Testing debugging guide:

Basic class layout-

Important classes:

Layout does all the optimization work

Skelton stores partial designs, and the info for optimizing them

Origami designer handles the gui interactions, and calls the optimize function of the layout.

Paper is where the information about a design is sorted.

Nodes are part of both paper and skeletons. They define the design itself.\, along with the lists of connections(stored in the paper class).

Detailed information

Layout class:

The constructor sets up the paper to be optimized, and the info needed. Do not touch this code, as it will break.

Optimize takes this paper, and tries to find the best size for the design. Right now, the setup is:

Take an upper bound found in the constructor methods, and a lower bound of 0.

Set the global size to 1+(upper-lower bound)/2.

Try to find a design that does not overlap, while only searching at sizes lower than the global size.

There are three possibilities:

1: the search finds a design, and ends the recursion. Great! Now this size is our new upper bound. The old design is saved in case that one turns out to be the optimized one. The system should print out “improved” + any debugging info. Note: this new upper bound is the size of the new design, not upper bound. Sometimes we get lucky and find something much smaller than the global size.

2: the recursion finishes without finding a viable design. Great! Now we know that size is t0o low, and we can use it as the new lower bound. The system should output “again improved”, since the bounds were improved.

3: we have to cutoff the search, to prevent it from taking years to finish. This is also used as a new lower bound, as it is to hard to find a design at this size, even if it exists. To combat this, commented out code searches again and again, changing the sequence the nodes are added in, this is a band-aid solution to a larger problem, and would be N times slower than a solution that does not need it.

Now that we have more info on the upper and lower bounds, we repeat the search.

When we have the best design (that we can find with this method), we will have a working design at some size, and have searched at 1- best size, and found nothing there (these can be searched in either order). At that point, we can set the position of the nodes in out paper to the best ones, and end the optimization. That is done in the get paper method.

How each search works:

The basic idea is a branch and bound depth first search. First, we generate the first node, at 0,0.

The nest node can be placed in a lot of places. That is done in the makePlaced method, which defines the rules that make this a mathematically functional origami design. This code will also break if modified. Make new Gen takes this list of viable places and makes new skeletons for each possibility.

The key function is optimizeDF. This takes a partial design, and finds all the places the next node could be added. makeNewGen takes care of the origami math, and only finds deigns smaller that the global size. Once a list of ways to add the new node is found, its time for recursion. This is where the code can be played around with again. Still testing new designs for rejecting branches that leed nowhere. This runs the risk of rejecting a good branch by accident though.