Touch and stop particle packing in R

This work is an evolution of my previous work available on Figshare here and here. The previous code was in MATLAB language; I have now optimised my original code and converted it to R, so it is more broadly available but also better documented.

My original MATLAB code underpins a few academic articles, which you can find below. I am sure that the same or similar principles can be used in other fields of research, however!

- Generation of virtual asphalt mixture porosity for computational modelling
- Stochastic generation of virtual air pores in granular materials
- Generation Of 3D Soil/Asphalt Porosity Patterns For Numerical Modelling

The code seeks to fill up a rectangle with growing circles (also called "particles" in the code), following a "touch-and-stop" model. In practice, circles are seeded with a very small size and then grow until they touch another circle or the boundary of the rectangle. The code works to meet a target planar void ratio, defined as follows: 1-(area covered by circles)/(area of the rectangle)

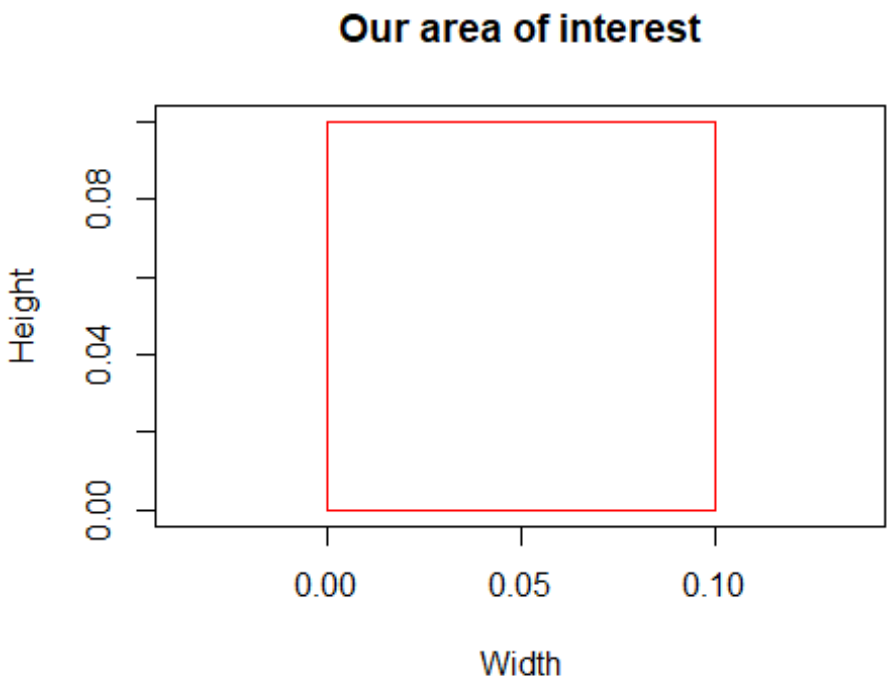
The code starts by setting this target planar void ratio, which can be any value you like. The smaller the value, the longer it will take the code to meet this requirement.

```
# Planar void ratio as a %
target_planar_void_ratio <- 40
```

At this stage, you need to pick a few parameters that define the area where the circles grow. At present, this is set to be a square with a square with a side 0.1 m in length.

```
# All sizes in metres (m)
origin_of_cartesian_axes <- 0 # This is just for reference

# Width of the rectangle
rectangle_width <- 0.100

#Height of the rectangle
rectangle_height <- 0.100
```

Let's plot the area of interest:



Now, you need to specify some parameters about the particles that will grow in this area of interest (measures in metres). All these values are arbitrary, but will of course highly affect your results.

