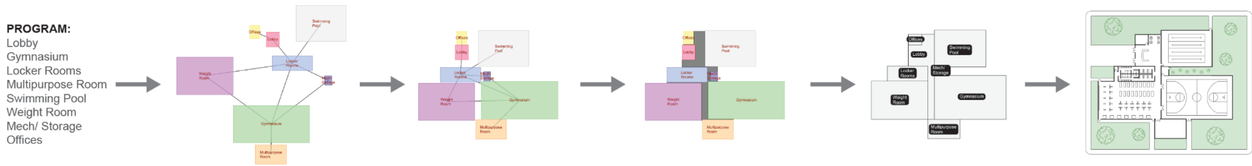

Text2Plan: A Generative Workflow for Conceptual Floor Plans though Visualized Converging Data Graphs

By Greg Budhijanto

For architects and designers, a new project typically begins with considering its program of spaces. A common exercise used to ideate the logic of the floor plan is to use bubble diagrams to show various adjacencies. However, generating multiple schemes to find a winning idea can be tedious and budget-exhausting. To streamline this process, this project presents a graph convergence algorithm that converts the program input into data objects and returns a schematic plan layout complete with space adjacencies. With this tool, architects and designers can generate multiple conceptual floor plans quickly, freeing up time that would otherwise be spent manually drawing diagrams. This allows architects more time to make informed design decisions after reviewing a holistic picture of possible design outcomes.

The algorithm begins with a multiline text input of each room and their properties. Figure 1 can be seen for a sample input where in each line, properties such as room name, area, minimum dimensions, and their adjacencies. Each line is converted to a room_node data object with the final property, adjacencies, as a set of other room_node objects. A data diagram of a sample room node can be seen in Figure 2. This input is the framework of the graph's data structure and sets the logic of laying out the program.

```
                                                                                  {0}
 0 room_name, area(ft^2), max_dim(ft), min_dim(ft), height, color, adjacency_list
 1 Entry, 50, None, 6, 12, e6194b, [Living Room]
 2 Living Room, 300, None, 16, 15, 3cb44b, [Entry, Kitchen, Dining Room, Hallway, Powder Room]
 3 Kitchen, 150, None, 12, 15, 4363d8, [Living Room, Dining Room, Mud Room]
 4 Dining Room, 200, None, 12, 15, f58231, [Living Room, Kitchen]
 5 Mud Room, 50, None, 6, 12, a9a9a9, [Kitchen, Garage, Laundry Room]
 6 Garage, 400, None, 20, 12, 911eb4, [Mud Room]
 7 Hallway, 75, None, 3, 12, 000075, [Living Room, Bedroom 1, Bedroom 2, Primary Bedroom, Bathroom 1]
 8 Powder Room, 20, None, 3, 12, ffe119, [Living Room]
 9 Bedroom 1, 150, None, 12, 12, 9a6324, [Hallway, Bathroom 1]
10 Bedroom 2, 150, None, 12, 12, 469990, [Hallway, Bathroom 1]
11 Bathroom 1, 75, None, 6, 12, 42d4f4, [Bedroom 1, Bedroom 2, Hallway]
12 Primary Bedroom, 225, None, 14, 15, f032e6, [Hallway, Primary Bathroom]
13 Primary Bathroom, 100, None, 8, 12, bfef45, [Primary Bedroom]
14 Laundry Room, 50, None, 6, 12, 800000, [Mud Room]
```

