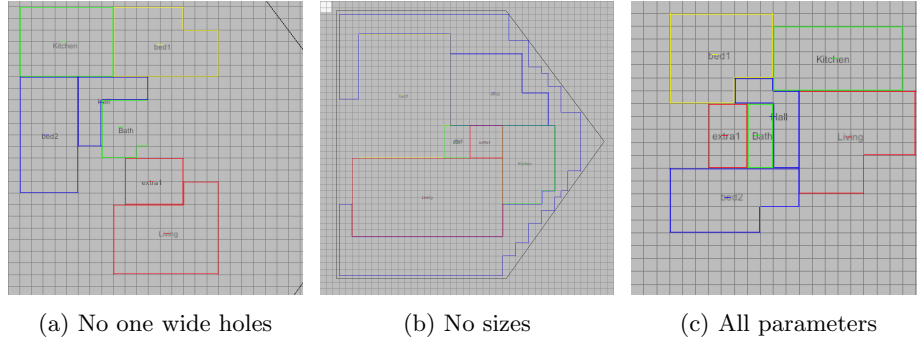


Figure 15: Results without the specified fitness parameter and finally a result with all parameters.

score. Although one could think that the fitness function would be enough, its values vary greatly based on the number of rooms and other factors. Therefore we created this measure that calculates a value based on how well the resulting floor-plan has met its input parameters.

The Requirement function looks at 7 parameters, based on the input specifications:

- The connectivity, is each room connected to the rooms specified by the connectivity graph (we count a valid connection when the room is adjacent to it by at least two cells)
- The size, counted when its value is between the minimum and maximum specified size.
- The ratio, as for the size counted when in between the minimum and maximum specified ratio.
- The adjacency, where it makes sure that rooms are not disconnected.

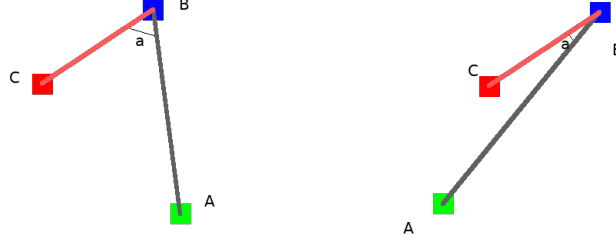


Figure 16: Room placement examples, showing an accepted positioning (Left) and a positioning requiring to remove and replace room C (Right).

- The bounding area, which has to be under the sum of all maximum sizes of all rooms.

Additionally we penalize corridors or areas that are one cell wide.

All of these parameters are either met or not met, which means that we know for each floor-plan the maximum possible value, therefore to be easily comparable between all types of houses we can scale it to give a score up to 100. Where 100 would mean that all requirements are met and no penalization has been added by one cell wide areas.

5.2 Distance Function

To measure the diversity we created a distance measure that compares two floor-plans and returns a value of 0 when both floor-plans are the same or are a simple rotation of the same floor-plan, and give higher values the more different they are.

For this function, we compare 5 parameters from the two resulting floor-plan layouts:

- Adjacency, where we compare each pair of analogue rooms and take the differences in their adjacent rooms.
- The size, where for each pair of rooms we take the difference the room's sizes.
- The ratio, where we take the difference between each room's ratios.
- The shape, where for each pair of analogue rooms we compare the number of corners.
- The bounding box ratio, where we compare the overall difference between each floor-plan bounding box ratio.

5.3 Qualitative Scoring

To be able to verify the effectiveness of the Requirement and the Distance measures, we also gave a qualitative score based on our perception of the generated floor-plan layouts.

For the quality, we gave a score from 0 to 4 where 0 is a very bad layout and 4 is a very good layout. When choosing these values we followed some simple rules of thumb:

- A score of VG very good(4) was given if no changes had to be made to the layout in order for it to be logical.
- A score of G good(3) was given if only a small adaptation (e.g. moving a room by a few cells, changing a room size or shape a little...) was needed to make the layout logical.
- A score of A average(2) was given if one or two moves (e.g. Switching two rooms, moving a room to the other side...) were needed to make the layout logical.
- A score of B bad(1) was given if 3 moves were needed to make the layout logical.
- A score of VB very bad(0) was given if more than 3 moves were needed to make the layout logical.

We consider a layout logical when it could be used as it is by an architect, even if some aspects could still be seen as odd or imperfect.

When giving a score to the distance we also gave a value from 0 to 4, where in this case 0 means that two floor-plans are equivalent and 4 that they are completely different. No further rules were specified, but when comparing we considered that simple rotations and flips of a layout did not count as different, and we specifically paid attention to room clusters and their adjacencies, meaning that two layouts that have similar clusters featuring the same adjacencies (without considering room shape) were seen as similar.

It is important to mention that these scores were given by one person and that no user study was performed.

6 Results

To get results the method was developed using unity for the visualization. We performed 17 experiments, for each we generated 10 floor-plans. First we ran the program with 3 different house types, with their respective connectivity graphs seen in figure 17 and their respective room parameters (see Table 4, 5 and 6).

For these experiments, it took the method around 3 million iterations (see Table 7), with each house type slightly increasing in time with the complexity. And on average it took 62 minutes to generate each floor-plan of the house type

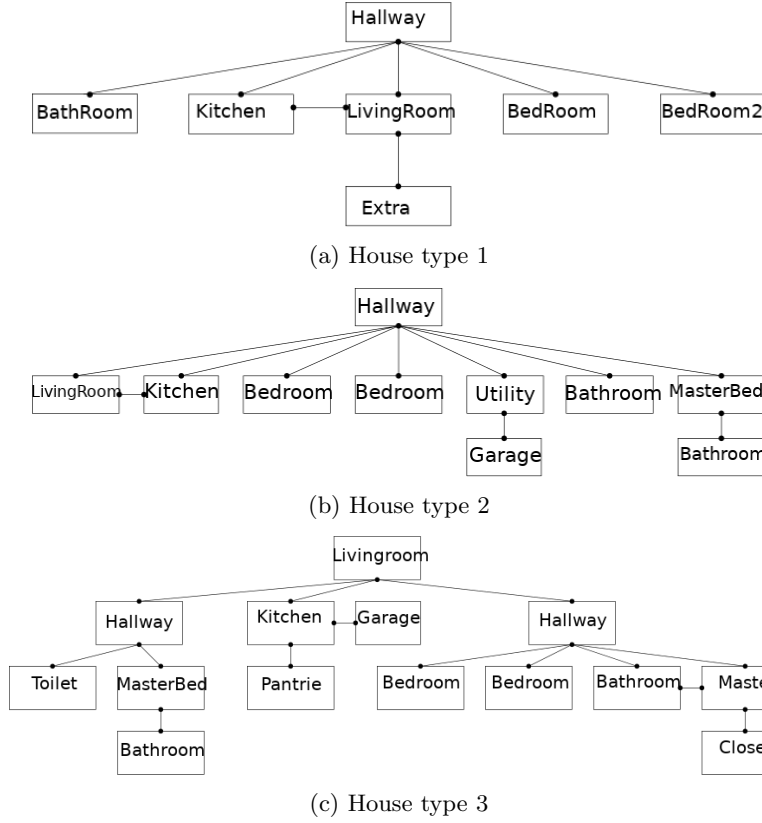


Figure 17: Connectivity graphs for each house type.

1, 55 minutes for each floor-plan of the house type 2 and finally 98 minutes per floor-plan for the house type 3, while the number of iterations stayed around 3M. Although the initial time to completion is quite high, the method seems to scale quite well. Where from 7 rooms in house type 1 to 14 rooms in house type 3, meaning that for an increase of 200% the time only increased by 158%. For the subsequent experiments the time needed varied but it is not relevant since they were mostly bound to how the fit function changed. The bottleneck of our implementation is mostly the grid size since to find holes the method explores all outside cells around the floor-plan, the following bottleneck is also bound by the number of cells because of the check for a potential split point in the remove cell move type. This means that by reducing the plot size and/or increasing the width of the cell we could reduce the time needed drastically. To test this we performed an experiment with the cell width to 0.8cm double our normal size. This made each floor-plan take 29 minutes on average, therefore reducing the time needed by half.

Additionally to the increase in time, the results from the three different

House Type		MinSize	MaxSize	MaxRatio
1	Hallway	-	-	-
1	Living room	50	60	1.5
1	Kitchen	40	50	1.5
1	Bedroom	40	50	1.5
1	Bedroom	40	50	1.5
1	Bathroom	10	20	1.5
1	Extra	10	20	1.5

Table 4: Room parameters for house type 1.

House Type		MinSize	MaxSize	MaxRatio
2	Hallway	-	-	-
2	Living room	30	40	1.5
2	Kitchen	20	30	1.5
2	Bedroom	15	25	1.5
2	Bedroom	15	25	1.5
2	Bathroom	10	20	1.5
2	utility	5	15	1.5
2	Garage	30	40	1.5
2	Master Bedroom	20	30	1.5
2	Bathroom	10	20	1.5

Table 5: Room parameters for house type 2.

house type seem to suggest that the method will perform worse the more rooms and connections we have. In the table 8 we can see how the quality is worse for the house type 3 and the distance between the floor-plans increases.

The fitness score due to the use of SA is very volatile at the start and slowly increases while reducing its spread. In figure 18 we can see how fitness score progresses and how it seems to stabilize around the sample 650 (2.9M iteration). Additionally, it seems as if the method never fully reaches stability which could be due to our sample rate (3000 iterations) which is bigger than our stopping condition (2000 iterations, and could suggests that our stopping condition should be longer.

When looking at the results from the experiments without one of the fitness parameters (see Figures 12, 13, 14, 15, and Table 9), we can see how much of an impact each parameter has. In some cases the resulting floor-plans are to be expected, examples of this are the removal of the maximum size, all rooms will take as much space as possible, or the removal of the rectangularity parameter which will make the rooms grow randomly and end up with a stair-like blob shape. The opposite case, when removing the minimum size, does not make the rooms become one cell but do seem to make the rooms become smaller than they should. Removing the ratio does not seem to affect the

House Type		MinSize	MaxSize	MaxRatio
3	Hallway	-	-	-
3	Hallway	-	-	-
3	Living room	30	40	1.5
3	Kitchen	30	40	1.5
3	Toilet	5	15	1.5
3	Bedroom	20	30	1.5
3	Bedroom	20	30	1.5
3	Bedroom	20	30	1.5
3	Bathroom	10	20	1.5
3	Pantry	5	15	1.5
3	Garage	30	40	1.5
3	Master Bedroom	20	30	1.5
3	Closet	5	15	1.5
3	Bathroom	10	20	1.5

Table 6: Room parameters for house type 3.