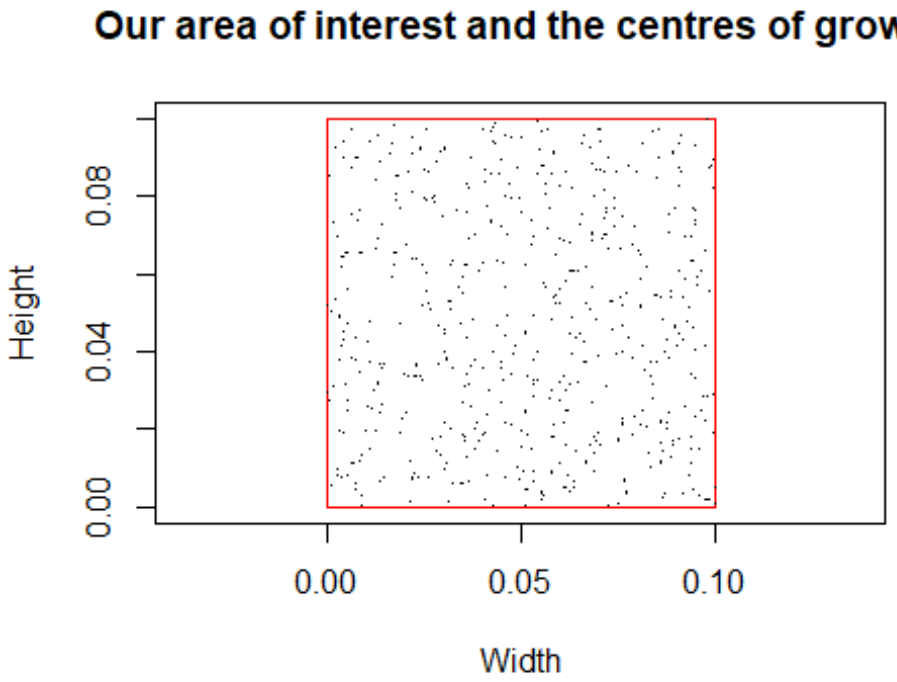
```
# Number of particles in the first generation seeded
number_of_particles <- 500

# Maximum size of a particle (m)
maximum_radius <- 0.07

# Initial size of the particles in metres (m)
minimum_radius_seeded <- 0.1e-4
```

The input stage is now over, and there are no other variables you need to specify. It is time to initiate the first generation of circles that we wish to grow. Note that this is all implemented as matrices or vectors, so that the calculations later on are more efficient. The variables "centre" (n-by-2 matrix) and "radius" (column vector) include the x and y coordinates of the circles and their radii, respectively. They are first initialised as zeros and then populated using random positions in the area of interest. Note that this code is using a uniform distribution ("runif"), but you could use any other approach to randomisation

```
# Initialisation of variables
centre <- cbind(rep(0, number_of_particles), rep(0, number_of_particles))
radius <- c(rep(0, number_of_particles))
```

```
# Generation of a vector of radii, one row for each particle
radius <- minimum_radius_seeded*c(rep(1, number_of_particles))

# Generation of the centres with random positions, one row for each particle
centre[,1] <- minimum_radius_seeded+(rectangle_width-
2*minimum_radius_seeded)*runif(number_of_particles)
centre[,2] <- minimum_radius_seeded+(rectangle_height-
2*minimum_radius_seeded)*runif(number_of_particles)
```

Let's visualise these circles that, effectively, are centres of growth.



The process continues by starting to grow each of these circles, one by one. This is achieved via a while loop, that applies six condition to verify if a circle can grow:

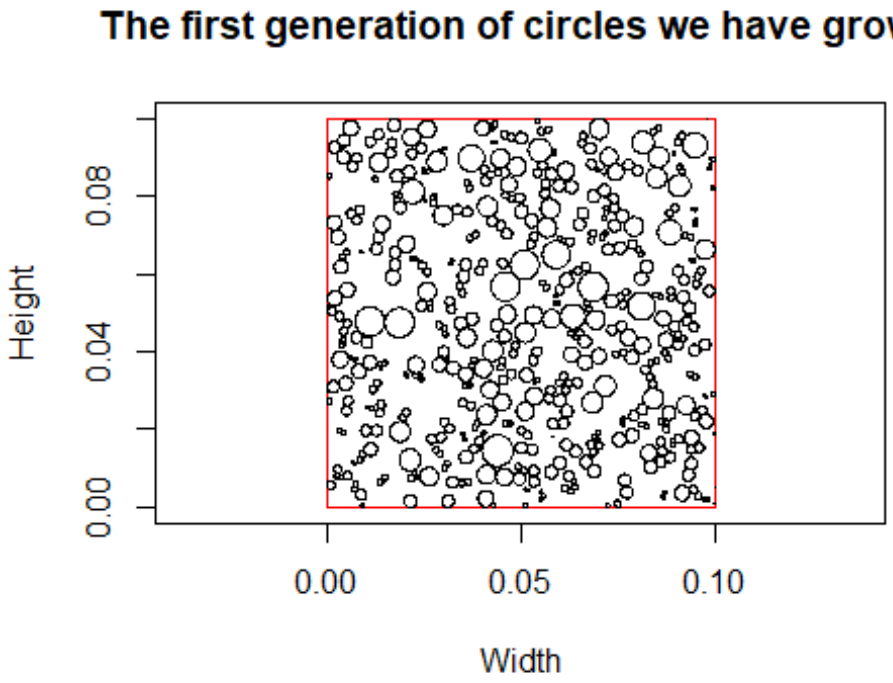
- Condition 1: This checks if the distance between particle k and every other particle is smaller than the sum of their radii plus the minimum radius growth allowed (i.e. "minimum_radius_seeded"). This should return 0 when particle k is compared to all other particles and 1 when particle k is compared to itself (the comparison to itself occurs because we are using vectors of centres and radii, and particle k is obviously included in these). Therefore, condition 1 must return 1 for the particle to be eligible to grow.
- Conditions 2-5: If the radius grows by 'minimum_radius_seeded', will it still be inside the area of interest defined above?
- Condition 6: If the radius grows by 'minimum_radius_seeded', will it still be smaller than the maximum radius allowed?

If a circle satisfies all these conditions, then it is allowed to grow. Once all circles have been checked, the current void ratio is compared to the target void ratio. If the current void ratio is found to be lower than the target, then the algorithm is stopped (see "break").

```
# Beginning of the growth process
iteration <- 2
planar_void_ratio <- c(rep(100,20000))
planar_void_ratio[1] <- c(101)

while (planar_void_ratio[iteration] != planar_void_ratio[iteration-1] &&
planar_void_ratio[iteration]>target_planar_void_ratio){
  iteration=iteration+1;

  for (k in 1:number_of_particles){
    condition <- rep(13, 6)

    condition[1] <- sum(sqrt((centre[,1]-centre[k,1])^2+(centre[,2]-
centre[k,2])^2)<(radius[k]+minimum_radius_seeded+radius))
    condition[2] <-
(centre[k,1]+radius[k]+minimum_radius_seeded<rectangle_width)
    condition[3] <- (centre[k,1]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[4] <-
(centre[k,2]+radius[k]+minimum_radius_seeded<rectangle_height)
    condition[5] <- (centre[k,2]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[6] <- (radius[k]+minimum_radius_seeded<=maximum_radius)

    # Logical condition: if all the conditions above are met, the particle k
is allowed to grow
    can_it_grow <- condition[1]==1 && condition[2]==1 && condition[3]==1 &&
condition[4]==1 && condition[5]==1 && condition[6]==TRUE

    # The radius of particle k is increased by "minimum_radius_seeded"
    if (can_it_grow==TRUE){
      radius[k] <- radius[k]+minimum_radius_seeded
    }

  }

  # The planar void ratio is calculated for the current iteration
  squared_sum_of_radii <- sum(radius[1:number_of_particles]^2)
  planar_void_ratio[iteration] <- (rectangle_width*rectangle_height-
pi*squared_sum_of_radii)/(rectangle_width*rectangle_height)*100

  if (planar_void_ratio[iteration] < target_planar_void_ratio){
    #% If the planar void ratio is lower than the target, the calculation
must stop
    break
```

```
  }


}
```

Let's see what things look like after the first generation of circles has been grown:



At this stage, none of the circles we have seeded originally can continue growing, as all six of the above conditions are returning 0. The only way to continue filling the space is to generate new circles: to achieve this, it is essential to make sure that the new centres do not fall inside any of the existing circles.