Figure 1. Input Text Block

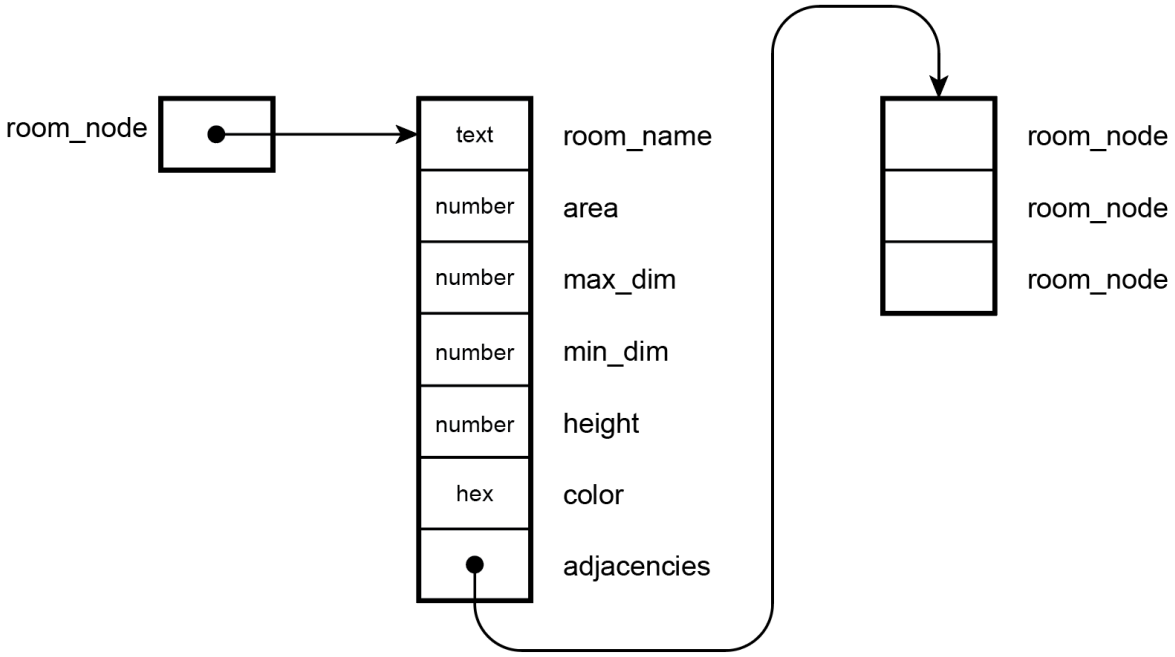

Figure 2. Representation of Room Node Data Object

Graph data structures can be summarized as a series of nodes that are connected by edges to form a network. In architectural terms, these nodes can represent different programming elements, such as rooms or spaces, and edges can represent required adjacencies. Once the input text block is processed, the underlying graph data structure is laid out with a brute force recursive back-tracking algorithm. The basic steps of this algorithm are as follows and Figure 3 can be seen for visual representation.

1. Determine initial room
2. Until all room nodes are placed:
    a. Determine next room based on most adjacencies
    b. Place next room based on Gaussian Distribution
        i. Not intersect?
            1. Call recursive function
        ii. Edges intersect?
            1. Different location
    c. No success?
        i. Remove previously placed
        ii. Call recursive

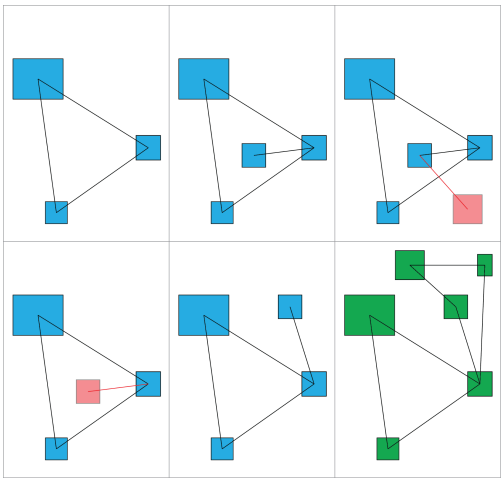

Figure 3. Graph Layout Algorithm

The algorithm utilizes the area and minimum dimension property of the room_node object to randomly generate a rectangle representative of the size of the room. Rooms are placed one-by-one with the next room determined as the room with the most adjacencies to what is placed. From a gaussian distribution, the room is placed closer to its adjacencies and further from its non-adjacencies to ensure a clear graph. The room is placed and a line connecting it to its adjacencies is set. This line represents an edge in the graph and is stored in a new room_node property, edges. If the edge intersects non-adjacent rooms or other already placed edges, the room is placed in a different location from the gaussian distribution. If after 10 attempts with no success, the previously placed is removed and calls the recursive function again. The function completes once all room nodes and edges have been successfully placed. Figure 4 depicts several generated graph layouts for the sample input in Figure 1.