Type	Avg Iterations	Total Time (sec)	Avg per Floor-plan (min)
1	3,136,510.6	37,251.74	62.09
2	2,824,203.2	33,040.88	55.07
3	3,108,501.75	59,377.33	98.9

Table 7: Running time for the generation of 10 floor-plans for each type of house.

generation too much as the sizes are still met and the rectangularity plus the random growth make the rooms tend to a squared shape. When removing the connection parameters the layout looks similar but there is no logic in the placement of the rooms. In the results without the holes penalization, no holes actually appeared but we have seen it happen in other runs. Additionally, when removing the one wide penalty we get long "arms" additions to the rooms, but it seems to help the corridor shape, although they would be considered too narrow. Removing these parameters also affects the requirement score as the results without rectangularity get the highest average with 95.1, since the method is no longer restricted to a room shape and makes it easier to reach the connections. These removals also seem to increase distance, which suggest that removing restrictions will increase diversity due to the randomness of the method.

From the resulting generation, we can see that our method manages to produce very different and diverse layouts. Where the distance measure only return 0 when comparing the same floor-plan together. This can also be seen in the qualitative evaluation in table 10 where all house types have an average distance score above the 2 (the middle between 0 and 4), and with house type 3 having

Type	Avg Fitness	Fitness SD	Avg Req*	Req* SD	Avg Distance
1	271.1	384.2	78.3	15.8	30.7
2	618.1	254.4	85.6	6.1	40.7
3	629.5	424.5	63.4	41.8	51.36

Table 8: Statistics from the experiments from each type of houses. *Req refers to the Requirement score.

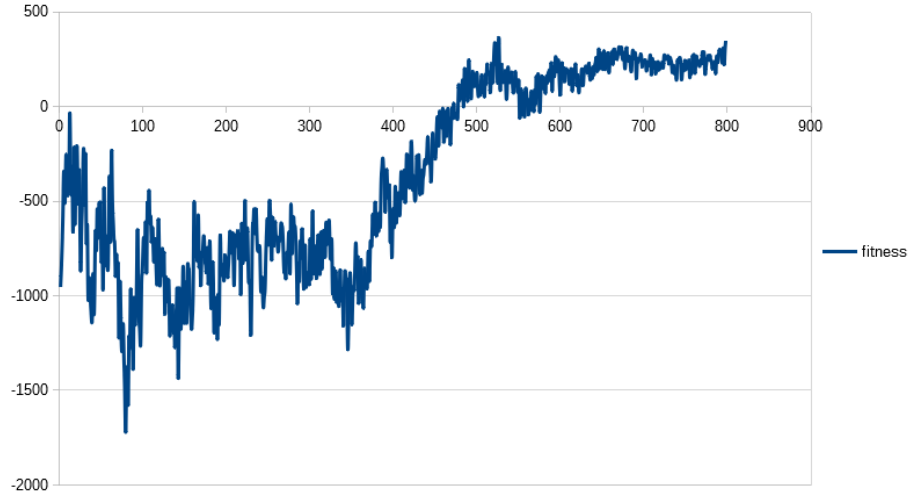


Figure 18: This graph shows the progression of the fit function over time. The X axis is the sample number taken every 3000 iterations, and the Y axis is the fitness score. For this experiment we used room type 1.

a maximum score as the average.

When analyzing the measures with the qualitative score (see Table 10) we find that there is a decently high correlation between the fitness score and the requirement score of 0.52. But this does not translate so well when compared to our qualitative score from 0 to 4. Where the correlation stays at 0.22. When comparing the distance function to our qualitative values we get a correlation of 0.33 which is better but still not ideal. This seems to point out that our measures are too simple compared to what we look for in a house layout. Of course, the requirement score is only showing how well the input conditions were followed and not the quality. And that the distance measure is only looking at internal values and not at more complex considerations.

Type	Avg Fit	Fit SD	Avg Req*	Req* SD	Avg Dist
Normal 1	271.1	384.2	78.3	15.8	30.7
Ratio	395.6	198.9	78.6	7.8	26.1
Rect	1072.8	75.5	95.1	5.7	75.7
1 Wide	612.3	94.8	32.8	19.3	21.2
Min Size	352.5	107.5	73.7	12.6	31.9
Max Size	-860.50	500.7	46.8	14.1	108.7
Size	-562.6	343.9	22.1	30.9	108.1
Holes	389.4	135.8	76.5	10.8	27.7
1 WideHoles	382.2	166.5	63.1	36.1	32.7
Dist	202.7	167.9	68.3	14.8	25.3
Connectivity	-2.7	113.2	64.6	22.8	20.5
BoundingBox	260.1	305.3	79.7	4.7	31.9

Table 9: Statistics from the experiments without each fitness parameter. *Req refers to the Requirement score.

Type	AVG Quality	SDTD Quality	Avg Dist	SDTD Dist
1	2.17	1.33	2.47	1.06
2	2.17	0.98	3.07	0.8
3	1.17	0.76	4	1.2

Table 10: Results of the qualitative evaluation.

7 Conclusion & Future Work

In conclusion, we have a system that manages to generate very diverse floor-plans to the detriment of achieving consistent quality. We believe this method can be used by a architect during the early stages of residential projects to generate layout ideas that could be further explored. The method is simple which makes it easy to implement in most architectural software which also makes it easily customizable and adaptable to specific project requirements while still producing a good amount of diversity.

We believe that adding more restrictions to the method could improve its consistency in quality while not reducing the diversity too much. For example adding an entryway room connected to the hallway with an additional parameter to the fitness function would help it stay at the border of the layout, which could be also applied to other rooms that we might want to stay at the border (Garages, Bedrooms or others) would greatly increase the correctness of the results. Alternatively the method could have an entryway as mentioned but no hallway during the generation, and like some previous methods, add the hallway in a post-processing step if needed. Another improvement could be the addition of an adjacency graph that would work the same way as the connectivity graph but represent only suggestions to the position of rooms. This could greatly increase the logic in the layouts without reducing the diversity.

Additionally, we noticed that our method would not allow the creation of a layout similar to the image reference we got for the house type 3 (see Figure 24 in the appendix), which features an L-shaped house with a garden. This could be potentially fixed by adding the garden to the connectivity graph, or by removing the bounding box from the fitness function.

To improve the speed of our method we believe that by placing the rooms together at the initialization and not allowing the remove cell move type to create holes, would allow us to avoid our main bottleneck all together and would make the system a few times faster. This placement could be done by placing the connectivity graph as a planar graph. Furthermore re-implementing the system with C++, and avoiding Unity would probably also make the system faster. Then there is always the possibility of changing the grid size to increase performance but also reducing complexity to the results.

Further improvements can always be made tweaking the fitness weights, cooling rate and stopping condition. And more features could be added. Having different room types be affected differently by the fitness function (for example having stick-on rooms being heavily pushed to their respective rooms or the entryway as mentioned above).

Additional future work could be the accommodation of the method to work in a 3D grid. This way terrain height could be taken into account, and the addition of multiple floors. And the method could be expanded to further assist the architect throughout the duration of the project.

Finally, further work could be done to both metrics to more accurately represent our perception of the quality and distance. This could be coupled with a user study as a mean of verification and as a way of understanding what aspects of a floor layout we look for and more accurately design these measures.

References

- [1] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.
- [2] Mark Bruls, Cornelis Huizing, and Jarke J. van Wijk. “Squarified Treemaps”. In: *VisSym*. 2000.
- [3] Daniel Camozzato. “A method for growth-based procedural floor plan generation”. PhD thesis. Pontificia Universidade Catolica do Rio Grande do Sul, 2015.
- [4] *City Engine*. URL: https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview?rmedium=www_esri_com_EtoF&rsource=/en-us/arcgis/products/esri-cityengine/overview. (accessed: 14.02.2021).
- [5] Subhajit Das, Colin Day, John Hauck, John Haymaker, and Diana Davis. “Space plan generator: Rapid generation & evaluation of floor plan design options to inform decision making”. In: *ACADIA* (2016), pp. 106–115.

- [6] Adam Doulgerakis. “Genetic programming+ unfolding embryology in automated layout planning”. PhD thesis. UCL (University College London), 2007.
- [7] Prosenjit Bose Evan Hahn and A.D. Whitehead. “Persistent realtime building interior generation”. In: *SIGGRAPH symposium on Videogames* (2006). DOI: <https://doi.org/10.1145/1183316.1183342>.
- [8] *Grasshopper3D*. URL: <https://www.grasshopper3d.com/>. (accessed: 15.02.2021).
- [9] Zifeng Guo and Biao Li. “Evolutionary approach for spatial architecture layout design enhanced by an agent-based topology finding system”. In: *Frontiers of Architectural Research* 6 (2016), pp. 53–62. DOI: <https://doi.org/10.1016/j.foar.2016.11.003>.
- [10] M.E. J.F.BrotchieB.C.E. D.Eng.M.P.T.LinzeyB.E. “A model for integrated building design”. In: *Building Science* 6.3 (1971), pp. 89–96. DOI: [https://doi.org/10.1016/0007-3628\(71\)90020-X](https://doi.org/10.1016/0007-3628(71)90020-X).
- [11] K. Kozminski and E. Kinnen. “An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits”. In: *21st Design Automation Conference Proceedings*. 1984, pp. 655–656.
- [12] Robin S Liggett. “Automated facilities layout: past, present and future”. In: *Automation in Construction* 9.2 (2000), pp. 197–215. ISSN: 0926-5805. DOI: [https://doi.org/10.1016/S0926-5805\(99\)00005-9](https://doi.org/10.1016/S0926-5805(99)00005-9). URL: <https://www.sciencedirect.com/science/article/pii/S0926580599000059>.
- [13] Ricardo Lopes, Tim Tutenel, Ruben M Smelik, Klaas Jan De Kraker, and Rafael Bidarra. “A constrained growth method for procedural floor plan generation”. In: *Proc. 11th Int. Conf. Intell. Games Simul.* 2010, pp. 13–20.
- [14] Fernando Marson and Soraia Raupp Musse. “Automatic real-time generation of floor plans based on squarified treemaps algorithm”. In: *International Journal of Computer Games Technology* 2010 (2010).
- [15] Jess Martin. “Procedural house generation: A method for dynamically generating floor plans”. In: *Proceedings of the Symposium on Interactive 3D Graphics and Games*. 2006, pp. 1–2.
- [16] Paul Merrell, Eric Schkufza, and Vladlen Koltun. “Computer-generated residential building layouts”. In: *ACM SIGGRAPH Asia 2010 papers*. 2010, pp. 1–12.
- [17] Jeremy Michalek, Ruchi Choudhary, and Panos Papalambros. “Architectural layout design optimization”. In: *Engineering Optimization* 34.5 (2002), pp. 461–484.
- [18] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical aspects of local search*. Springer Science & Business Media, 2007.

