STEP 1: GeNEsIS_create

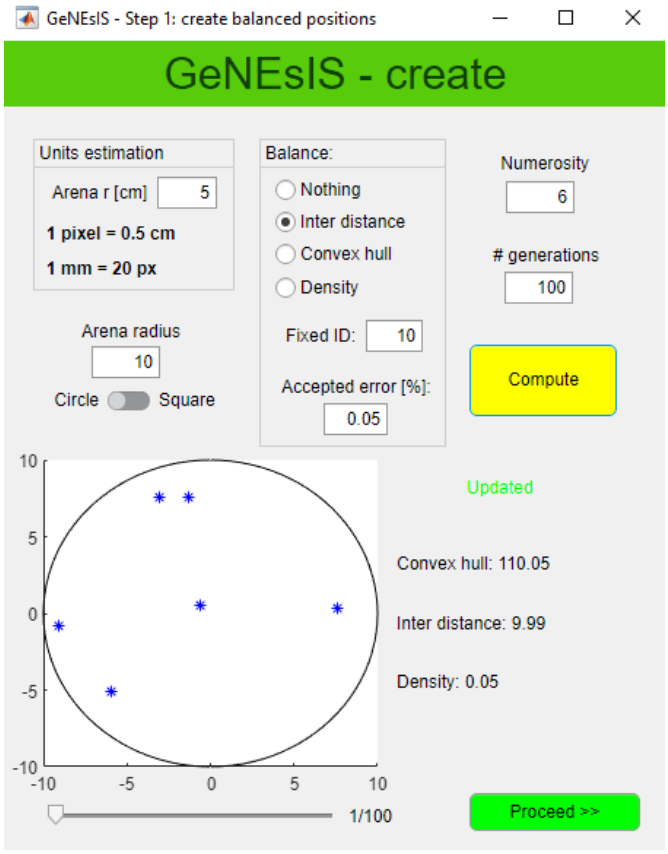
This step is divided in two parts:

A. create a ‘balanced’ disposition of your stimuli
B. create a ‘balanced’ shape of your stimuli

For this step you need to have both *GeNEsIS_create.mlapp* and *shapesGenesis.mlapp* in the same working directory of Matlab.

A. Create your balanced stimuli distribution in space

In this window you can create different combinations of points distributed in space with custom characteristics:



- *Arena radius*: radius of the area in which the points will be generated. Suggested value: 10 (see later for a more detailed discussion).
- *Numerosity*: number of elements (n)
- *# generations*: number of combinations that you want to create
- *Balance*: you can select '*Nothing*' for a random distribution of points, '*Inter distance*' to obtain a distribution with fixed ID, '*Convex hull*' for a fixed CH, '*Density*' for a fixed ratio between numerosity and area (D=n/CH). Choose the fixed value referred to the selected characteristic: all the different generations that will be created will have that fixed balanced characteristic.
- *Accepted error*: tolerance range in parameters estimation
- *Units estimation*: conversion tool pixel/centimeters

Press *'Compute'* in order to start the generation of points; this could take a long time (see later). Once finished an imaged will be displayed where you can visualize the different generations and, on the bottom, the values for the main characteristics of the stimulus are reported.

How to refer to dimensions

The program works in pixel. You will have the possibility to set your effective dimensions (in cm) in Step 2, for the moment don't worry about this and take as reference the arena radius to rescale all the other elements proportionally. '*Units estimation*' can help you.

(Notice that, if for example you enlarge your window, the figure dimension as visualized on screen will change, but the effective size in pixel unit is always the same).

For better performances, in general I suggest to keep the arena radius fixed at 10 pixels. Instead, adapt the arena radius to the number of elements, i.e. increase it if you need to create a large number of elements (more than 20). If you increase the arena size and you want a circular arena this could become incompatible; thus, for big arenas keep a square shape only.

How to properly choose parameter values

We have understood that it is important to set the proper parameters in order not to make the computation fail. In order to clarify which are acceptable values of the parameters, I suggest starting with a fast computation of points in a random way (selecting '*Nothing'* on '*Balance'*). Since this option has no constraints it is always very fast (even for large number of generations); still, as pointed out before, problems can arise if you increase the arena radius in the circular configuration.
You can now investigate the range of values the parameters take, looking at them on the right: it is a good parameter choice a number that is inside the range of values generated randomly.

Remember that you probably would like to balance the parameters also between different numerosity, so try to investigate the range of possible values for both numerosity and choose a good one, fitting both cases!

N.B. If you don't need a precision of 100% in your balanced parameter, you can introduce an accepted margin of error (expressed in percentage): in this way the software will accept also condition that are fulfilling your constraints ± error. Already an error of 0.1% will improve a lot the computational time!