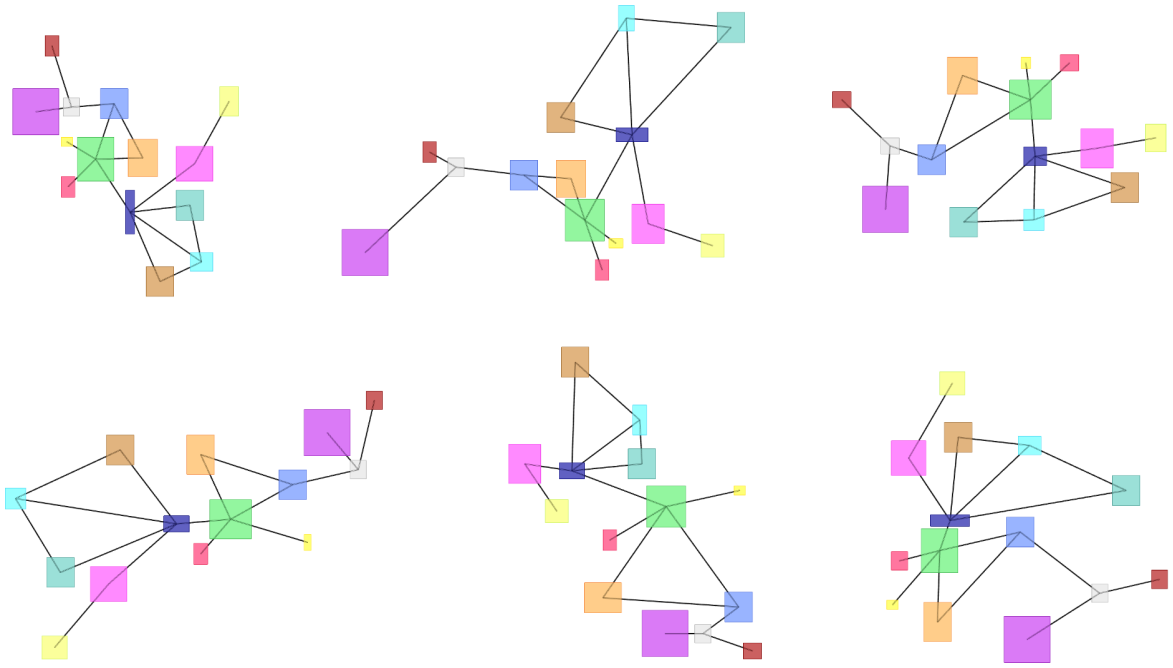

Figure 4. Generated Graph Layouts for Sample Input

Once all graph nodes and edges are placed, a force directed graph algorithm can be used to converge all room spaces based. Over iterations, this algorithm uses attraction and repulsion vectors to unite nodes that are apart and push out nodes that are too close or overlap. Once the nodes are at a point of limited movement, the algorithm completes with a schematic layout of the project's program. The basic steps of this algorithm are as follows and Figure 5 can be seen for visual representation.

1. For each iteration:
    a. For each room:
        i. Calculate attraction vector
        ii. Move rectangle per factor of attraction vector
    b. For each room:
        i. Calculate repulsion vector
        ii. Move rectangle per factor of repulsion vector
    c. Reduce factor