- [19] Ralph H J M Otten. “Graphs in floor-plan design”. In: *International Journal of Circuit Theory and Applications* 16.4 (1988), pp. 391–410. ISSN: 1097-007X. DOI: 10.1002/cta.4490160405.
- [20] Marc Pirlot. “General local search methods”. In: *European Journal of Operational Research* 92.3 (1996), pp. 493–511. ISSN: 0377-2217. DOI: [https://doi.org/10.1016/0377-2217\(96\)00007-0](https://doi.org/10.1016/0377-2217(96)00007-0). URL: <https://www.sciencedirect.com/science/article/pii/0377221796000070>.
- [21] *Revit Architecture*. URL: <https://www.autodesk.com/products/revit/architecture>. (accessed: 15.02.2021).
- [22] Eugénio Rodrigues, Adélio Rodrigues Gaspar, and Álvaro Gomes. “An approach to the multi-level space allocation problem in architecture using a hybrid evolutionary technique”. In: *Automation in Construction* 35 (2013), pp. 482–498.
- [23] Peter J M Van Laarhoven and Emile H L Aarts. “Simulated annealing”. In: *Simulated Annealing: Theory and Applications*. Springer, 1987, pp. 7–15.

Appendix

In this first figures (see Figures 22, 23 and 24) you can see the images of the house we used as templates for our house types.

In the following figures you will find the our choice of the best 6 floor-plans for each experiment.

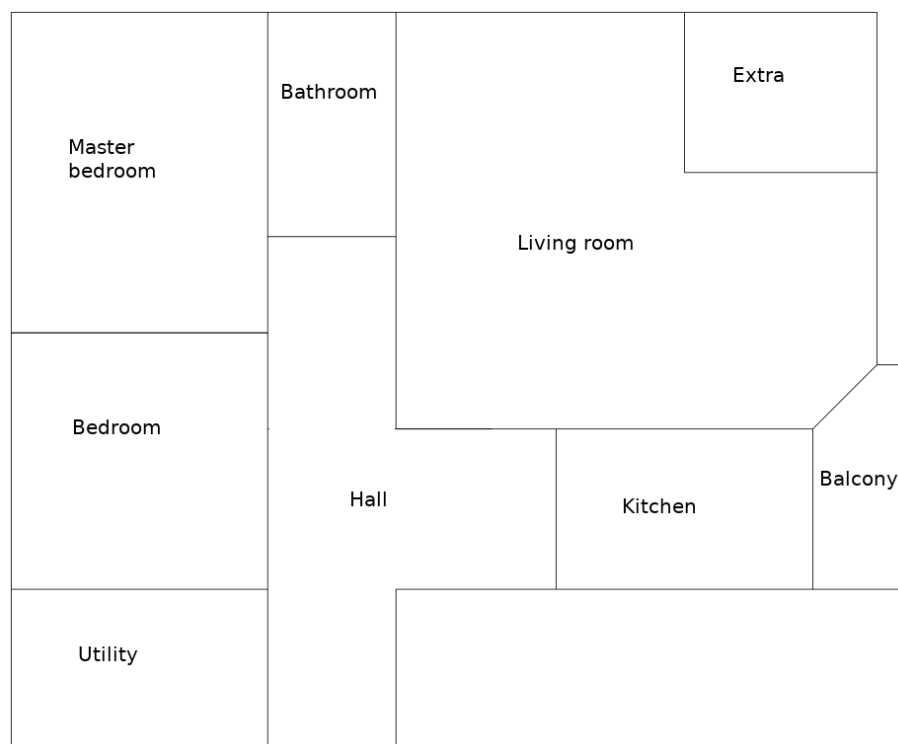


Figure 19: Floor-plan used as a template for the house type 1.

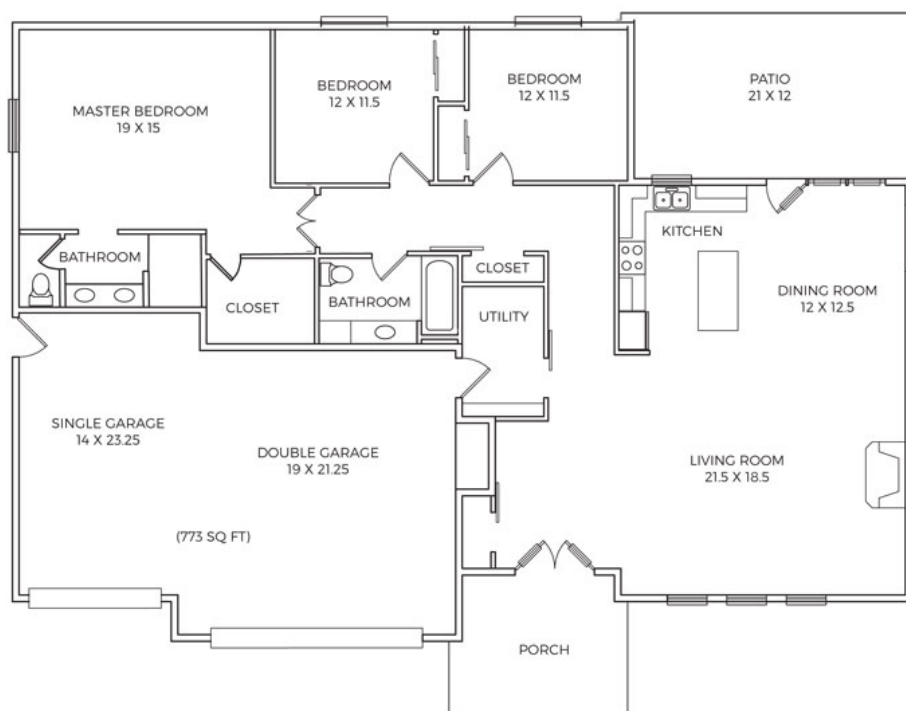


Figure 20: Floor-plan used as a template for the house type 2.

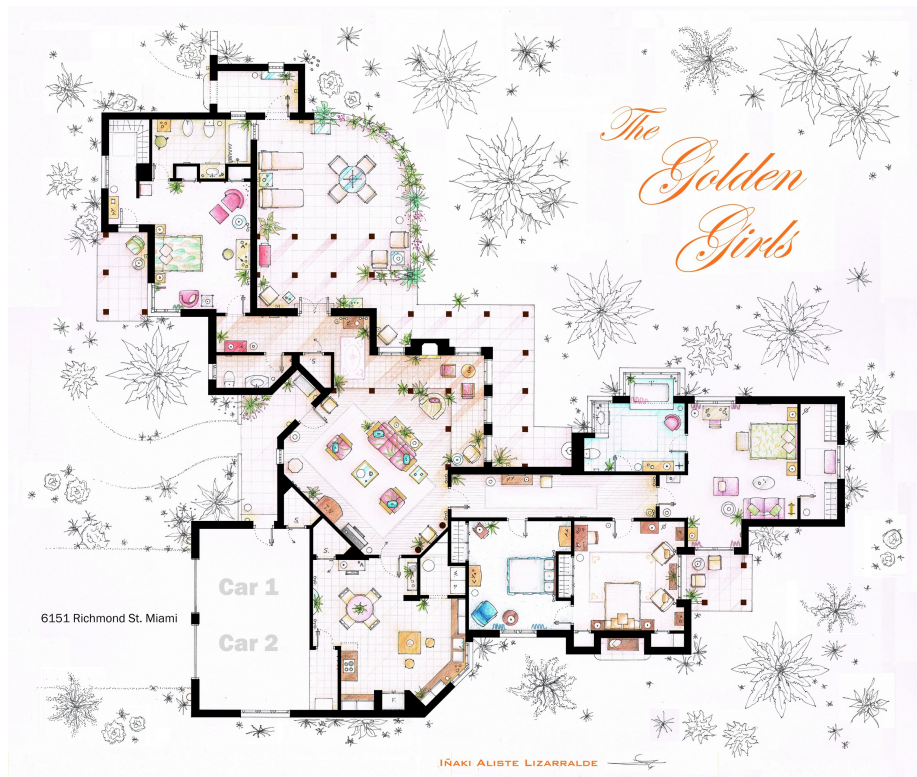
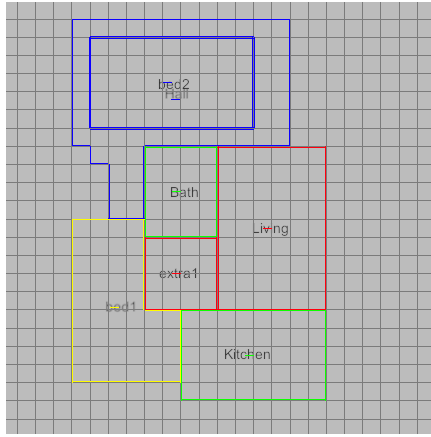
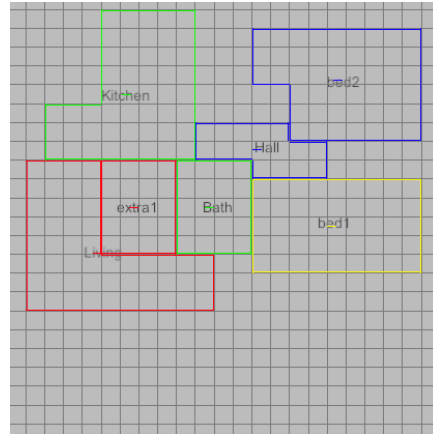


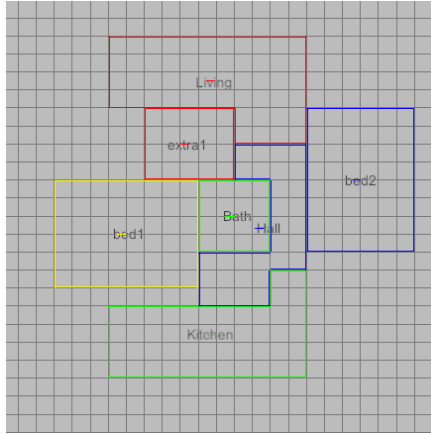
Figure 21: Floor-plan used as a template for the house type 3.