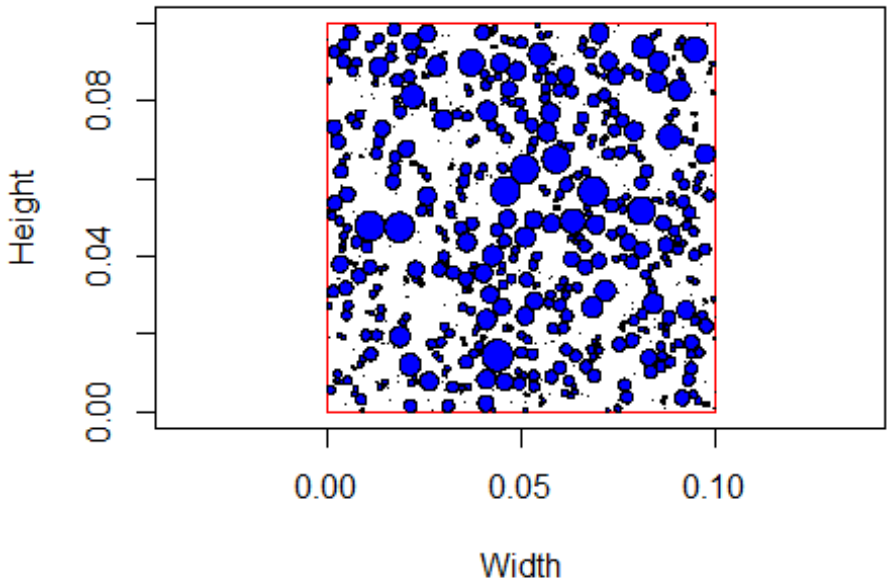
The process to generate new centres is based on the same six conditions noted above. The code seeks to create a chosen number of new centres of growth (see the variable "new_centres_desired"); at present, the code is set to generate more centres the lower the planar void ratio. You can delete or edit this condition as you like.

Candidate centres and radii are created randomly and added to the vectors "centre" and "radius". Then, the new circle is tested based on the six conditions: if it meets them, it is kept in the variables "centre" and "radius". If it doesn't, it is deleted. This process seeks to ensure that only valid candidates are added to the variables "centre" and "radius". This while loop continues until the number of new circles generated is equal to "new_centres_desired".

```
# Generate new acceptable centres

starting_particles_before_new_generation <- dim(centre)[1]

start_generating_new_centres <- 1

while (start_generating_new_centres==1){

if(planar_void_ratio[iteration]>45){
  new_centres_desired <- 200
}
if(planar_void_ratio[iteration]<45){
  new_centres_desired <- 400
}
if(planar_void_ratio[iteration]<35){
    new_centres_desired <- 600
}

new_centres_accepted <- 0

while (new_centres_accepted<new_centres_desired){
  candidatex <- minimum_radius_seeded+(rectangle_width-
2*minimum_radius_seeded)*runif(1)
  candidatey <- minimum_radius_seeded+(rectangle_height-
2*minimum_radius_seeded)*runif(1)
  candidaterad <- minimum_radius_seeded
  candidate_centre <- cbind(candidatex, candidatey)
  centre <- rbind(centre, candidate_centre) # We hypothetically add the
candidate centre to the "centre" matrix. This allows us to use vector
operations in the conditions below
  radius <- c(radius, candidaterad)

  condition <- rep(13, 3)

  k = dim(centre)[1]

  condition[1] <- sum((sqrt((centre[,1]-
centre[nrow(centre),1])^2+(centre[,2]-
centre[nrow(centre),2])^2))<(candidaterad+minimum_radius_seeded+radius))
  condition[2] <-
(candidatex+candidaterad+minimum_radius_seeded<rectangle_width)
  condition[3] <- (candidatex-candidaterad-
minimum_radius_seeded>origin_of_cartesian_axes)
  condition[4] <-
(candidatey+candidaterad+minimum_radius_seeded<rectangle_height)
  condition[5] <- (candidatey-candidaterad-
minimum_radius_seeded>origin_of_cartesian_axes)
  condition[6] <- (candidaterad+minimum_radius_seeded<=maximum_radius)
```

```
  is_it_acceptable <- condition[1]==1 && condition[2]==1 && condition[3]==1
&& condition[4]==1 && condition[5]==1 && condition[6]==TRUE

  if (is_it_acceptable==TRUE){
    new_centres_accepted <- new_centres_accepted + 1
    }else{
    centre<-(centre[-nrow(centre),]) # If the centre is not acceptable based
on the conditions set, we remove the last row of the "centre" matrix, where
it was stored hypothetically
    radius <- radius[1:(length(radius)-1)] # The same is true for any
unwanted radius
    }

  }

start_generating_new_centres <- 0 # Stop generating new centres, as we have
reached the number desired

}
```