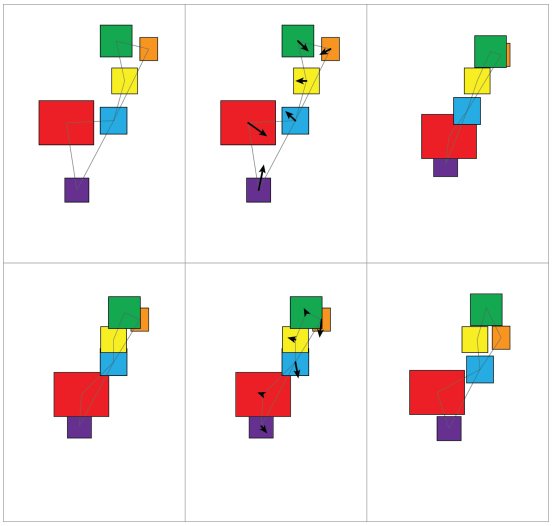

Figure 5. Graph Convergence

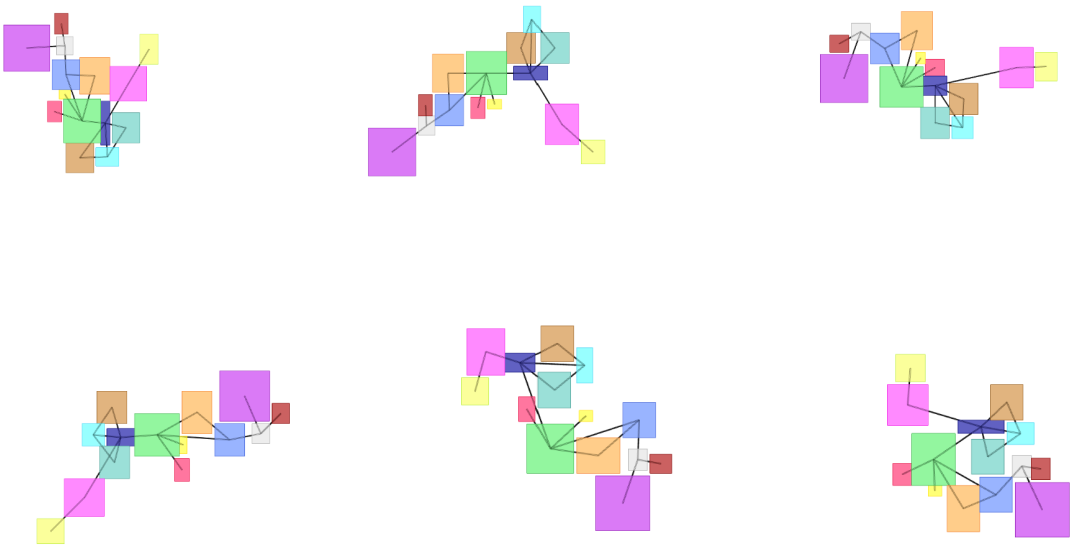

Figure 6. Converged Graphs for Sample Input

Finally, in 3D, each room_node's height property can be extruded to give architects a spatial sense of the program's mass. The program input can also be manipulated to include outdoor spaces such as courtyards with minimal height. Figure 7 illustrates a program that includes this type of outdoor space. Additionally, various program typologies can be entered as can be seen in Figure 8.