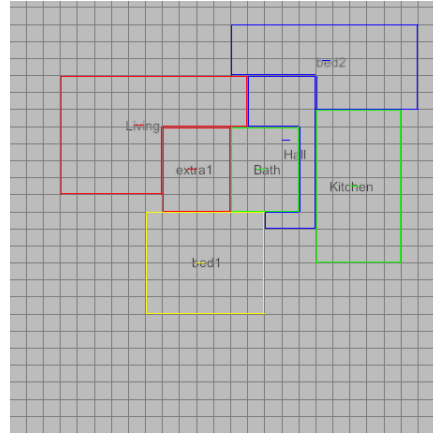
(a) Fitness Score: 313



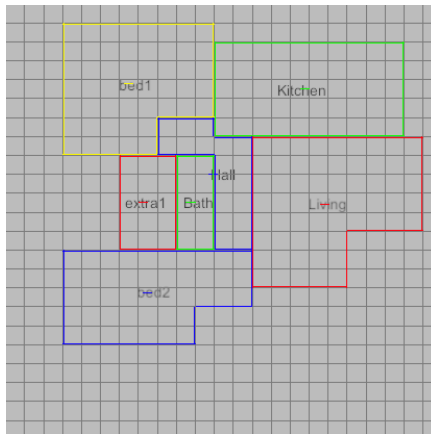
(b) Fitness Score: 447



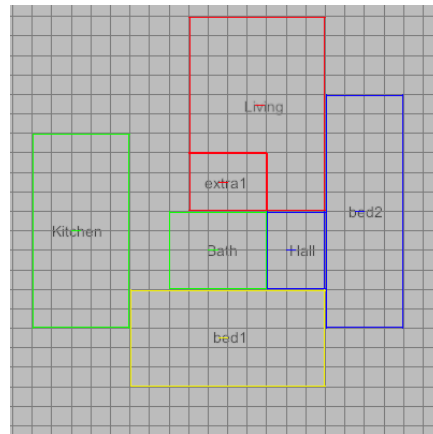
(c) Fitness Score: 466



(d) Fitness Score: 485

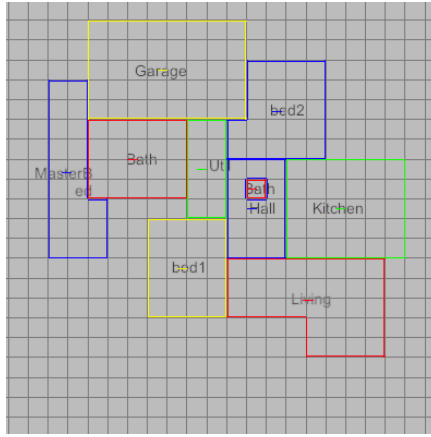


(e) Fitness Score: 518

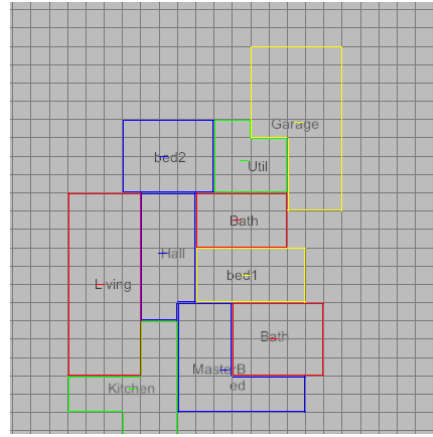


(f) Fitness Score: 483

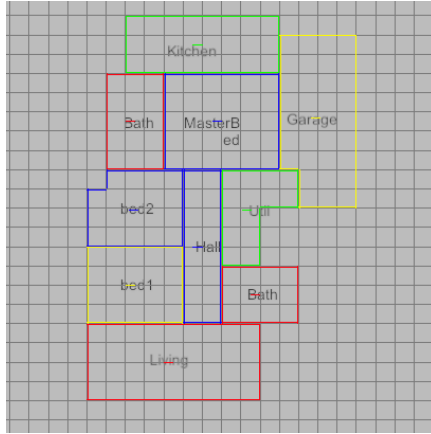
Figure 22: Result for the generation of the house type 1.



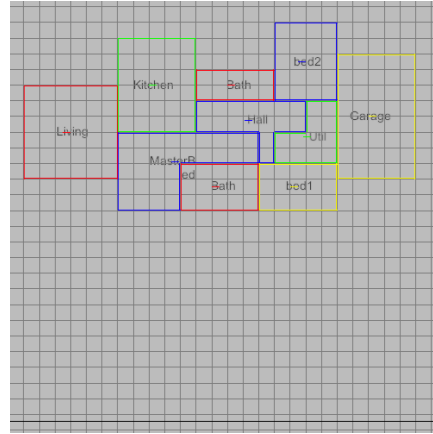
(a) Fitness Score: 680



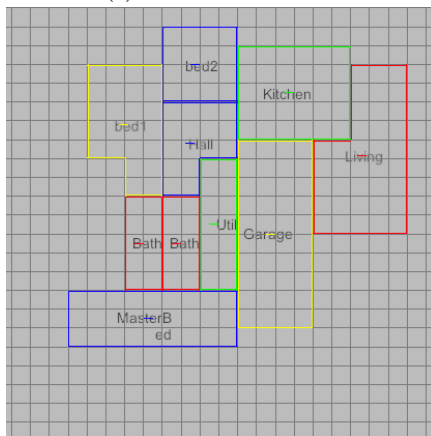
(b) Fitness Score: 714



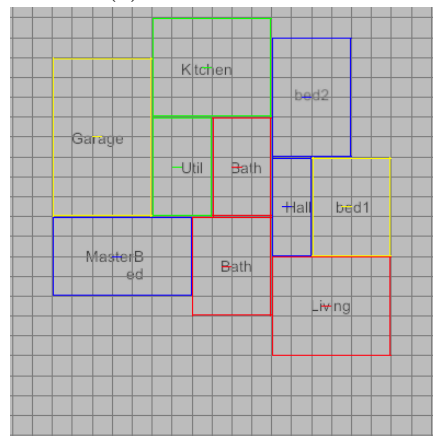
(c) Fitness Score: 720



(d) Fitness Score: 737

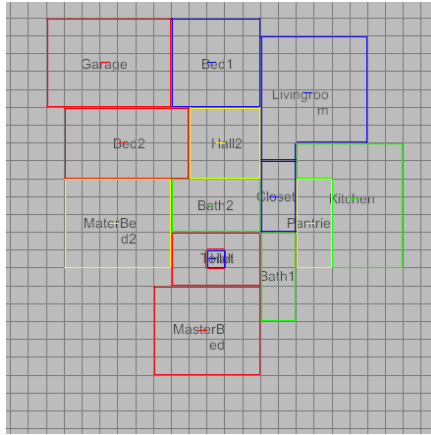


(e) Fitness Score: 763

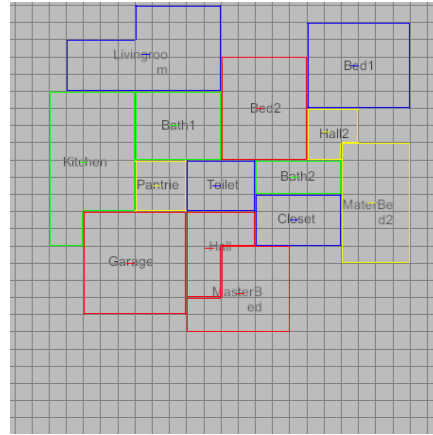


(f) Fitness Score: 793

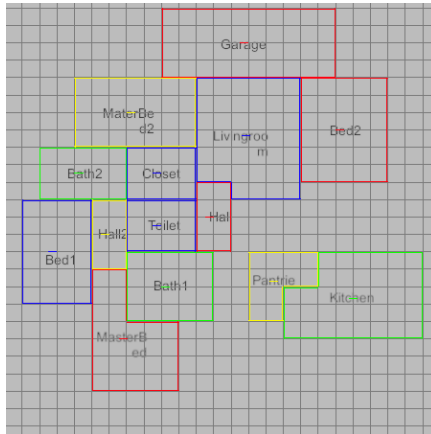
Figure 23: Result for the generation of the house type 2.



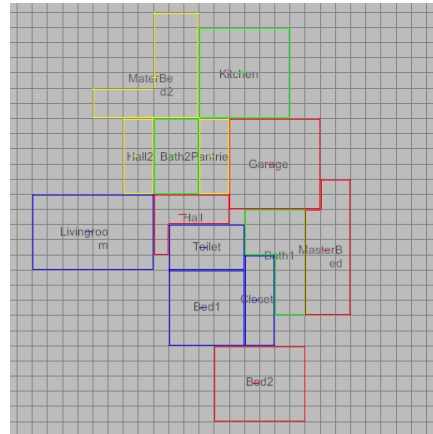
(a) Fitness Score: 1012



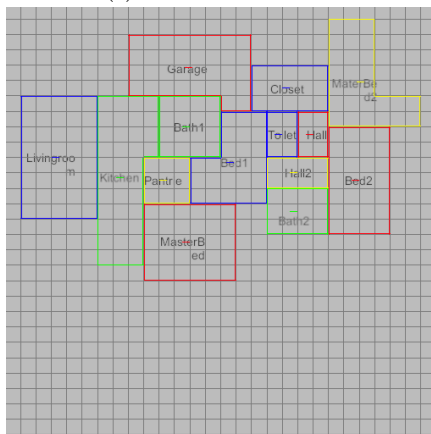
(b) Fitness Score: 1101



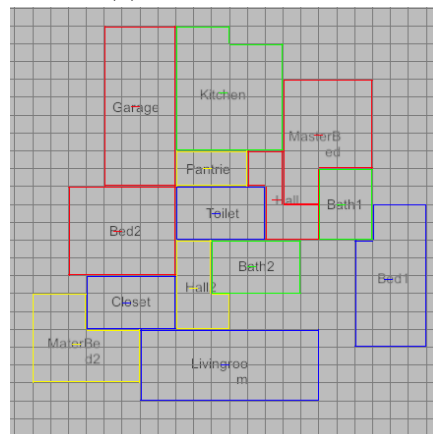
(c) Fitness Score: 804



(d) Fitness Score: 849

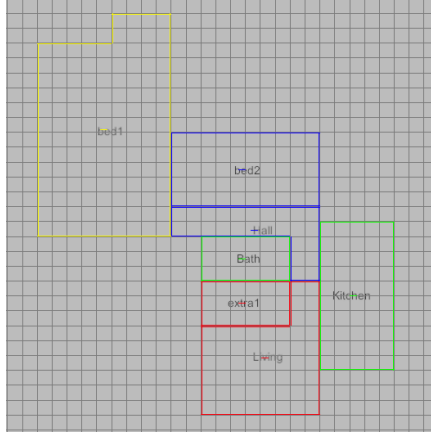


(e) Fitness Score: 901

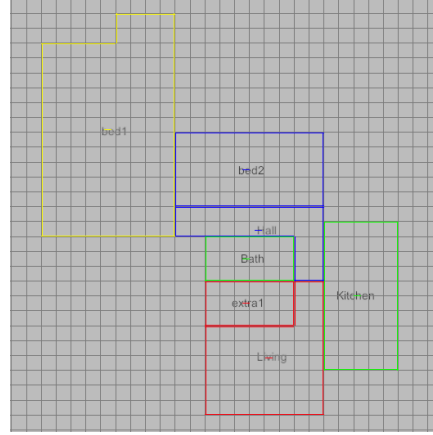


(f) Fitness Score: 923

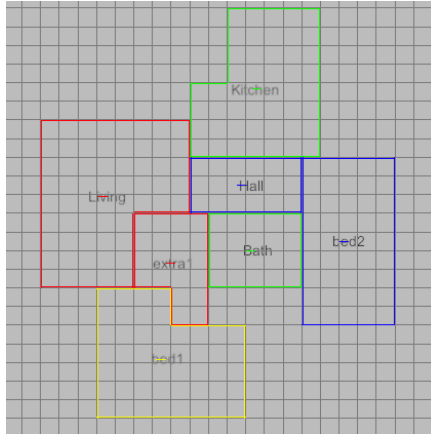
Figure 24: Result for the generation of the house type 3.