This is what things look like with a fully grown first generation and seeds for the next generation:

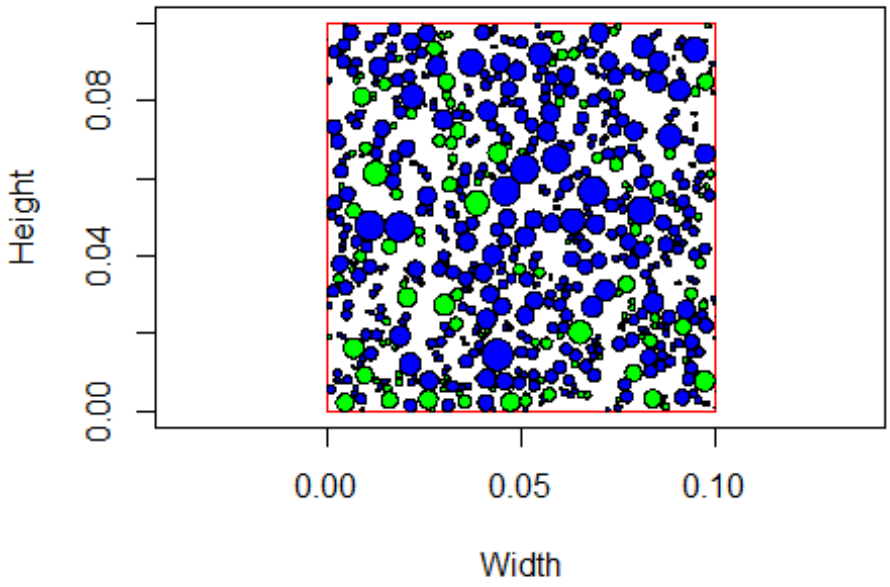


The obvious next step is to grow the new circles. This is the exact same process as above:

```
# Grow the new centres

iteration <- iteration+1
planar_void_ratio[iteration] <- 100 #Resetting this for the next calculation

while (planar_void_ratio[iteration] > target_planar_void_ratio &&
```

```
start_generating_new_centres==0){

  iteration <- iteration+1

  for (k in (starting_particles_before_new_generation+1):dim(centre)[1]){
    condition <- rep(13, 6)
    condition[1] <- sum(sqrt((centre[,1]-centre[k,1])^2+(centre[,2]-
centre[k,2])^2)<(radius[k]+minimum_radius_seeded+radius))
    condition[2] <-
(centre[k,1]+radius[k]+minimum_radius_seeded<rectangle_width)
    condition[3] <- (centre[k,1]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[4] <-
(centre[k,2]+radius[k]+minimum_radius_seeded<rectangle_height)
    condition[5] <- (centre[k,2]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[6] <- (radius[k]+minimum_radius_seeded<=maximum_radius)
    can_it_grow <- condition[1]==1 && condition[2]==1 && condition[3]==1 &&
condition[4]==1 && condition[5]==1 && condition[6]==TRUE
    if (can_it_grow==TRUE){
      radius[k] <- radius[k]+minimum_radius_seeded
    }
  }

  squared_sum_of_radii <- sum(radius[1:dim(centre)[1]]^2)
  planar_void_ratio[iteration] <- (rectangle_width*rectangle_height-
pi*squared_sum_of_radii)/(rectangle_width*rectangle_height)*100

  if (planar_void_ratio[iteration] < target_planar_void_ratio){
    start_generating_new_centres <- 0
    break
  }

  if(planar_void_ratio[iteration] == planar_void_ratio[iteration-1]){
    start_generating_new_centres <- 1
  }

}
```