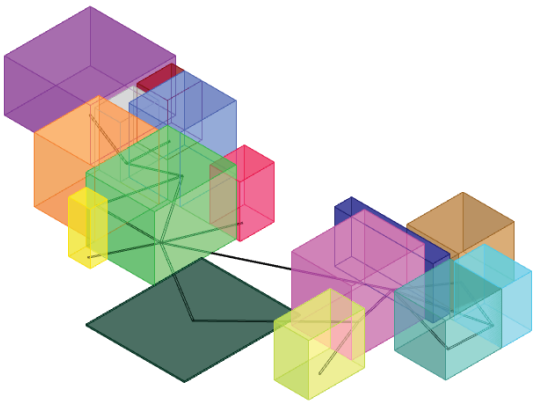

Figure 7. Axonometric Convergence of Extruded Spaces

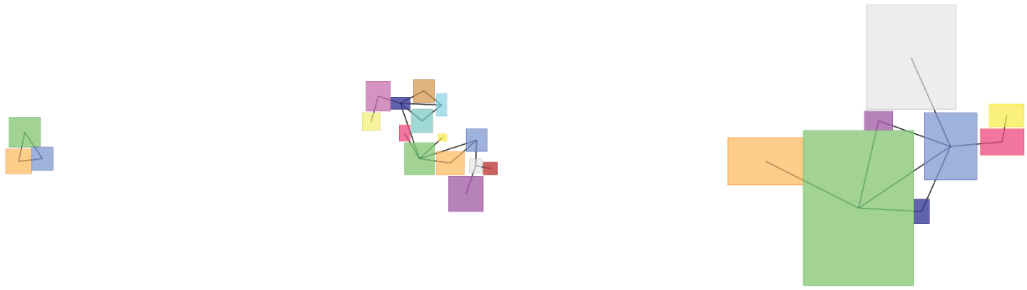

Figure 8. Scales of Inputs

In conclusion, this script offers architects and designers a faster and more efficient way to generate multiple conceptual floor plans. By iterating through various schemes, architects can gain a sense of the program relative to the size of its spaces and adjacencies. When a scheme shows potential, architects can save the layout and further develop it in their preferred style. By enabling architects to visualize these schemes quickly, they can spend more time designing and less time exhausting monotonous efforts. Whether an option is good or bad, architects can only know by seeing it. If only a handful of layouts are explored, architects may miss out on better design options. This tool empowers architects to explore a wider range of possibilities, leading to more informed design decisions and ultimately, better design outcomes.

Further examples of the workflow for a Recreation Center can be seen in the following Figures.