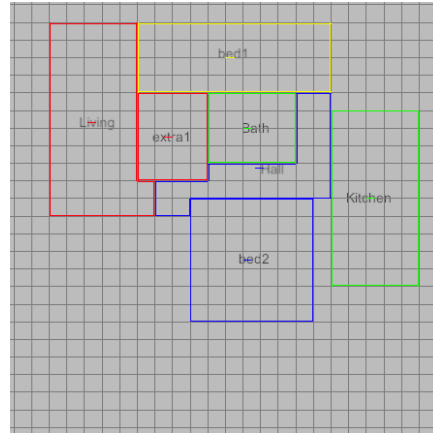
(a) Fitness Score: 147



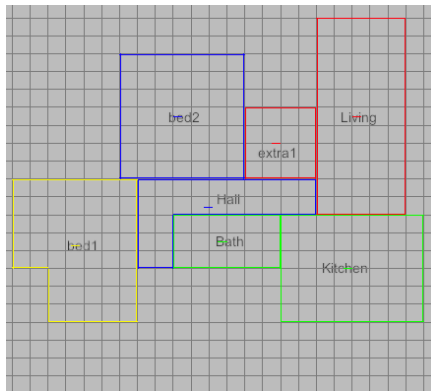
(b) Fitness Score: 147



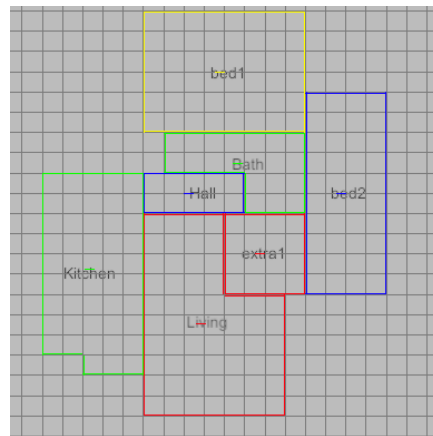
(c) Fitness Score: 439



(d) Fitness Score: 470

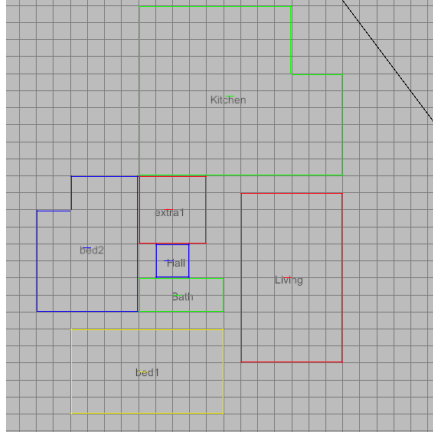


(e) Fitness Score: 471

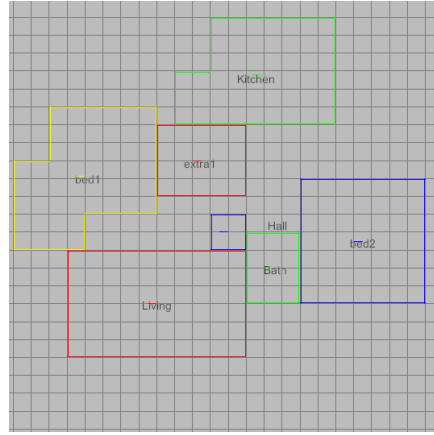


(f) Fitness Score: 539

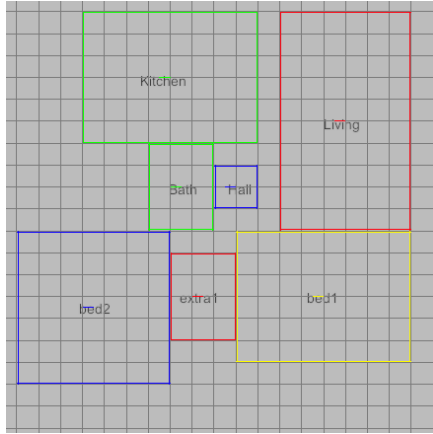
Figure 25: Result for the generation of the house type 1 without the Bounding parameter in the fitness function.



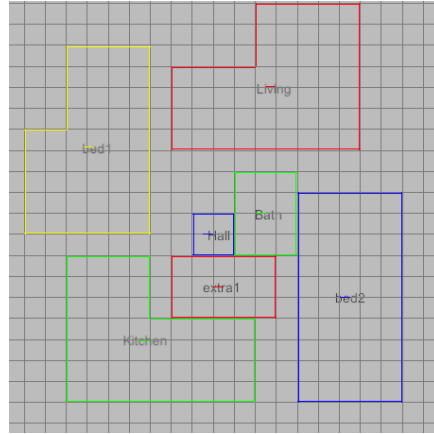
(a) Fitness Score: -222



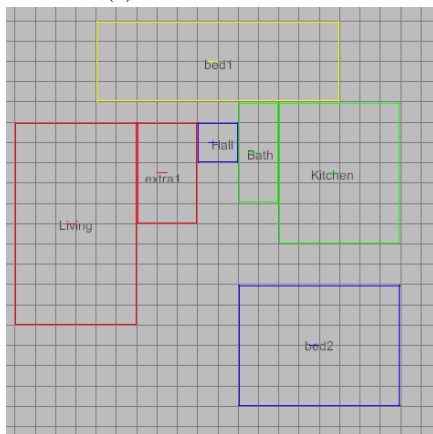
(b) Fitness Score: -94



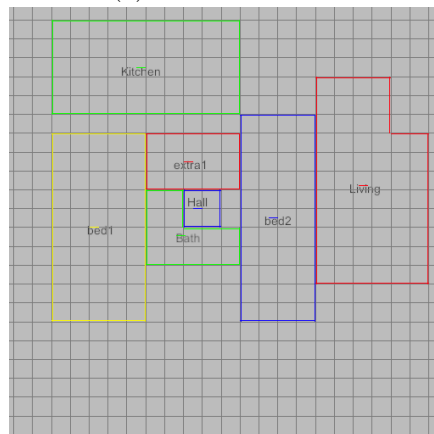
(c) Fitness Score: 130



(d) Fitness Score: 40

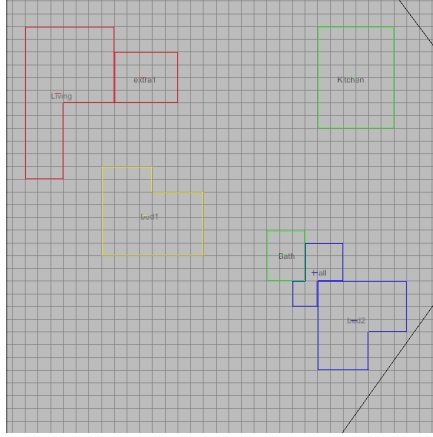


(e) Fitness Score: 81

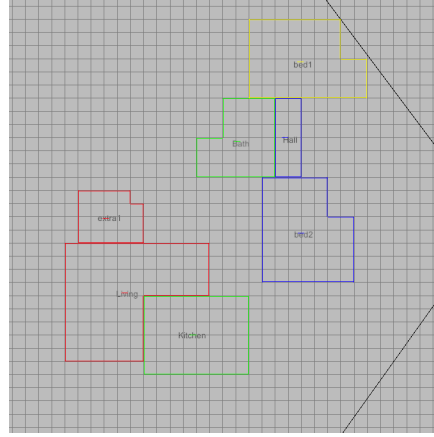


(f) Fitness Score: 98

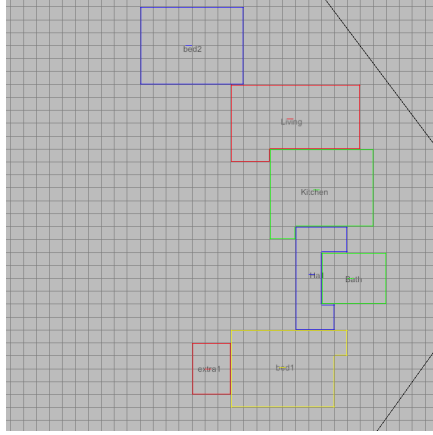
Figure 26: Result for the generation of the house type 1 without the Connections parameter in the fitness function.



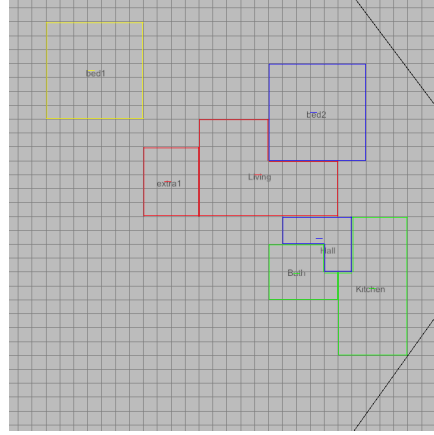
(a) Fitness Score: 108



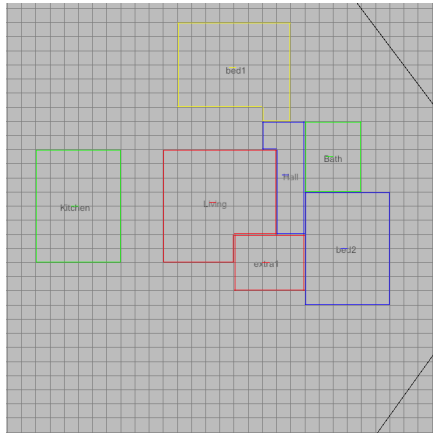
(b) Fitness Score: 206



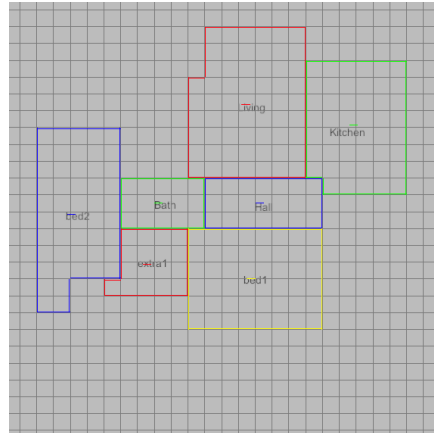
(c) Fitness Score: 322



(d) Fitness Score: 346

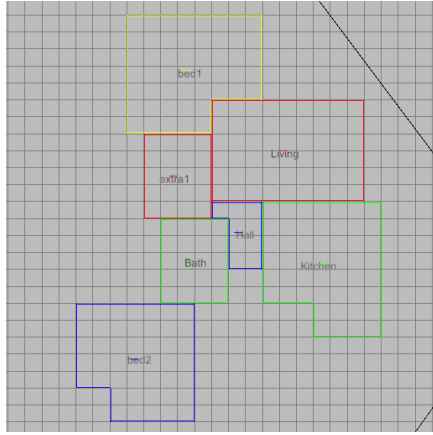


(e) Fitness Score: 424

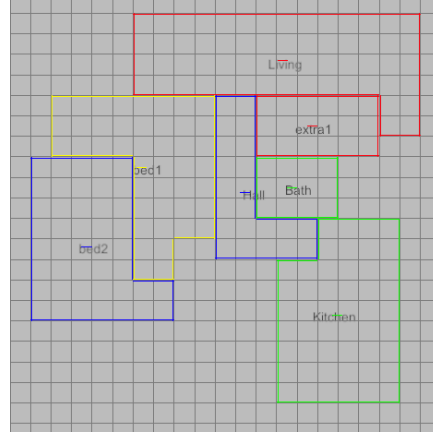


(f) Fitness Score: 441

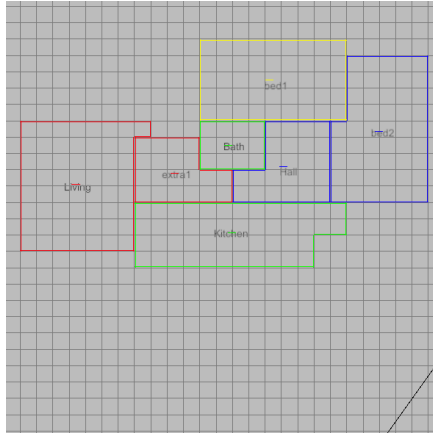
Figure 27: Result for the generation of the house type 1 without the Distance parameter in the fitness function.