Let's look at the circles once again:



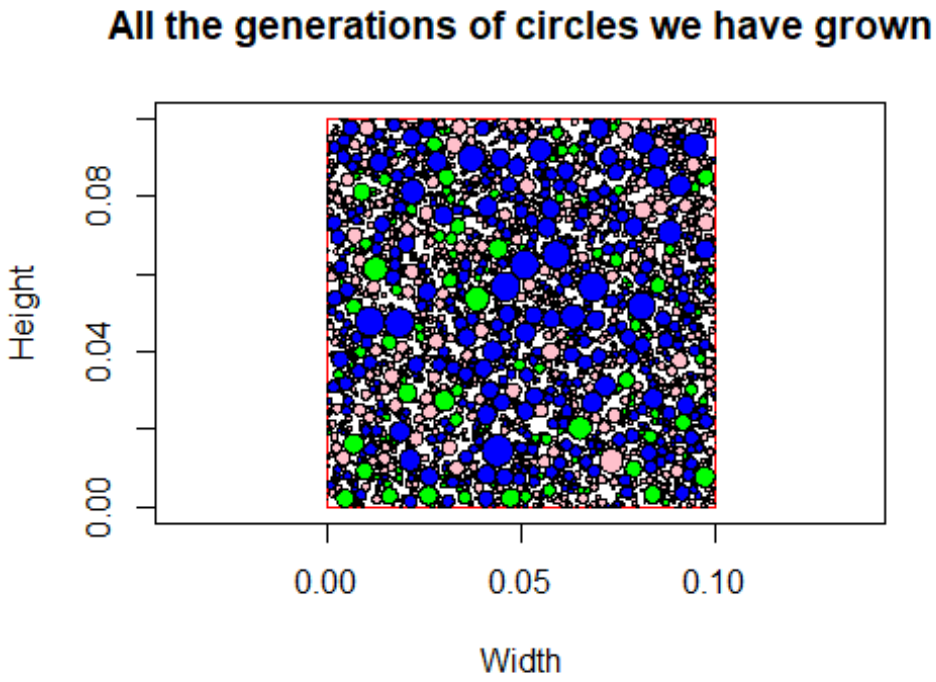
The two above steps can easily be automated. In this case, I have shown them separately for the sake of explaining the code. However, new generations can be seeded and grown automatically, as follows:

```
# Generate new acceptable centres
starting_particles_before_automated_loop <- dim(centre)[1]

start_generating_new_centres <- 1

while (start_generating_new_centres==1){

if(planar_void_ratio[iteration]>45){
  new_centres_desired <- 200
}
if(planar_void_ratio[iteration]<45){
  new_centres_desired <- 400
}
if(planar_void_ratio[iteration]<35){
    new_centres_desired <- 600
}

new_centres_accepted <- 0

while (new_centres_accepted<=new_centres_desired){
  candidatex <- minimum_radius_seeded+(rectangle_width-
2*minimum_radius_seeded)*runif(1)
  candidatey <- minimum_radius_seeded+(rectangle_height-
2*minimum_radius_seeded)*runif(1)
  candidaterad <- minimum_radius_seeded
  candidate_centre <- cbind(candidatex, candidatey)
```

```
  centre <- rbind(centre, candidate_centre)
  radius <- c(radius, candidaterad)

  condition <- rep(13, 3)

  k = dim(centre)[1]

  condition[1] <- sum((sqrt((centre[,1]-
centre[nrow(centre),1])^2+(centre[,2]-
centre[nrow(centre),2])^2))<(candidaterad+minimum_radius_seeded+radius))
  condition[2] <-
(candidatex+candidaterad+minimum_radius_seeded<rectangle_width)
  condition[3] <- (candidatex-candidaterad-
minimum_radius_seeded>origin_of_cartesian_axes)
  condition[4] <-
(candidatey+candidaterad+minimum_radius_seeded<rectangle_height)
  condition[5] <- (candidatey-candidaterad-
minimum_radius_seeded>origin_of_cartesian_axes)
  condition[6] <- (candidaterad+minimum_radius_seeded<=maximum_radius)

  is_it_acceptable <- condition[1]==1 && condition[2]==1 && condition[3]==1
&& condition[4]==1 && condition[5]==1 && condition[6]==TRUE

  if (is_it_acceptable==TRUE){
    new_centres_accepted <- new_centres_accepted + 1
    }else{
    centre<-(centre[-nrow(centre),])
    radius <- radius[1:(length(radius)-1)]
    }

  }

start_generating_new_centres <- 0 #
particles_to_grow <- dim(centre)[1]-starting_particles_before_automated_loop

# Grow the new centres

iteration <- iteration+1
planar_void_ratio[iteration] <- 100 #Resetting this for the next calculation

while (planar_void_ratio[iteration] > target_planar_void_ratio &&
start_generating_new_centres==0){

  iteration <- iteration+1

  for (k in (particles_to_grow+1):dim(centre)[1]){