The computation

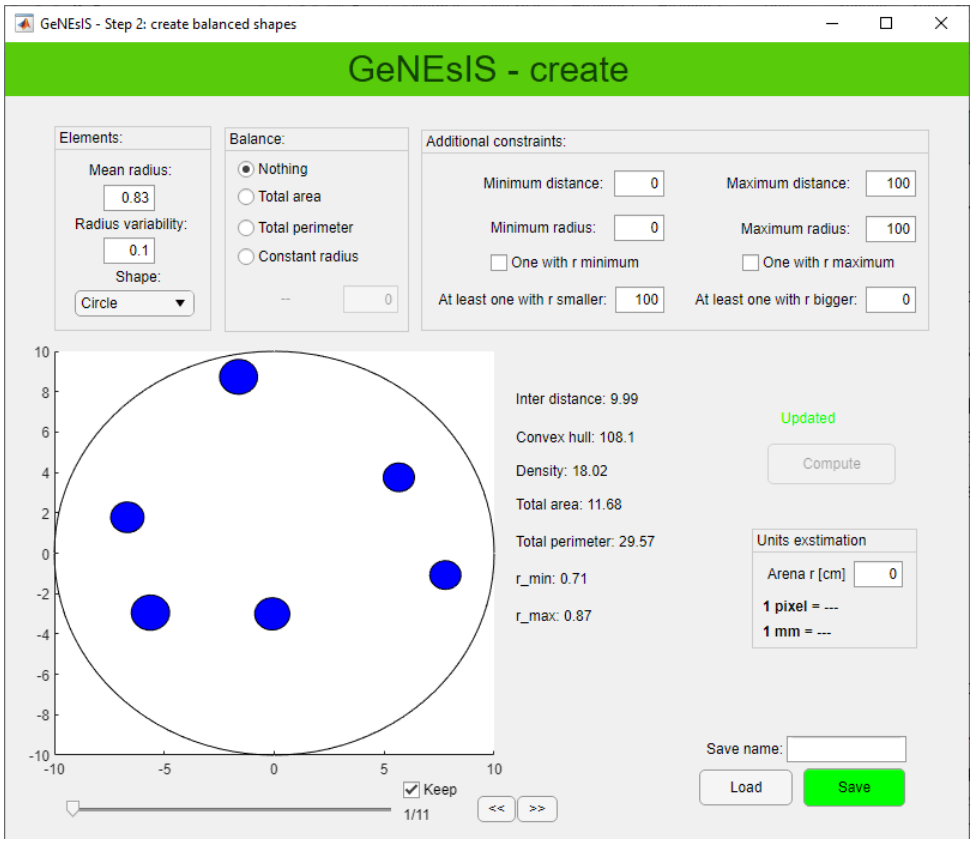
Every time you press '*Compute',* the program will create a set of generations with points randomly distributed according to the constraints you selected; precisely, it generates random points and it will keep them, only if they fulfill the selected characteristics. For this reason, the computation can take a long time (especially with circular arena).
It is important to notice that, if you select impossible conditions, the program will fall in an endless loop, generating always new random points and excluding them since not as desired; if you think this is the case you could have to stop the program (killing it, CtRL+C).

For example, if you have an arena radius of 10 pixel and you start a computation setting an ID of 30, clearly any combination will be never found and the program will be stacked in an endless loop. It will abort after some minutes.

Once you have found the proper values to create your balanced disposition of points you can compute the final dataset. The logic of the program in the following steps is to take these points and construct the elements with defined areas and perimeters around them. Since many different constraints can be considered in the following, most of the generations that you are creating in this first step will be discarded later: for this reason, it is important to generate a high number of generations. The final number of good figures you will end with depends on many factors, thus it is difficult to estimate the number of generations to compute, but it could be useful to set it at least around 1000 or more. Clearly it could take a long time to compute all the generations, but this is the only long-lasting part. If you are sure the computation proceeds (meaning it is not stacked at 0%) you can simply let the program run and compute a big number of data.

Once the program has finished you can proceed on next step, pushing on '*Proceed*'.

B. Create your balanced stimuli shape



In this second window you can set all your geometrical shape characteristics. The program generates the shapes around the previously computed points, and these shapes have dimensions distributed normally around a mean value *Mean radius* and a variability given by *Radius variability*. The bigger the variability is, the more different the dimension of the elements will be (0.1 for elements all similar in dimensions).

N.B. If you choose a different shape than the circle, the value *radius* will refer to the characteristic size of the selected shape, more precisely to the value of the side.

Constraints

You can choose different constraints: in the *Balance* tab you can set the total area of your stimuli, the total perimeter or choose a constant radius for all the elements. In the *Additional constraints* tab, you can set minimum and maximum distance between two elements; set the minimum or maximum radius the elements could have, and if there must be an element with that precise min/max radius (*'One with r minimum/maximum'*); set if you want at least one element smaller/bigger than a threshold size (*'At least one with r smaller/bigger'*).

The program generates all the shapes and then it will discard the combinations overlapping, going outside the arena or not fulfilling your constraints: for this reason, in this passage you will lose most of the generations created before. This is way it is important to have a big starting dataset from the previous point.

N.B. Every time you perform a new computation the program recreates the shapes randomly, thus, even if you set exactly the same parameters you won't never end with the same final stimuli!

If you obtain configurations that you don't like visually, you can still remove them by unselecting the '*Keep*' field.

Once you have generated all your stimuli and you have finished, you can save a Matlab file with all the generations created. At the same time an excel file with all your parameters for each generation is automatically saved too. You can also load a previously saved file and modify it.

Good practice: I suggest to save the file with a name that indicates the parameters of interest.

For example if you have balanced total area at 7 and convex hull at 30 for three elements you can save something like 'n3_TA7_CH30'.

Your stimuli are now defined, you can close the program and pass to step 2 for the appearance and effective dimension adjustments.