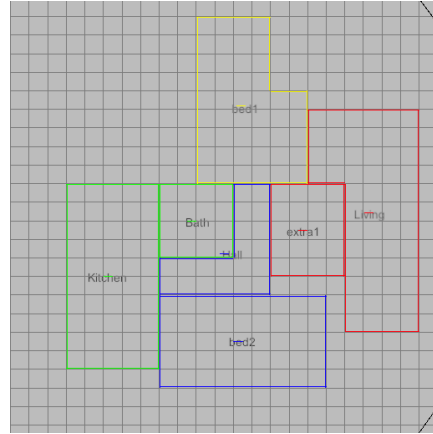
(a) Fitness Score: 322



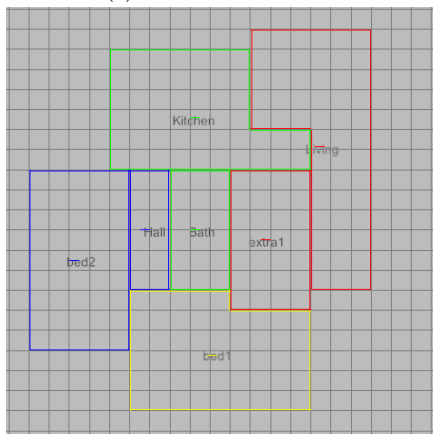
(b) Fitness Score: 347



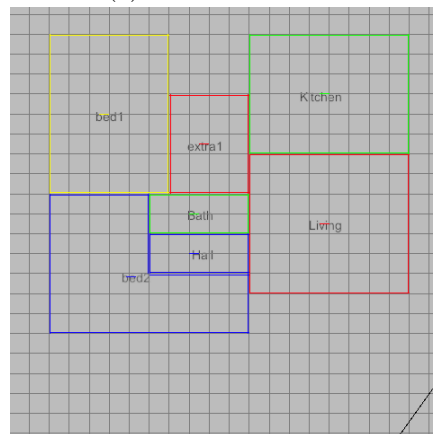
(c) Fitness Score: 386



(d) Fitness Score: 459

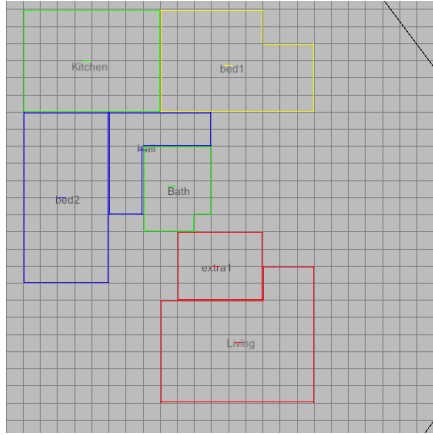


(e) Fitness Score: 555

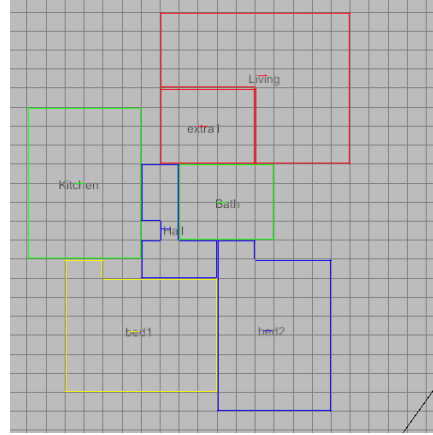


(f) Fitness Score: 595

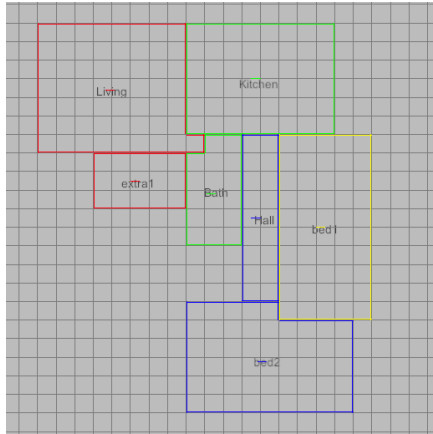
Figure 28: Result for the generation of the house type 1 without the Fill parameter in the fitness function.



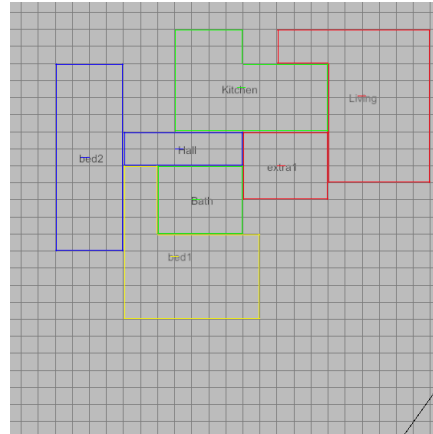
(a) Fitness Score: 266



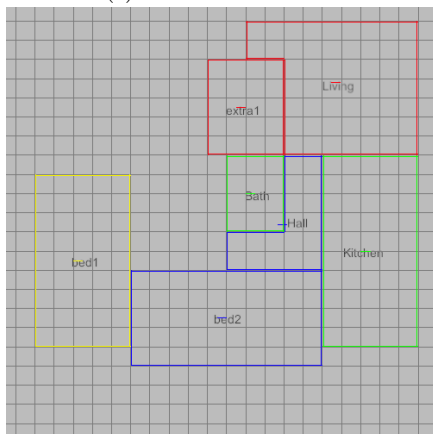
(b) Fitness Score: 392



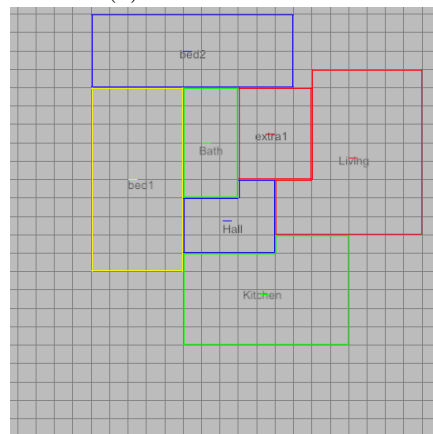
(c) Fitness Score: 443



(d) Fitness Score: 478

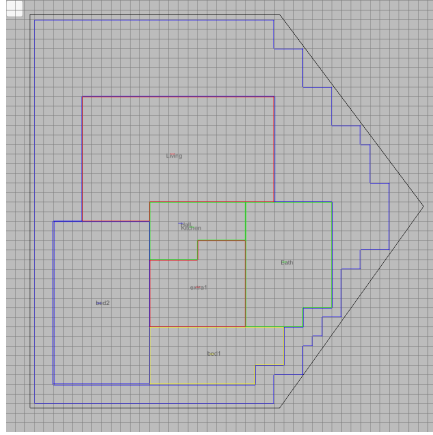


(e) Fitness Score: 459

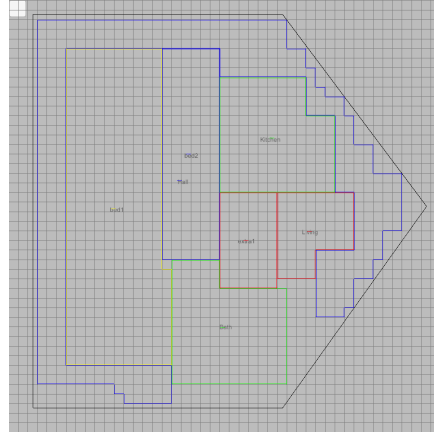


(f) Fitness Score: 580

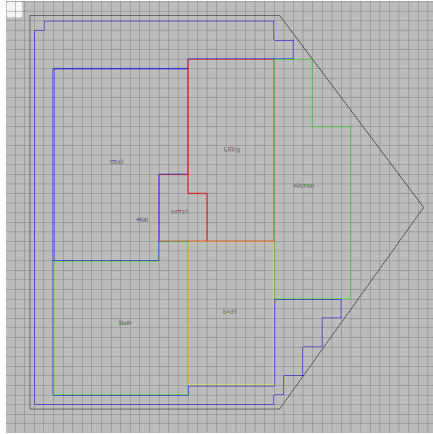
Figure 29: Result for the generation of the house type 1 without the One Wide Holes parameter in the fitness function.