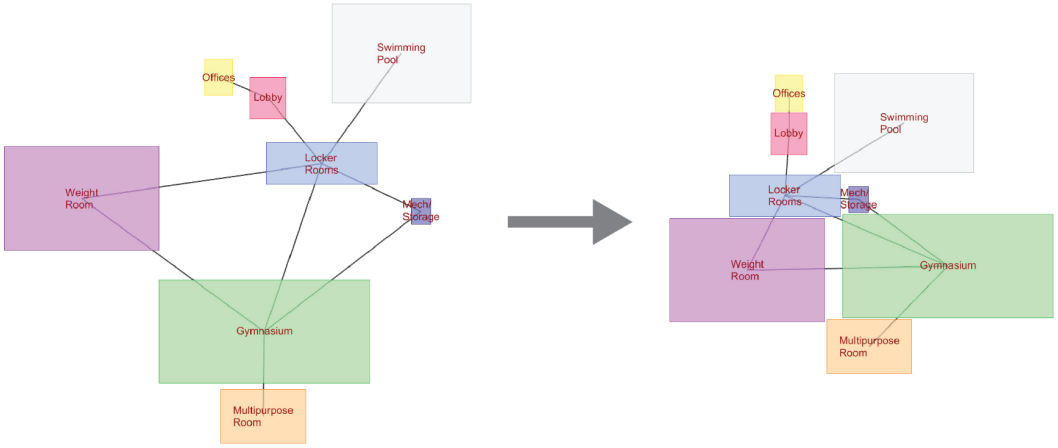

Figure 9. Convergence for Recreation Center Program

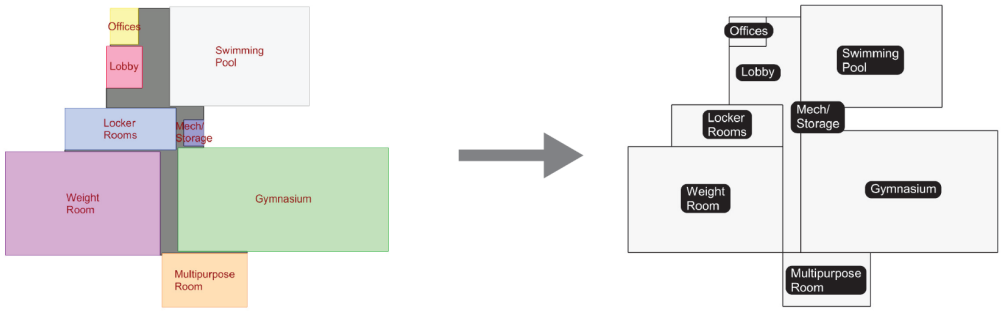

Figure 10. Plan Architectural Development

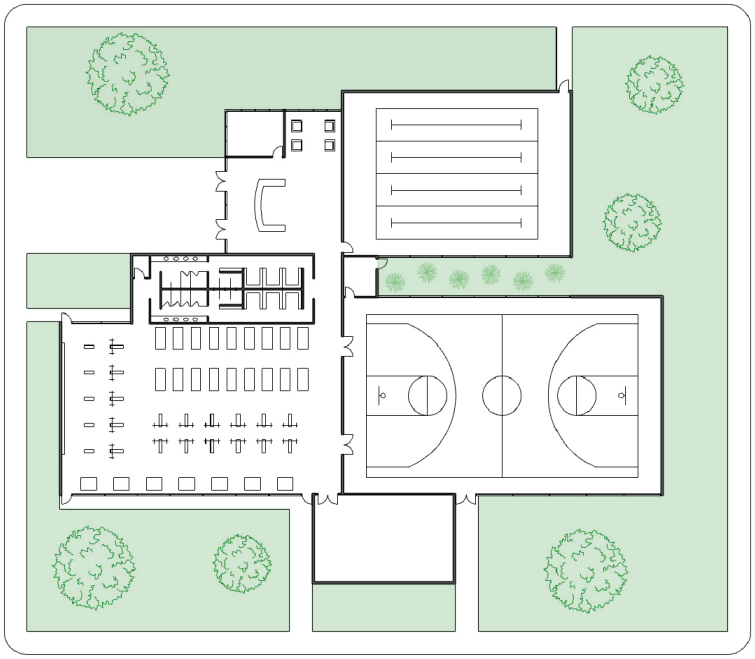

Figure 11. Further Developed Floor Plan

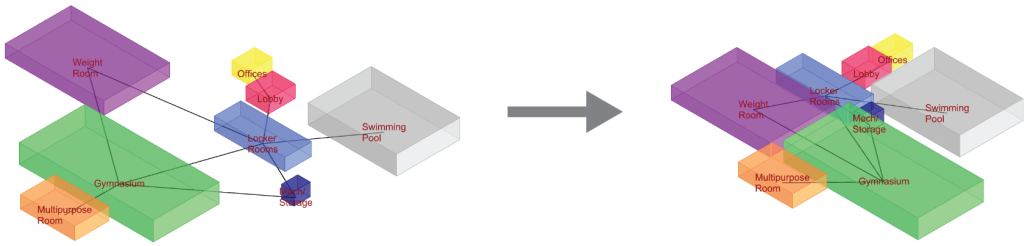

Figure 12. Axonometric Convergence for Recreation Center Program

Figure 13. Architectural Development in 3D

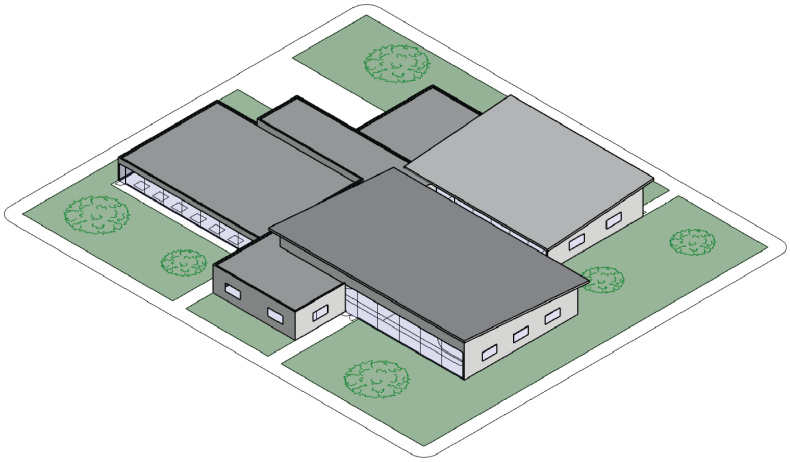

Figure 14. Further Developed 3D Model