    condition <- rep(13, 6)
    condition[1] <- sum(sqrt((centre[,1]-centre[k,1])^2+(centre[,2]-
centre[k,2])^2)<(radius[k]+minimum_radius_seeded+radius))
```

```
    condition[2] <-
(centre[k,1]+radius[k]+minimum_radius_seeded<rectangle_width)
    condition[3] <- (centre[k,1]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[4] <-
(centre[k,2]+radius[k]+minimum_radius_seeded<rectangle_height)
    condition[5] <- (centre[k,2]-radius[k]-
minimum_radius_seeded>origin_of_cartesian_axes)
    condition[6] <- (radius[k]+minimum_radius_seeded<=maximum_radius)
    can_it_grow <- condition[1]==1 && condition[2]==1 && condition[3]==1 &&
condition[4]==1 && condition[5]==1 && condition[6]==TRUE
    if (can_it_grow==TRUE){
      radius[k] <- radius[k]+minimum_radius_seeded
    }
  }

  squared_sum_of_radii <- sum(radius[1:dim(centre)[1]]^2)
  planar_void_ratio[iteration] <- (rectangle_width*rectangle_height-
pi*squared_sum_of_radii)/(rectangle_width*rectangle_height)*100

  if (planar_void_ratio[iteration] < target_planar_void_ratio){
    start_generating_new_centres <- 0
    break
  }

  if(planar_void_ratio[iteration] == planar_void_ratio[iteration-1]){
    start_generating_new_centres <- 1
  }

  }

}
```

Let's visualise all the other generations of circles grown to meet the "target_planar_void_ratio" we have defined at the top:



The code is now complete, and we have successfully grown a number of circles in the area of interest selected. Here's a summary of what we have achieved:

```
print(paste("We have created and grown", dim(centre)[1], "circles."))

## [1] "We have created and grown 2507 circles."

print(paste("The first generation (blue) included", number_of_particles,
"circles."))

## [1] "The first generation (blue) included 500 circles."

print(paste("The second generation (green) included",
(starting_particles_before_automated_loop-
starting_particles_before_new_generation), "circles."))

## [1] "The second generation (green) included 200 circles."

print(paste("We have achieved a void % (i.e. the white space) of:",
min(planar_void_ratio)))

## [1] "We have achieved a void % (i.e. the white space) of:
39.9823762025117"
```

If you run this code on your machine, the results will always differ - and that's exactly what I originally set out to achieve. Please note that I am not a professional programmer, so this

code may not be perfect or might be improved. As you can see, it works well as things stand, but do get in touch if you have any suggestions.