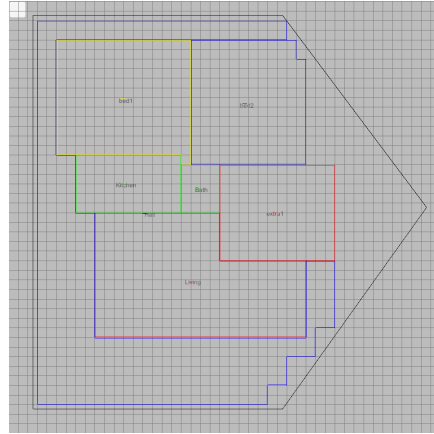
(a) Fitness Score: -251



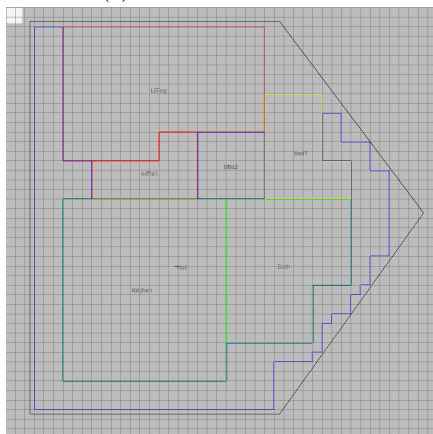
(b) Fitness Score: -446



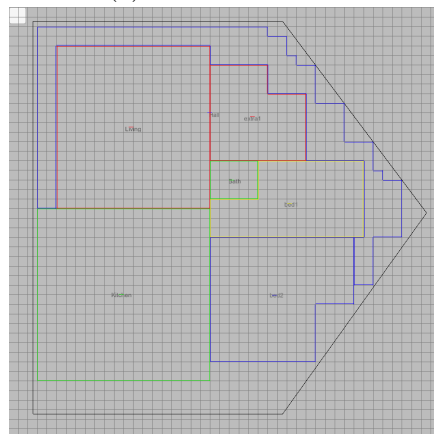
(c) Fitness Score: -634



(d) Fitness Score: -82

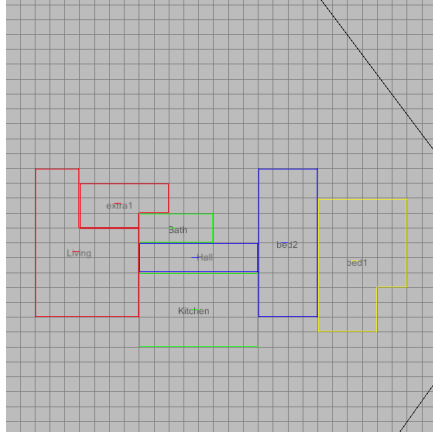


(e) Fitness Score: -837

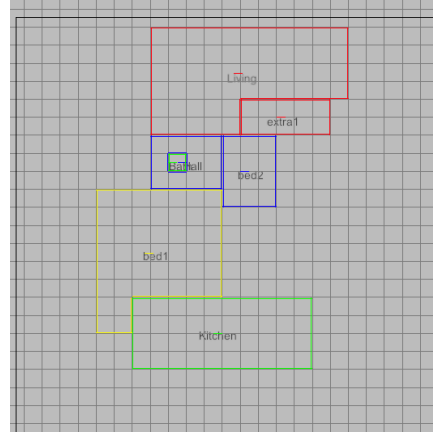


(f) Fitness Score: -961

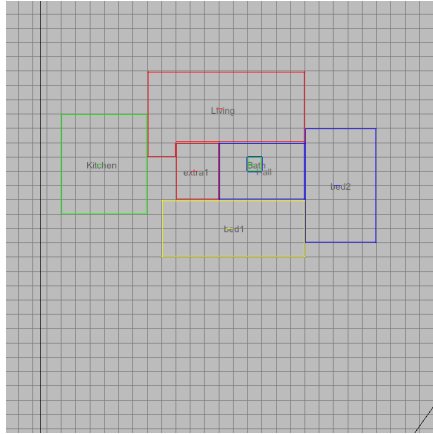
Figure 30: Result for the generation of the house type 1 without Maximum Size parameter in the fitness function.



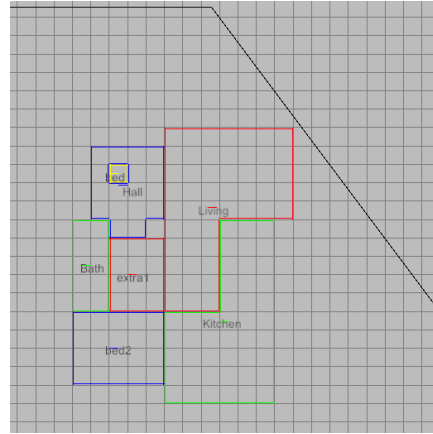
(a) Fitness Score: 350



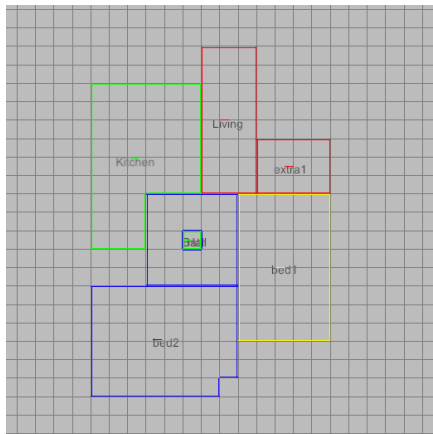
(b) Fitness Score: 365



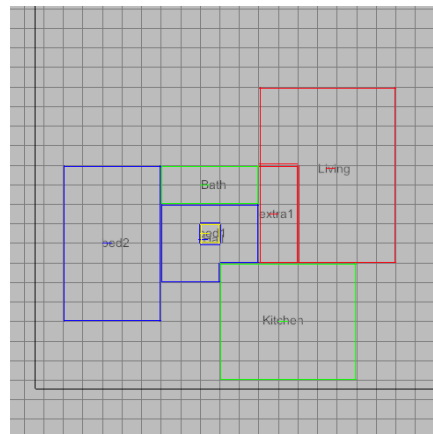
(c) Fitness Score: 398



(d) Fitness Score: 429

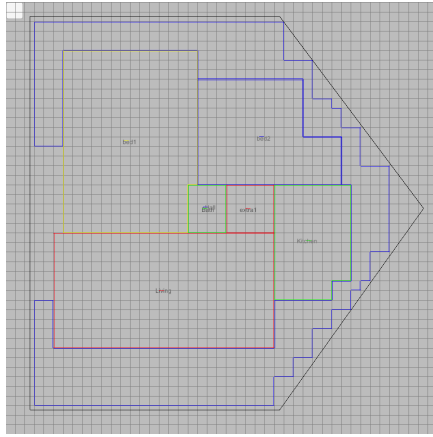


(e) Fitness Score: 461

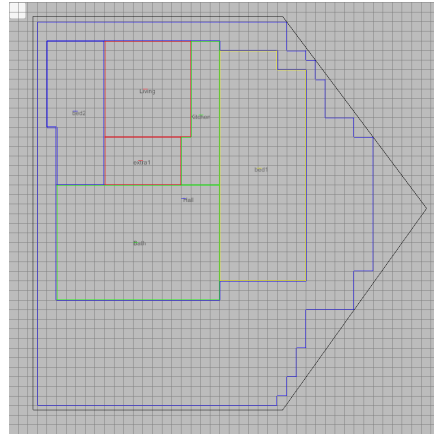


(f) Fitness Score: 499

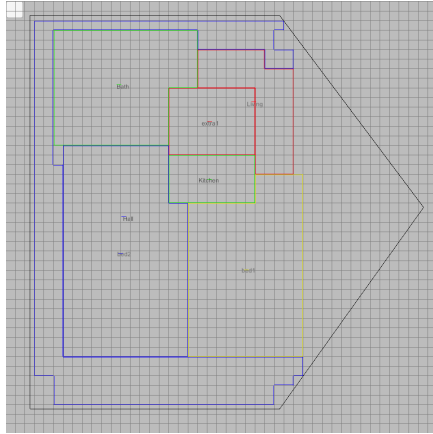
Figure 31: Result for the generation of the house type 1 without Minimum Size parameter in the fitness function.



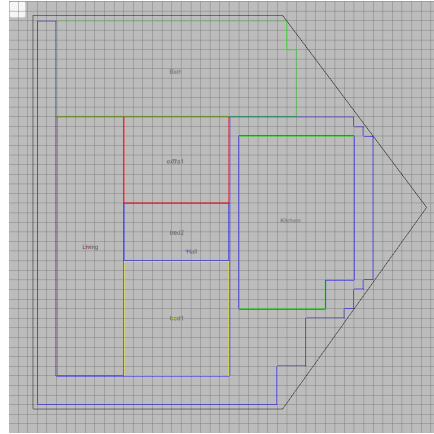
(a) Fitness Score: -205



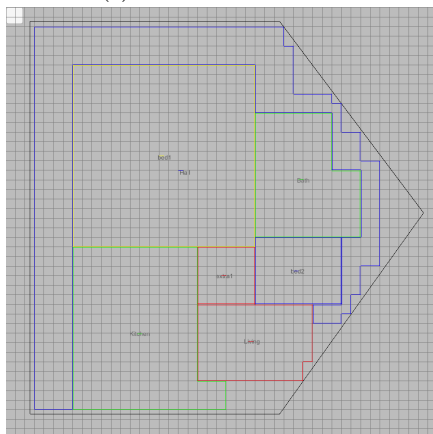
(b) Fitness Score: -57



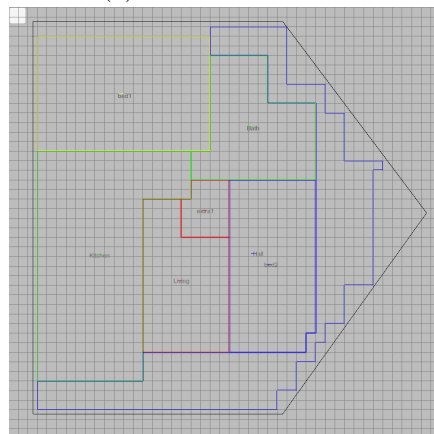
(c) Fitness Score: -574



(d) Fitness Score: -655

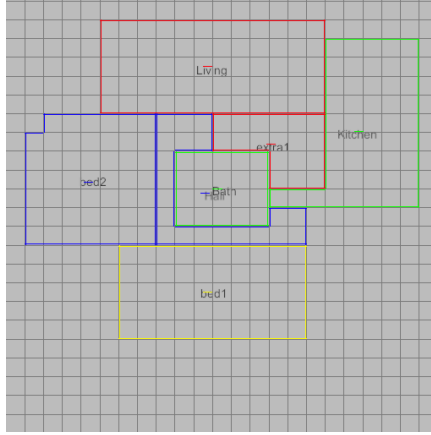


(e) Fitness Score: -748

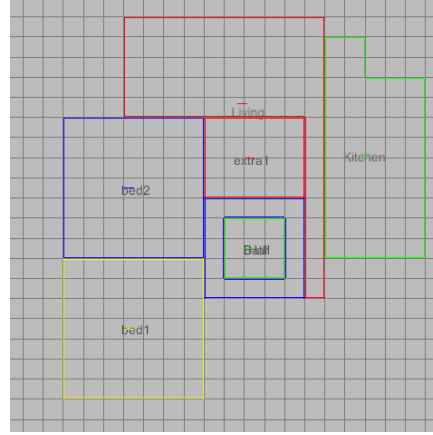


(f) Fitness Score: -769

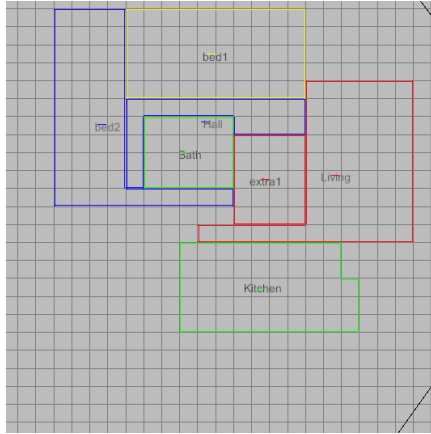
Figure 32: Result for the generation of the house type 1 without Size parameters in the fitness function.



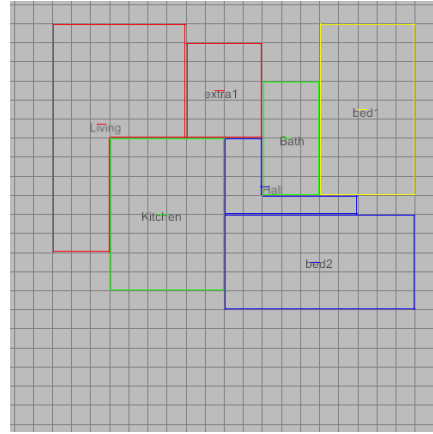
(a) Fitness Score: 635



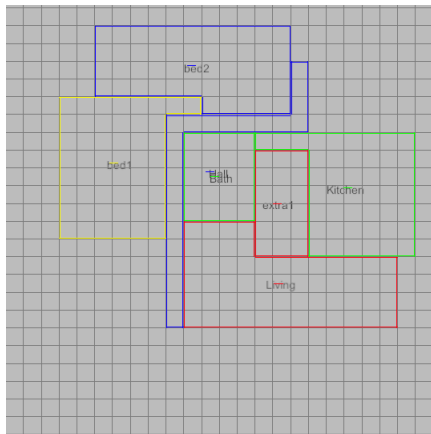
(b) Fitness Score: 368



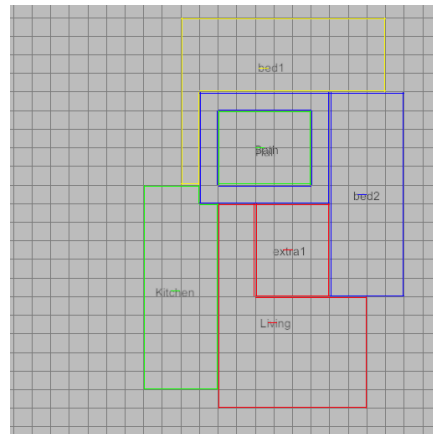
(c) Fitness Score: 653



(d) Fitness Score: 662

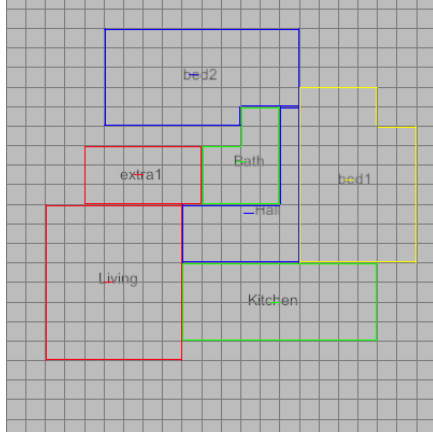


(e) Fitness Score: 693

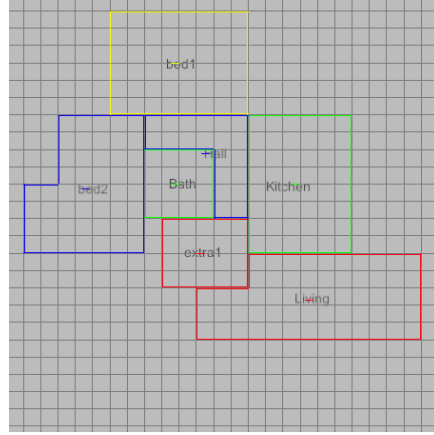


(f) Fitness Score: 763

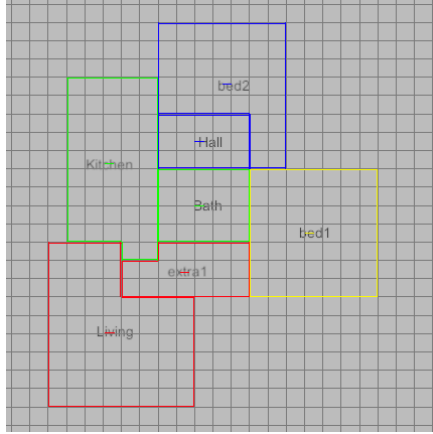
Figure 33: Result for the generation of the house type 1 without One Wide parameter in the fitness function.



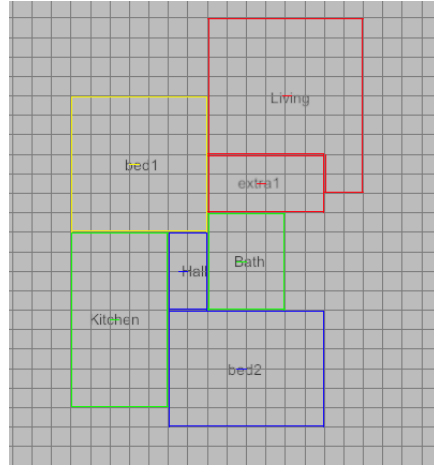
(a) Fitness Score: 452



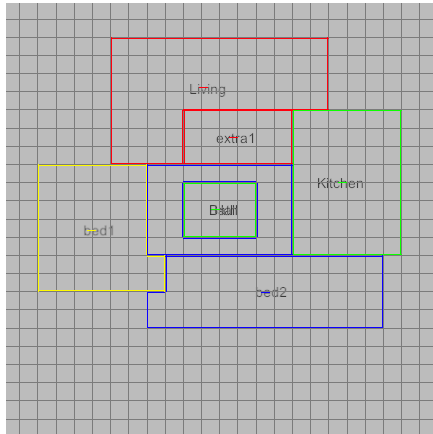
(b) Fitness Score: 488



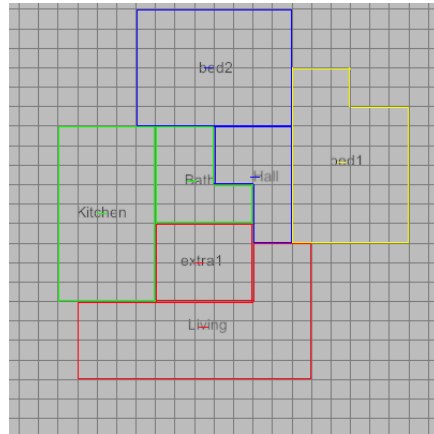
(c) Fitness Score: 491



(d) Fitness Score: 518

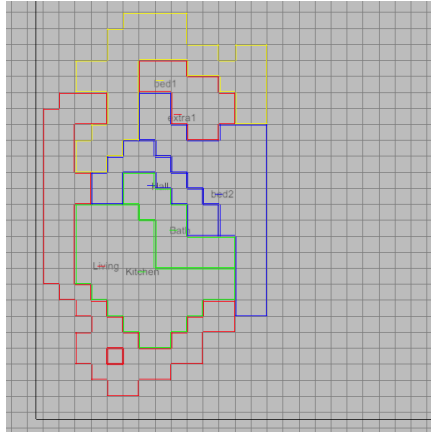


(e) Fitness Score: 541

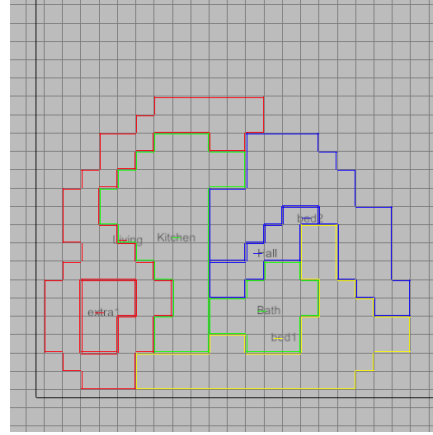


(f) Fitness Score: 584

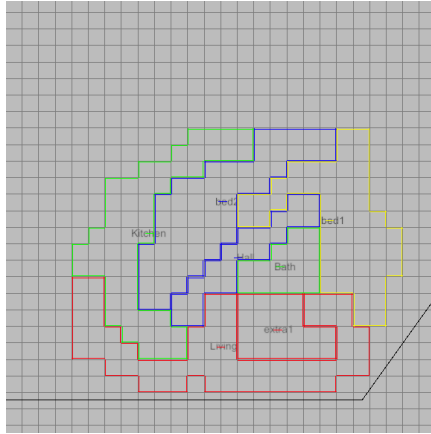
Figure 34: Result for the generation of the house type 1 without the Ratio parameter in the fitness function. 43



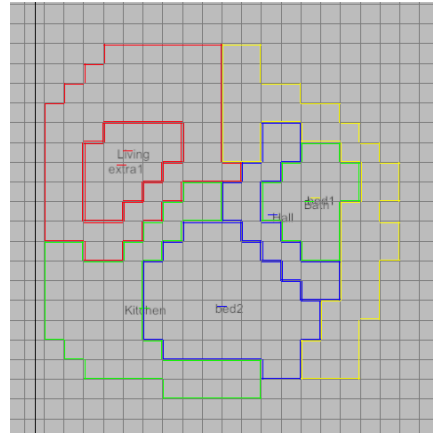
(a) Fitness Score: 1050



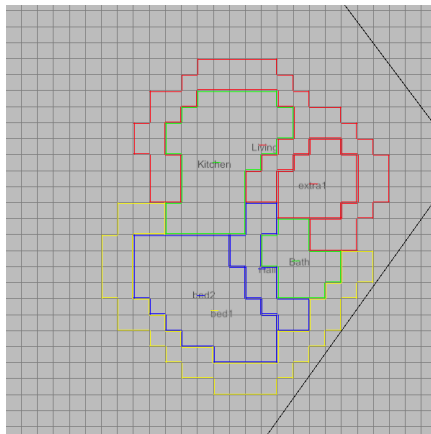
(b) Fitness Score: 1065



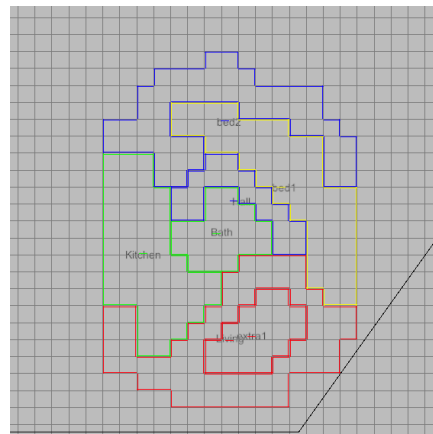
(c) Fitness Score: 1097



(d) Fitness Score: 1116



(e) Fitness Score: 1120



(f) Fitness Score: 1125

Figure 35: Result for the generation of the house type 1 without the Rectangularity parameter in the fitness function.